

Júlio Battisti

**Aprovado em 30 Exames de
Certificação Microsoft:**

MCP 2000 e 2003, MCP+I, MCSE +I,
MCSE 2000, MCSE 2003, MCSA 2000,
MCSA 2003, MCSD, MCDBA e MCDST

ASP.NET

**Uma Revolução na
Construção de Sites e
Aplicações Web**

Site: www.juliobattisti.com.br

e-mail: webmaster@juliobattisti.com.br

Nota sobre direitos autorais:

Este ebook é de autoria de Júlio Battisti, sendo comercializado diretamente através do site www.juliobattisti.com.br ou através do site de leilões Mercado Livre: www.mercadolivre.com.br, pelo usuário GROZA. **Nenhum outro usuário, pessoa ou site está autorizado a vender este ebook.**

Ao adquirir este ebook você tem o direito de lê-lo na tela do seu computador e de imprimir quantas cópias desejar. É vedada a distribuição deste arquivo, mediante cópia ou qualquer outro meio de reprodução, para outras pessoas. **Se você recebeu este ebook através do e-mail ou via ftp de algum site da Internet, ou através de um CD de Revista, saiba que você está com uma cópia pirata, não autorizada. A utilização de uma cópia pirata, não autorizada, é crime de Violação de Direitos Autorais, sujeita a pena de Cadeia.**

O valor cobrado por este arquivo é praticamente simbólico, pelas horas e horas de trabalho que ele representa. Novos e-books somente poderão ser desenvolvidos pela honestidade de pessoas que adquirem o arquivo do e-book e não o distribuem livremente para outras pessoas. Se você recebeu uma cópia deste arquivo sem tê-la adquirido diretamente com o autor, seja honesto, entre em contato com o autor, através do e-mail webmaster@juliobattisti.com.br, para regularizar esta cópia.

Ao regularizar a sua cópia você estará remunerando, mediante uma pequena quantia, o trabalho do autor e incentivando que novos trabalhos sejam disponibilizados.

Se você tiver sugestões sobre novos cursos que gostaria de ver disponibilizados, entre em contato pelo e-mail: webmaster@juliobattisti.com.br.

Visite periodicamente o site www.juliobattisti.com.br para ficar por dentro das novidades:

- ◆ Cursos de informática.
- ◆ Artigos e dicas sobre Certificações da Microsoft.
- ◆ Artigos sobre Carreira e Trabalho.
- ◆ Dicas de livros e sites sobre diversos assuntos.
- ◆ Simulados gratuitos, em português, para os exames da Microsoft.

PIRATARIA É CRIME, COM PENA DE CADEIA. EU AGRADEÇO PELA SUA HONESTIDADE. SE VOCÊ COMPROU UMA CÓPIA DESTE CURSO, DIRETAMENTE COM O AUTOR, NÃO DISTRIBUA CÓPIAS PARA OUTRAS PESSOAS. SE VOCÊ RECEBEU UMA CÓPIA ILEGAL DESTE ARQUIVO, NÃO ADQUIRIDA DIRETAMENTE COM O AUTOR JÚLIO BATTISTI, ENTRE EM CONTATO E REGULARIZE A SUA CÓPIA.

Agradecimentos

Escrever sobre um assunto novo e empolgante como ASP.NET é, ao mesmo tempo, estimulante e desafiador. Estimulante porque representa uma mudança de paradigma e desafiador porque explicar de uma maneira clara e objetiva um assunto como ASP.NET é uma tarefa árdua.

Aos heróis, muitas vezes anônimos, que se lançaram na aventura de entender o ASP.NET e, principalmente, divulgar suas descobertas e vitórias, em inúmeros sites gratuitos e listas de discussão da Internet, ficam os meus sinceros agradecimentos. O espírito de colaboração e coleguismo destas pessoas é realmente admirável.

À minha esposa Lu, pelo carinho, amor, dedicação, companheirismo e tolerância.

À dona Lucy, minha mãe, por sempre me apoiar e ser uma grande admiradora e incentivadora de tudo o que faço.

A meu Pai, em memória, pelo jeito simples e pacato, que me ensinou a parar e refletir nos momentos difíceis. Aos meus irmãos agradeço pelos bons momentos que juntos passamos.

Aos amigos de Boqueirão do Leão, pelas poucas horas de lazer e diversão que pude usufruir neste ano de tanto trabalho. A vocês gostaria de dizer que sinto sinceras saudades da companhia de todos.

À equipe editorial da Axcel por acreditar e produzir mais um trabalho de minha autoria. Também não posso deixar de agradecer aos demais membros da Axcel Books, pela sua paciência em produzir, corrigir e revisar mais este trabalho, sempre com sugestões para a melhoria do mesmo.

A Deus por nos dar a inteligência e a determinação na busca de cada vez fazer as coisas de uma maneira melhor e mais simples.

Sobre o Autor

Júlio Battisti é profissional certificado da Microsoft, tendo sido aprovado em 21 Exames da Microsoft, com os quais obteve certificações como: MCP, MCP+I, MCSE 2000, MCSE+I, MCDBA 2000 e MCSD. Trabalha como Gerente de rede na Secretaria da Receita Federal, e trabalha com o Framework .NET e o ASP.NET desde a versão Beta 1 do Framework .NET, lançada em Junho de 2000. Também é autor de artigos de informática e trabalha com o desenvolvimento e administração de Web Sites. Atua como instrutor de cursos de informática tanto na Secretaria da Receita Federal como para turmas em Universidades e outros cursos.

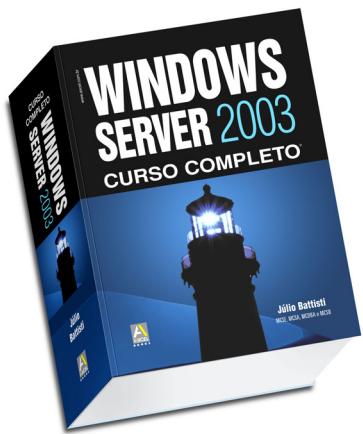
Conheça Outros Livros do Autor Júlio Battisti



Manual de Estudos Para o Exame 70-217 – 752 páginas

Chega ao mercado editorial mais um aguardado lançamento da Axcel Books Editora - Certificação Microsoft - Guia de Estudos Para o MCSE - Exame 70-217, onde o autor Júlio Battisti descreve, de forma detalhada e com exemplos passo-a-passo, todos os tópicos que fazem parte do programa oficial da Microsoft para o exame de certificação 70-217. A obra apresenta e explica desde os princípios básicos, incluindo os fundamentos do Active Directory; passando por serviços tais como DNS, gerenciamento de compartilhamentos, Master Operations, permissões NTFS, Grupos de Usuários, Unidades Organizacionais e Group Policy Objects, os GPOs; além de ainda tratar de questões como a configuração de Auditoria de Objetos, o gerenciamento do Schema, entre outros.

Um curso completo de Active Directory para o Windows 2000 Server



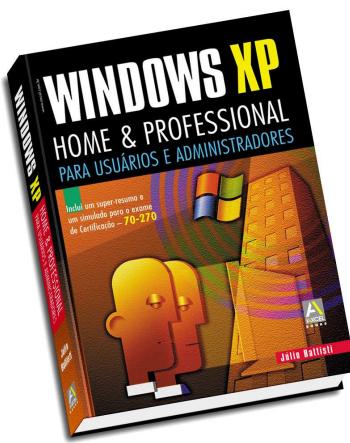
Windows Server 2003 – Curso Completo – 1568 páginas

O livro ensina desde os fundamentos básicos do Active Directory, passando pela instalação do Windows Server 2003 e por dicas sobre o projeto, implementação e migração do Windows 2000 Server para o Windows Server 2003. Você aprenderá, em detalhes, sobre os serviços de compartilhamento de arquivos e impressoras, segurança, como tornar o Windows Server 2003 um servidor Web, aprenderá sobre os serviços de rede: DNS, DHCP, WINS, RRAS, IPSec, Análise de Segurança, Group Policy Objects e muito mais. Confira, vale a pena.



Manual de Estudos Para o Exame 70-216 – 712 páginas

Neste aguardado lançamento da Axcel Books Editora - Certificação Microsoft - Guia de Estudos Para o MCSE - Exame 70-216, o autor Júlio Battisti descreve, de forma detalhada e com exemplos passo-a-passo, todos os tópicos que fazem parte do programa oficial da Microsoft para o exame de certificação. A obra apresenta e explica desde os princípios básicos, incluindo os fundamentos do protocolo TCP/IP; passando por instalação, configuração e administração do DNS, DHCP, WINS e RRAS; além de ainda tratar de questões quanto ao roteamento, NAT, Certificados Digitais, IPSec, entre outros.



Windows XP Home & Professional – 840 páginas

O novo mundo do Windows XP, que representa a nova era do sistema operacional para usuários e administradores está reunido nesta obra. Júlio Battisti apresenta a nova interface do sistema, completamente redesenhada e com a experiência de um profissional certificado da Microsoft. Na obra, os leitores irão aprender a implementar, configurar e utilizar o Windows XP, desvendando as funcionalidades, além das configurações de segurança, de desempenho e de estabilidade do sistema. O livro aborda ainda toda a parte de Internet do Windows XP - conectando e usando a Internet; configurando o firewall de conexão; além dos novos recursos do correio eletrônico. Veja também os detalhes sobre o Active Directory, as configurações de rede e protocolo TCP/IP, criptografia, registry do Windows, entre tantos outros assuntos. O leitor ainda vai poder contar com um capítulo exclusivo e um simulado com 100 questões/respostas destinados aos interessados no exame de Certificação 70-270 da Microsoft.



Manual de Estudos Para o Exame 70-290 – 1021 páginas

Um Manual de Estudos Completo para o **Exame 70-290: Gerenciando e Mantendo um Ambiente Baseado no Windows 2000 Server**. (Managing and Maintaining a Microsoft Windows Server 2003 Environment). Este é o exame de Administração do Windows 2000 Server, equivalente ao Exame 70-215 para o Windows 2000 Server. O Exame 70-290 é um exame obrigatório para o candidato que deseja obter a certificação MCSE-2003. **UM GUIA INDISPENSÁVEL PARA QUEM QUER PASSAR NO EXAME 70-290** Este manual apresenta as seguintes características: Curso em Português, baseado no Windows Server 2003 em Português, Completo, abrangente, abordando todos os tópicos do programa oficial, Repleto de exemplos práticos, passo-a-passo, detalhadamente explicados, Simulado com 60 questões, Questões com respostas e comentários detalhados.



SQL Server 2000: Administração & Desenvolvimento – Curso Completo - 816 páginas

O lançamento é destinado aos usuários/leitores da versão anterior do SQL Server, o SQL 7, além de redes de computadores em geral, Windows 2000 Server, TCP/IP, Bancos de Dados em geral, do Microsoft Access e do Visual Basic. O leitor aprenderá na obra destinada do iniciante ao avançado detalhes sobre o modelo de dados relacional, como instalar o SQL Server 2000 em diferentes plataformas, além da criação e administração de bancos de dados, tabelas e outros objetos. Aprenda ainda Como criar páginas ASP que acessam os dados do SQL Server 2000.

Sumário

Introdução: Uma Nova Revolução no Desenvolvimento de Software?	1
Quem Deveria Ler Este Livro?	2
Equipamento e Software Necessários	2
Instalando e Testando o IIS 5.0	4
Como Obter e Instalar o Microsoft .NET Framework SDK	9
Configurações do Servidor Utilizado Neste Livro	12
Uma Visão Geral do Conteúdo do Livro	13
É Hora de Começar	13
 PARTE 1: O Framework .NET e a Linguagem C#	 14
 Capítulo 1: Entendendo a Nova Proposta da Microsoft – o Framework .NET	 15
Introdução	15
O Primeiro Contato com o Framework .NET	16
Apresentando o Conceito de Serviços – Web Services	16
Apresentando o CLR – Common Language Runtime	18
.NET Framework Class Library	19
Um Pequeno Parênteses Para “falar mal” dos Modelos Anteriores	21
Ai que Saudade do MS-DOS???	21
Prazer. Eu Sou o Windows!	22
Redes e Internet – Mais Problemas (ou Soluções) à Vista!	23
Modelo em 2 Camadas	23
Aplicações em 3 Camadas	25
Aplicações em Quatro Camadas	26
Questões a Considerarmos nos Modelos de 3 ou Mais Camadas Utilizados Atualmente	28
De Volta ao Framework .NET: os Demais Elementos	29
Linguagens Habilitadas ao .NET	29
Um Rápida Apresentação do VB.NET	30
Uma Rápida Apresentação do C#	30
Common Type System	32
Metadata	33
Assemblies	34
Interfaces com o Usuário	35
Windows Forms	35
Web Forms	36
Os Servidores .NET	37

SQL Server 2000	39
Exchange Server 2000	39
BizTalk Server 2000	39
Commerce Server 2000	40
Application Center 2000	40
Host Integration Server 2000	40
Internet Security and Acceleration Server 2000	40
Mobile Information 2001 Server	40
ADO.NET	41
Conclusão	43
Capítulo 2: Entendendo o CLR	45
Introdução	45
CLR – Common Language Runtime	46
Compilar ou não Compilar, eis a Questão?	47
O Papel dos Metadados (Metadata)	47
Integração Entre Diferentes Linguagens: Promessa ou Realidade?	47
O Processo de Execução de Código do CLR	48
Mais Algumas Observações Sobre o JIT	49
Assemblies	50
Uma Definição em Poucas Palavras	51
Funções do Assembly	51
Componentes do Assembly	52
O que Temos no “Manifesto”?	54
CTS – Common Type System	54
Classificação dos Tipos do CTS	55
.NET Framework Class Library – Biblioteca de Classes do Framework .NET	55
Conceitos Básicos de Orientação a Objetos	60
O que é um Objeto?	61
Mensagens	62
Classes	62
Herança	64
Reutilização de Código	68
Mais Alguns Detalhes Antes de Iniciarmos a Parte Prática	69
Conclusão	72
Capítulo 3: Apresentando o C#	73
Introdução	73
Mais uma Linguagem de Programação?	74
Como é “a cara” de um Programa Escrito em C#?	75

Tipos da Linguagem C#	82
Value Types	83
O Tipo Struct	86
Os Tipos de Enumeração	90
Reference Types	92
O Tipo String	92
O Tipo Arrays	96
Instruções de Fluxo de Controle no C#	99
Instruções de Seleção	100
A Instrução If	100
A Instrução Switch	105
Instruções de Repetição	109
A Instrução For	110
A Instrução While	114
A Instrução do/while	117
A Instrução Foreach	120
Instruções de Salto (Jump)	123
A Instrução Break	123
A Instrução Continue	123
A Instrução Goto	123
A Instrução Return	124
A Instrução Throw	124
Conclusão	124

Capítulo 4: Classes, Métodos, Herança e Polimorfismo com o C#	125
Introdução	125
Operadores e Mais Operadores	126
Built-in Operators	126
Operadores Aritméticos	127
Operador de Incremento: ++	130
Operador de Decremento —	131
Operadores Relacionais e Lógicos	131
Operadores de Atribuição (Assignment)	133
Precedência de Operadores	134
Classes	137
Para Começar um Exemplo Simples	137
Membros de uma Classe	140
Campos (Fields)	140
Métodos (Methods)	140
Propriedades (Properties)	141
Constantes (Constants)	141

Indexadores (Indexers)	141
Eventos (Events)	141
Construtores e Destrutores de uma Classe	141
Modificadores Para os Membros de uma Classe	144
Adicionando Funcionalidade a uma Classe – Métodos	144
O Polimorfismo Posto em Prática – Override	155
Conclusão	159
Capítulo 5: Tópicos Diversos em C#	160
Introdução	160
A Classe System.Math – Operações Matemáticas	161
Campos da Classe System.Math	161
Métodos da Classe System.Math	162
A Estrutura System.DateTime – Datas e Horas	174
Campos da Estrutura System.DateTime	176
Propriedades da Estrutura System.DateTime	176
O Namespace System.IO – Operações com Arquivos e Pastas	178
A Classe System.IO.File – Operações com Arquivos	178
Métodos da Classe System.IO.File	179
A Classe System.IO.FileInfo – Informações Sobre Arquivos	182
Propriedades da Classe System.IO.FileInfo	182
O Tratamento de Exceções	184
Utilizando “try” e “catch”	185
Utilizando “try” e “finally”	188
Conclusão	195
PARTE 2: Fundamentos do ASP.NET	196
Capítulo 6: Uma Introdução ao ASP.NET	197
Introdução	197
Mais uma Versão?	197
Uma Introdução à Tecnologia ASP	198
Conteúdo Dinâmico na Internet	200
Novidades e Melhorias do ASP.NET	202
Faz Parte do Framework .NET	202
Suporte a Múltiplas Linguagens	202
Menos Código Para Mais Trabalho	203
Separação Entre o Código HTML e o Código ASP	203
Maiores Facilidades Para Criação e Utilização de Componentes	207
Compatibilidade com Qualquer Navegador	207

Check List Para Acompanhar os Exemplos Deste Livro	207
ASP e ASP.NET Podem Rodar no Mesmo Servidor?	209
Enfim Vamos Iniciar com ASP.NET	209
A Extensão do Arquivo Mudou – .aspx	209
Inserindo Código ASP.NET	210
Um Pequeno Exemplo, só Para Começar	212
Exemplo 1	212
Banco de Dados do SQL Server 2000 Para os Exemplos Deste Livro	216
O Segundo Exemplo – Conectando com um Banco de Dados	216
Cadê o meu Objeto Recordset?	219
Conectando com o Banco de Dados	219
Um Controle Poderoso Para Exibir os Dados	220
Posso Ter um Comando que Ultrapassa uma Linha?	220
O Bom e Velho SQL Continua o Mesmo?	220
Conclusão	221

Capítulo 7: HTML Server Controls	223
Introdução	223
Uma Classe Chamada Page	224
Eventos ao Carregar uma Página ASP.NET	224
A Classe Page	225
Os Eventos da Classe Page	225
As Propriedades da Classe Page	227
Métodos da Classe Page	234
O Processamento de uma Página ASP.NET	235
Manutenção de Estado é Fundamental	236
HTML Server Controls	236
Uma Definição Para HTML Server Controls	237
HTML Server Controls Disponíveis	238
A Classe Base System.Web.UI.HtmlControls.HtmlControl	238
HTMLForm Control	239
HTMLInputText Control	242
HTMLInputCheckBox	244
HtmlTextArea Control	250
HTMLInputRadioButton Control	251
Controles Para a Criação de Tabelas: HtmlTable, HtmlTableRow e HtmlTableCell Controls	254
O Controle HtmlTable	254
O Controle HtmlTableRow	255
O Controle HtmlTableCell	256
Um Exemplo Prático	256
HTMLSelect Control	263

HTMLAnchor Control	267
HtmlInputButton Control	269
HtmlButton Control	270
HtmlImage Control	273
Conclusão	277
Introdução	279

Capítulo 8: Validation Server Controls 279

Validation Controls: Definição e Propriedades Gerais	280
Como é que Utilizamos os Controles de Validação?	281
A Mãe de Todos? Ou Seria o Pai de Todos?	282
Principais Propriedades da Classe BaseValidator	282
Principais Métodos da Classe BaseValidator	282
Um Exemplo, só Para Começar	283
O Controle RequiredFieldValidator	287
O Controle CompareValidator	293
O Controle RangeValidator	297
O Controle CustomValidator	300
O Controle RegularExpressionValidator	305
Conclusão	308

Capítulo 9: Web Form Controls 309

Introdução	309
A Classe Básica – WebControl	310
Principais Propriedades da Classe WebControl	310
Principais Métodos da Classe WebControl	311
Principais Eventos da Classe WebControl	312
TextBox Web Server Control	312
Label Web Server Control	318
CheckBox Web Server Control	320
RadioButton Web Server Control	324
Button Web Server Control	326
ListBox Web Server Control	329
Table, TableCell e TableRow Web Server Controls	331
Panel Web Server Control	337
Image Web Server Control	343
HyperLink Web Server Control	345
LinkButton Web Server Control	347
ImageButton Web Server Control	350
Conclusão	353

Capítulo 10:Acessando Bancos de Dados com ASP.NET – Parte 1	355
Introdução	355
Uma Visão Geral do Acesso a Dados	356
Quais as Principais Diferenças do ADO.NET em Relação ao ADO?	358
Bancos de Dados Utilizados nos Exemplos	359
O Banco de Dados do Microsoft Access – NorthWind.mdb	359
O Banco de Dados do SQL Server – Pubs	360
Uma Introdução ao ADO.NET	362
Informando que Você Deseja Utilizar Classes de um Determinado Namespace	362
Classe ou Objeto; Objeto ou Classe?	363
Estabelecendo Conexões	363
Estabelecendo uma Conexão com o SQL Server 2000 – SqlConnection	364
Estabelecendo uma Conexão com o Microsoft Access – OleDbConnection	369
Uma Visão Geral do Processo de Acesso a Dados	374
Criando Objetos Command	376
Retornando Dados com DataAdapter	380
O Objeto DataSet	382
O Objeto DataView	388
Um Pouco Mais Sobre o Controle DataGrid	405
Primeira Maravilha do DataGrid: Paginação	405
Segunda Maravilha do DataGrid: Mais do que um Conjunto de Dados na Mesma Página	407
Conclusão	413
Capítulo 11:Acessando Bancos de Dados com ASP.NET – Parte 2	414
Introdução	414
Um “tal de” Data Binding	415
À Maneira Antiga: Criando uma Lista Dinâmica com ASP 3.0	415
A Evolução: Data Binding com ASP.NET.	418
Data Binding de Valores Simples	422
Sintaxe Para o Data Binding	422
Data Binding de Múltiplos Valores. “repeated-value-binding”	425
O Controle CheckBoxList	425
O Controle DropDownList	434
O Controle RadioButtonList	435
Mais um Pouco Sobre o Controle DataGrid	440
Ordenação com o Controle DataGrid	440
Filtrando Dados com o Controle DataGrid	446
O Controle DataGrid em Detalhes	456
Um Exemplo de Criação Manual de Colunas – BoundColumn	460
Implementando Paginação com o Controle DataGrid	465

Conclusão	473
Capítulo 12: Acessando Bancos de Dados com ASP.NET – Parte 3	474
Introdução	474
Não Esqueça, Estamos em um Modelo Desconectado	475
O Objeto DataTable – Alterações nos Dados Desconectados	475
Inserindo, Excluindo e Adicionando Dados com o Objeto DataTable	477
Sincronizando Dados com o Banco de Dados	511
Excluindo Registros – a Propriedade DeleteCommand	529
Adicionando Registros – a Propriedade InsertCommand	531
Conclusão	532
PARTE 3: Conceitos Avançados do ASP.NET e Segurança	533
Capítulo 13: Web Services e Visual Studio .NET	534
Introdução	534
Uma Introdução aos Web Services	535
Possíveis Utilizações Para um Web Service	537
O que Diferencia Web Services das Tecnologias de Componentes Como COM ou CORBA?	538
Criando um Web Service	539
Sintaxe Para a Criação de um Web Service	539
Uma Maneira Fácil de Testar a Funcionalidade de um Web Service	541
Proxies: Conceito e Criação	545
Conceito	545
Criando o Proxy Utilizando o Utilitário Wsdl.exe	546
Compilando o Arquivo CalculosLegais.cs Para Gerar a DLL Correspondente	549
Disponibilizando a DLL Para que a Mesma Possa Ser Utilizada	550
Utilizando o Web Service em uma Página ASP.NET	552
O Novo Ambiente Gráfico de Desenvolvimento – Visual Studio .NET	558
O que há de Novo no Visual Studio .NET	558
Conclusão	577
Capítulo 14: Segurança de Aplicações Web com o IIS 5.0 e ASP.NET	579
Introdução	579
Autenticação de Usuários com o IIS 5.0	580
Autenticação Anônima	581
Como Definir a Conta Para Acesso Anônimo no IIS 5.0	583
Verificando as Configurações da Conta Para Acesso Anônimo no Windows 2000 Server	587
Configurando Permissões NTFS em Pastas do Servidor Web	592

Sistemas de Arquivos no Windows 2000 e Permissões NTFS	593
Definindo Permissões NTFS	596
Autenticação Básica	603
Autenticação Integrada do Windows	604
Autenticação Utilizando Certificados	605
Mapeamento do Certificado Cliente	606
Configurando o Tipo de Autenticação no IIS	606
Mais Configurações de Segurança do IIS	610
Configurando Opções Gerais de Segurança	610
Configurando de Segurança Para Aplicativos Web no IIS	613
Definindo Restrições de Acesso em Nível de Endereço IP	615
Mecanismos de Segurança do Banco de Dados	619
O que é uma Aplicação Web no IIS?	620
Uma Aplicação Cliente/Servidor Tradicional	620
Aplicações Web – um Novo Paradigma	621
O que é uma Aplicação Web no IIS?	622
Configurações de Segurança com o Arquivo Web.Config	629
Impersonation	629
O Arquivo Web.Config	630
Definindo o Tipo de Autenticação	634
Definindo os Usuários e Grupos que Têm Permissão de Acesso à Aplicação	634
Conclusão	639
Capítulo 15: Caixa de Ferramentas do ASP.NET	641
Introdução	641
Criação de Listas Dinâmicas	642
Edição de Dados com o Controle DataGrid	655
Acessando Dados de uma Planilha do Excel	667
Um Conceito Importante: "Code Behind"	670
O Objeto HttpRequest	675
O Objeto HttpResponse	681
Diretivas de Página	682
A Diretiva @Page	682
A Diretiva @Import	683
Outras Diretivas	683
Configurações de Segurança Através de Programação	684
Acessando Informações Sobre o Usuário Autenticado	684
Conclusão	691

Capítulo 16: Tratamento de Erros e Gerenciamento de Estado	693
Introdução	693
Tratamento de Erros no Framework .NET	694
Exceções e a Classe Exception	694
Revisando as Estruturas Try...Catch...Finally	696
Utilizando "try" e "catch"	696
Utilizando "try" e "finally"	700
Múltiplos Blocos Catch e Tratamento de Exceções Específicas	703
Um Exemplo de Tratamento de Exceção do Tipo SqlException	704
Conclusão	709
 ANEXO 1: Principais Tags do HTML e Criação de Contas e Grupos no Windows 2000	710
Introdução	710
Um Passeio Pelo HTML	710
Criação de Contas de Usuários e Grupos no Windows 2000	712
A Estrutura Básica de uma Página HTML	712
Relação das Principais Tags do HTML Utilizadas nos Exemplos do Livro	713
As Tags Para Criação de Títulos	713
Tags Para a Criação de Tabelas	714
Inserindo uma Linha Horizontal	717
Tags Para Formatação Básica do Texto	717
Formatação de Fonte	719
A Tag <A>	720
O Conceito de Contas de Usuários no Windows 2000	722
Alterando Propriedades Importantes das Contas de Usuários	727
Definindo as Políticas de Senhas	734
Grupos de Usuários e Tipos de Grupos Existentes no Windows 2000 Server	737
Criando Grupos de Usuários e Adicionando Usuários aos Grupos	740
Introdução	745
 Anexo 2: O Modelo de Dados Relacional	745
Conceitos Básicos de Bancos de Dados Relacionais	746
Entidades e Atributos	747
O Conceito de Chave Primária	749
Relacionamentos Entre Tabelas	750
Relacionamento do Tipo Um Para Um	751
Relacionamento do Tipo Um Para Vários	752
Relacionamento do Tipo Vários Para Vários	753
Integridade Referencial	754
Normalização de Tabelas	755

Primeira Forma Normal	755
Segunda Forma Normal	756
Terceira Forma Normal	757
Passos Para Projetar um Banco de Dados	758
Conclusão	760

ANEXO 3: A Linguagem SQL 761

Introdução	761
Noções Básicas da Linguagem SQL – Structured Query Language	761
Conhecendo o SQL	761
A Instrução SELECT	762
A Instrução UPDATE	769
A Instrução INSERT	771
A Instrução DELETE	771
Comandos Avançados da Linguagem T-SQL	772
Pesquisando Dados em Múltiplas Tabelas – Detalhes e Exemplos	772
Tipos de JOIN	774
JOIN com Mais do que Duas Tabelas	776
Utilizando Subconsultas	778
Conclusão	781

ANEXO 4: Sites Sobre ASP.NET, C# e XML 782

Introdução	782
Conclusão	785

Uma Nova Revolução no Desenvolvimento de Software?

INTRODUÇÃO

.NET, DOT.NET ou Microsoft.NET, não importa. O fato é que uma grande mudança está acontecendo. Esta mudança tem dois aspectos extremamente relevantes:

- ◆ É promovida pela Microsoft
- ◆ É uma mudança de Paradigma

Desde Junho de 2000 que a Microsoft vem anunciando a sua iniciativa .NET. Milhares de artigos já foram escritos sobre esta mais nova empreitada da gigante de Seattle. A idéia básica por detrás desta iniciativa é uma mudança, radical, no modelo de desenvolvimento, comercialização e utilização de Software. A iniciativa .NET é a visão da Microsoft para um mundo em que o software passa a ser comercializado na forma de Serviços, os quais são acessados através da Internet, utilizando-se protocolos padrão como XML (Extensible Markup Language) e SOAP (Simple Object Access Protocol).

São muitos os elementos que dão suporte ao .NET, dentre os quais destacam-se uma linha de servidores conhecidos como Servidores .NET, o sistema operacional Windows 2000 e suas futuras versões, além de um novo conjunto de ferramentas de desenvolvimento, conjunto este batizado de Visual Studio Next Generation ou comercialmente conhecido como Visual Studio .NET.

Um mundo em que o Software é visto como serviços, os quais podem ser acessados, através da Internet, por qualquer computador ou dispositivo autorizado, não é nenhuma novidade ou invenção da Microsoft. Empresas como IBM, Sun e ORACLE possuem suas próprias ferramentas e iniciativas nesta direção, algumas inclusive em estágios bastante avançados de desenvolvimento.

NOTA: Para uma relação completa de todas as alterações da versão Beta 2 em relação à Beta 1, consulte o seguinte endereço: <http://www.123aspx.com/b1to2changes/default.asp>.

A plataforma .NET é a aposta da Microsoft nesta direção. Porém todos sabemos que nenhuma empresa no mundo sabe divulgar tão bem suas iniciativas e produtos como a Microsoft. Quando o .NET foi anunciado em Junho de 2000, os elementos que davam suporte à iniciativa ainda encontravam-se na versão Beta 1. Em Junho de 2001, portanto um ano depois, foi lançada a versão Beta 2, conhecida como “Microsoft .NET Framework SDK”. A versão Beta 2 apresentou inúmeras alterações, inclusive com a retirada de alguns

componentes e a substituição por outros, o que fez com que códigos criados para a versão Beta 1 não fossem compatíveis com a versão Beta 2.

Durante este ano – entre o lançamento da versão Beta 1 e da versão Beta 2 –, proliferaram inúmeros sites na Internet, com milhares de artigos sobre o .NET. Isso fez com que a iniciativa da Microsoft tivesse uma visibilidade e divulgação jamais alcançadas por outras empresas. Inúmeros livros foram lançados baseados na versão Beta 1, o que indica o sucesso da iniciativa. Você já viu algum livro de Excel, por exemplo, baseado em uma versão Beta 1 do Office?

Nesta introdução veremos o que você precisa para iniciar com os estudos de .NET. Iremos mostrar qual o equipamento mínimo de que você deverá dispor para poder instalar o “Microsoft .NET Framework SDK” e acompanhar os exemplos deste livro. Também iremos orientá-lo sobre a obtenção e instalação do “Microsoft .NET Framework SDK”.

Quem Deveria Ler Este Livro?

Mesmo que você não conheça nada sobre .NET, C# ou ASP.NET, você será capaz de acompanhar todos os capítulos deste livro. Os capítulos foram organizados de tal maneira que os conceitos e exemplos são apresentados em uma ordem natural e intuitiva. Em outras palavras, procurei imaginar o que faria um leitor que nada conhecesse sobre .NET, C# e ASP.NET e quisesse iniciar os seus estudos. Com esta idéia em mente, procurei organizar os capítulos em uma seqüência que facilitasse o aprendizado de tais tecnologias. O conhecimento da tecnologia ASP pode facilitar e acelerar o aprendizado de alguns conceitos, porém não é um pré-requisito obrigatório. Ainda que o amigo leitor nada conheça sobre as versões anteriores do ASP, mesmo assim você será capaz de acompanhar todo o conteúdo deste livro.

NOTA: São várias as maneiras de se referir à nova plataforma .NET da Microsoft: DOT.NET, .NET, Plataforma .NET, Iniciativa .NET, Framework .NET

Para usuários que já trabalham com as versões anteriores do ASP, o livro oferece a possibilidade de conhecer os fundamentos básicos de uma nova linguagem – C#, bem como os elementos básicos do .NET e a nova versão do ASP – ASP.NET.

Em resumo, nos capítulos e anexos deste livro, você encontrará todas as informações necessárias para entender o modelo .NET. Também apresentaremos os elementos básicos da nova linguagem C#. Uma vez conhecendo o modelo .NET e a linguagem C#, estaremos aptos a criar aplicações utilizando ASP.NET.

Neste livro estarei utilizando as várias denominações, indiscriminadamente.

Equipamento e Software Necessários

Para acompanhar todos os exemplos propostos no livro, precisamos de um computador com Windows 2000 Server ou Windows 2000 Professional instalado. O IIS 5.0 – Internet Information Services também deve estar instalado. Mais adiante veremos como confirmar se o IIS está instalado e, caso o mesmo não esteja instalado, como proceder à instalação do mesmo.

NOTA: Poderíamos utilizar o Windows 98 ou Me, porém algumas opções não estariam disponíveis, uma vez que o servidor Web para esta plataforma é o PWS – Personal Web Server, o qual possui um número reduzido de funcionalidades em relação ao IIS.

A configuração mínima necessária depende de uma série de fatores, conforme descrito a seguir.

Se o equipamento vai ser utilizado como um servidor Web na Internet, vários são os aspectos a serem considerados, tais como:

- ◆ Aplicações implementadas e recursos de hardware necessários, tais como memória, disco e processador.
- ◆ Número estimado de acessos simultâneos e diários, sendo esta uma das estimativas mais difíceis de se obter, uma vez que o sucesso de um site pode ser determinado por um número muito grande de fatores, nem todos de fácil mensuração.
- ◆ Grau de segurança necessário e desejável. Neste caso, entram questões como criptografia, uso de certificados digitais, criação de VPN – Virtual Private Networks (Redes Privadas Virtuais), procedimentos de recuperação à falha, plano de contingência, etc.
- ◆ Percentual de “folga” desejado para recursos tais como: memória, processador, disco. Ao projetar um site, é bastante comum utilizar hardware que atenda as necessidades atuais com uma certa folga, para que seja possível atender a crescimentos maiores do que os estimados. Alguns sites famosos já enfrentaram problemas de crescimentos maiores do que o esperado. Quando isso acontece, mais recursos de hardware precisam ser adicionados com o site em funcionamento.

Para maiores informações sobre como planejar a capacidade de hardware para um servidor Web com o IIS 5.0, consulte o capítulo “Capacity Planning” do livro “Internet Information Services Resource Guide”, o qual é parte integrante do Windows 2000 Server Resource Kit.

Como equipamento para ser utilizado em casa, ou em um laboratório de teste na sua empresa, aconselho a seguinte configuração:

- ◆ Processador Pentium 450 ou superior.
- ◆ 128 MB de RAM, sendo 256 MB, altamente recomendáveis.
- ◆ 6 GB de Disco rígido.

Com menos de 128 MB de RAM, o desempenho, mesmo para uma máquina de testes, fica muito comprometido.

Além do Windows 2000 Server e do IIS 5.0, também precisaremos ter os seguintes programas e serviços instalados, para que possamos acompanhar os exemplos deste livro:

- ◆ Internet Explorer 5.01 ou superior.
- ◆ Microsoft .NET Framework SDK – Beta 2. Veremos como fazer o download e instalar este Framework mais adiante, nesta introdução.

Em cada um dos capítulos, estaremos apresentando uma série de exemplos práticos. Através destes exemplos, você poderá entender melhor a aplicação dos conceitos teóricos apresentados. Em muitas situações, a melhor maneira de

entender um determinado assunto é através da utilização do mesmo para resolução de um problema prático do dia-a-dia. Muitos dos exemplos apresentados podem ser facilmente adaptados para uso em aplicações que você esteja desenvolvendo. Situações como acesso a Banco de dados através de páginas Web, indexação e pesquisa de conteúdos, implementação de mecanismos de segurança, etc.; são comuns na maioria das aplicações Web atuais. Agora vamos verificar se o IIS 5.0 está instalado e, caso o mesmo não esteja, aprenderemos a fazer a instalação.

Instalando e Testando o IIS 5.0

Caso você não tenha instalado o IIS 5.0, quando da instalação do Windows 2000 Server ou Professional, é possível fazer a instalação a qualquer momento. Agora aprenderemos, passo a passo, a instalar o IIS 5.0. Nunca é demais lembrar que, sem o IIS 5.0, não será possível testar os exemplos práticos, propostos neste livro.

Para instalar o IIS 5.0:

1. Faça o logon no Windows 2000 Server ou Professional.
2. Abra o Painel de controle (Iniciar -> Configurações -> Painel de controle).
3. Abra a opção Adicionar ou remover programas.
4. Surgirá a janela indicada na Figura 1.

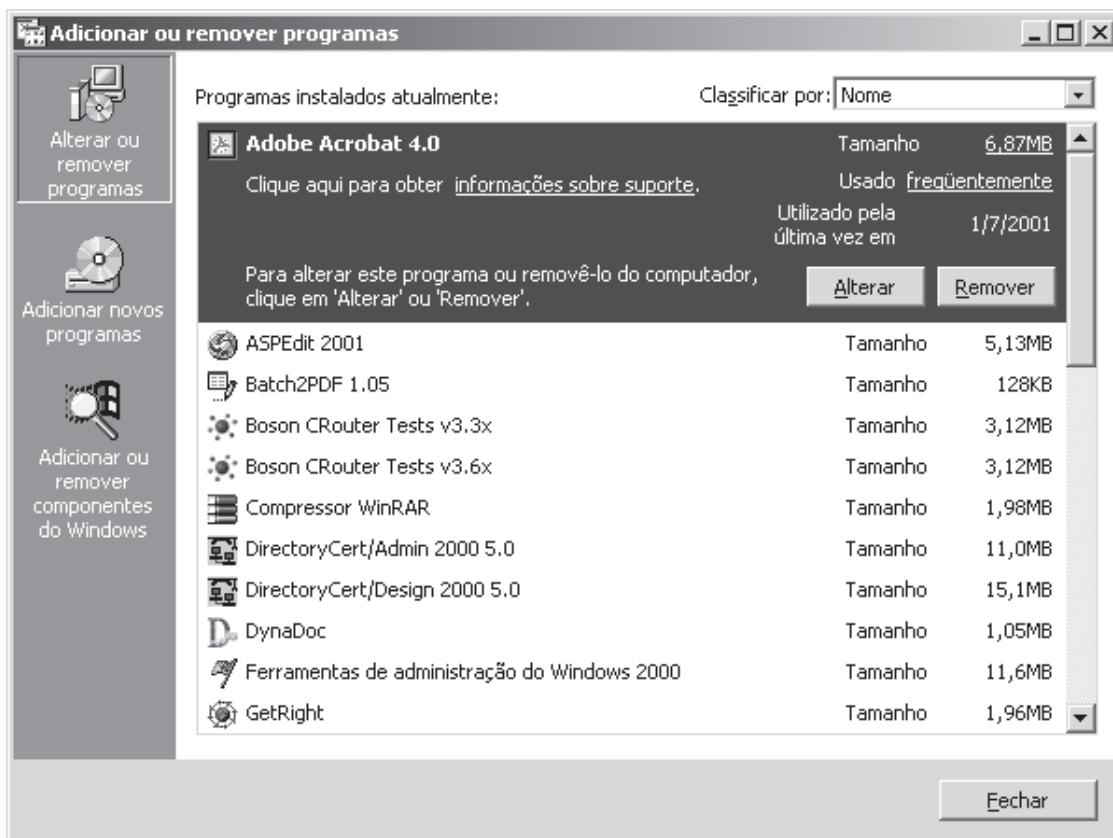


Figura 1: A janela Adicionar ou remover programas.

5. No lado esquerdo da janela, dê um clique na opção Adicionar ou remover componentes do Windows.

6. Surgirá a janela indicada na Figura 2.
7. Dê um clique no botão Componentes.
8. Será exibida, por breves instantes, uma mensagem “Aguarde...”. Depois surge a janela Assistente de componentes do Windows, conforme indicado na Figura 3.
9. Utilizando o Assistente de componentes do Windows, podemos adicionar novos serviços, ou remover serviços que não sejam mais necessários.
10. Vá descendo com a barra de rolagem vertical, até localizar o item Internet Information Services (IIS), conforme indicado pela Figura 4.

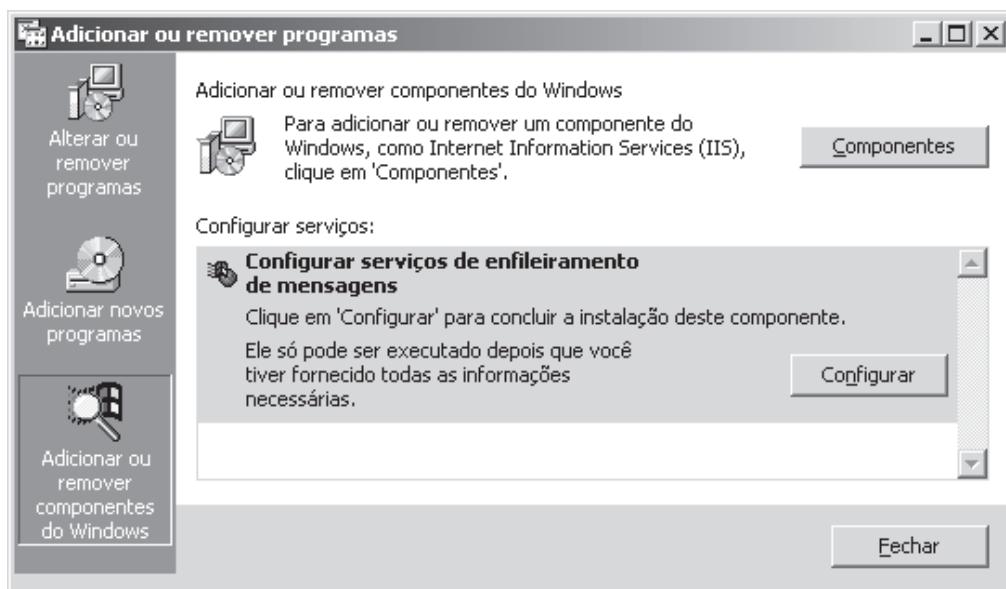


Figura 2: A janela para alterar a instalação do Windows.

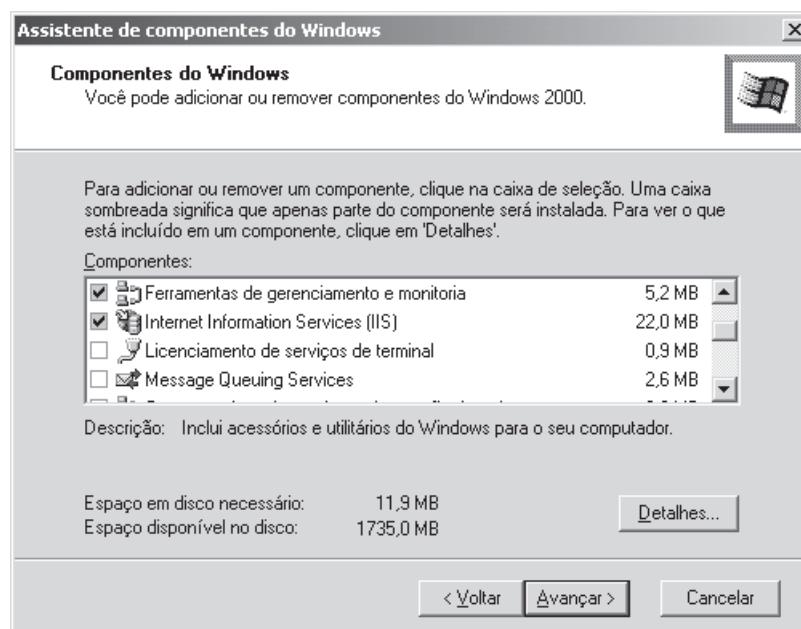


Figura 3: O Assistente de componentes do Windows.

11. Se esta opção estiver marcada, o IIS 5.0 já está instalado. Neste caso clique no botão Cancelar. Depois é só fechar a janela Adicionar ou remover programas e o Painel de controle.
12. Se esta opção estiver desmarcada, significa que o IIS 5.0 não foi instalado quando da instalação do Windows 2000. Marque esta opção para instalar o IIS 5.0.
13. Observe que, ao marcar a opção, o botão Detalhes é habilitado. O IIS 5.0 é formado por uma série de componentes e funcionalidades. Existe um servidor de páginas Web (servidor HTTP), um servidor de ftp, um servidor de notícias (NNTP), etc.

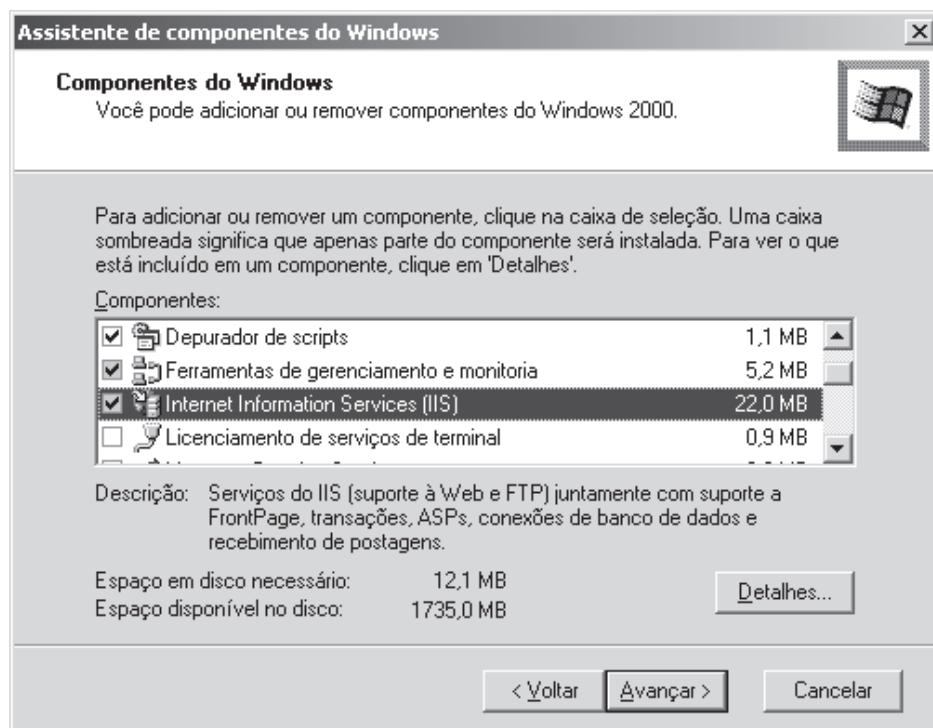


Figura 4: Serviço Internet Information Services (IIS).

DICA: Instale somente os serviços realmente necessários. Não é uma boa idéia instalar todos os serviços disponíveis, mesmo que somente alguns sejam utilizados. Quanto mais serviços instalados, maiores as possibilidades de ataque e quebra da segurança do site, por parte de um hacker, além de consumir memória e processador desnecessariamente.

14. Ao instalarmos o IIS 5.0, podemos escolher um ou mais dos seus componentes, dependendo das necessidades do nosso site. Não é necessário que todos os componentes do IIS 5.0 sejam instalados. Por exemplo, se o serviço de cópia de arquivos não for necessário, não temos por que instalar o serviço de ftp.
15. Clique no botão Detalhes.
16. Na lista de opções disponíveis, certifique-se de que somente as seguintes estão marcadas:
 - ◆ Arquivos comuns.
 - ◆ Documentação.
 - ◆ Extensões de servidor do FrontPage 2000.
 - ◆ Gerenciador de Internet Services (HTML).
 - ◆ Servidor File Transfer Protocol (FTP).
 - ◆ Servidor World Wide Web.

- ◆ Snap-In do Internet Information Services.
17. Observe que, após ter selecionado os componentes a serem instalados, o Windows 2000 Server exibe o espaço em disco necessário, conforme indicado pela Figura 5.
18. Dê um clique em OK. Você estará de volta ao Assistente de componentes do Windows.

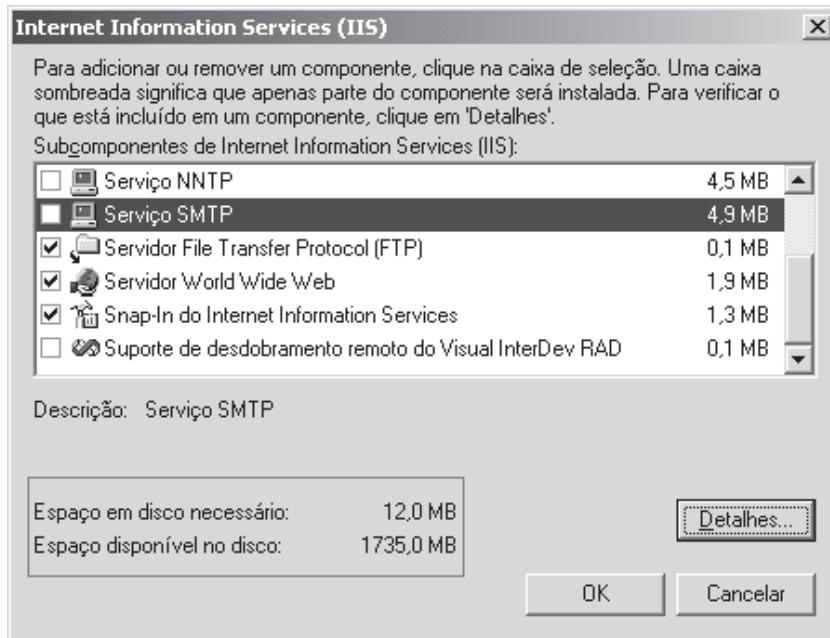


Figura 5: Indicação do espaço necessário no disco rígido.

19. Dê um clique no botão Avançar para ir para a próxima etapa do assistente.
20. O Windows 2000 Server exibe uma janela indicando o progresso da Instalação.
21. Caso o Windows 2000 Server não encontre os arquivos necessários à instalação, no Disco rígido, você será solicitado a inserir o CD de instalação do Windows, conforme indicado pela Figura 6.

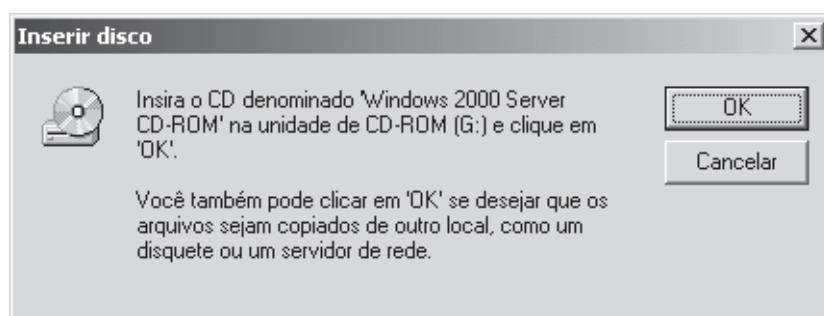


Figura 6: Mensagem solicitando o CD de instalação do Windows.

22. Insira o CD e dê um clique em OK. O Windows 2000 inicia o processo de cópia dos arquivos.
23. Depois de concluída a cópia dos arquivos, o Assistente emite uma mensagem dizendo que o processo foi concluído com sucesso.

24. Dê um clique no botão Concluir, para encerrar o Assistente.
25. Você estará de volta à janela Adicionar ou remover programas. Dê um clique no botão Fechar para sair desta janela.
26. Você estará de volta ao Painel de controle. Feche o Painel de controle.
27. Agora o IIS 5.0 está instalado e pronto para funcionar.

Agora que já temos o IIS 5.0 instalado, vamos testar se o mesmo está funcionando corretamente.

Para testar se o IIS 5.0 foi instalado com sucesso:

1. Abra o Internet Explorer.
2. Digite o seguinte endereço: <http://localhost>.
3. Deve surgir uma janela conforme indicado na Figura 7.

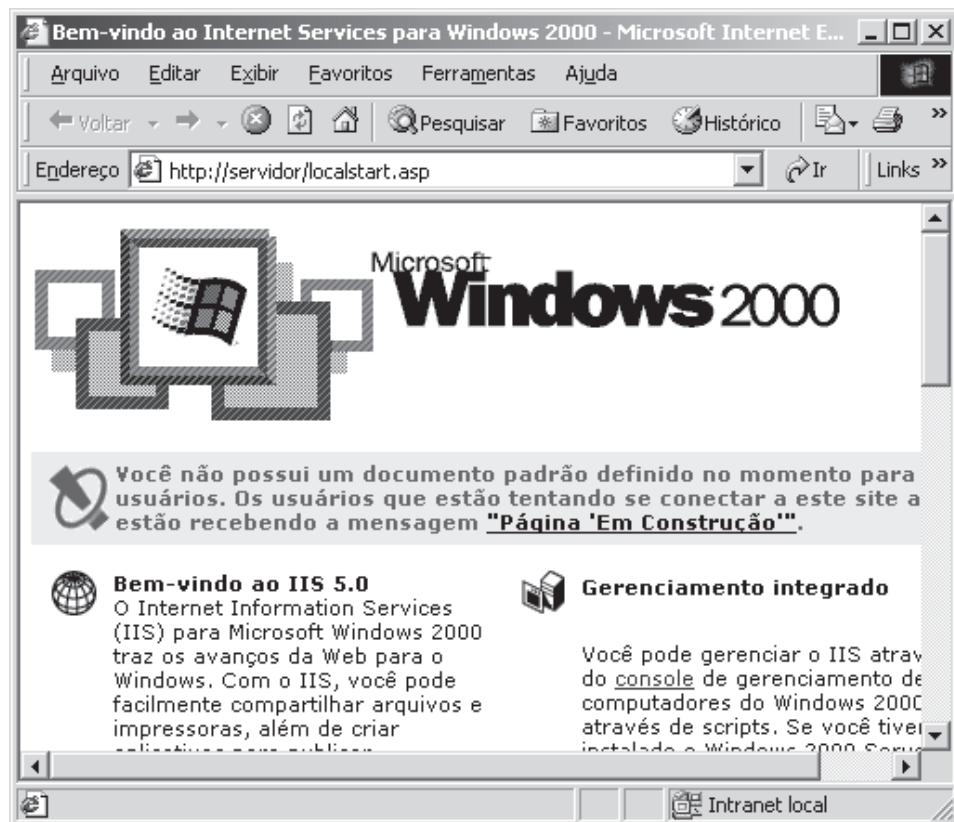


Figura 7: A página padrão do IIS 5.0, logo após a instalação.

4. Esta é a página inicial padrão, do IIS 5.0, logo após a instalação.
5. Isto comprova que o IIS 5.0 foi instalado com sucesso.
6. Feche o Internet Explorer.

Agora que já temos o IIS 5.0 instalado e testado, temos que partir para a instalação do “Microsoft .NET Framework SDK”. Este Framework instala todos os componentes necessários para que possamos criar páginas ASP.NET, utilizar a linguagem C#, além de todos os demais componentes necessários para o .NET. É importante salientar que o Visual

Studio .NET não faz parte deste Framework. No Capítulo 13 veremos a instalação e configuração do Visual Studio .NET. No momento em que este livro estava sendo escrito, o Visual Studio .NET encontrava-se em Beta 2.

Como Obter e Instalar o Microsoft .NET Framework SDK

Para que seja possível a criação de páginas ASP.NET no IIS 5.0 precisamos instalar o Microsoft .NET Framework SDK. O mesmo pode ser obtido a partir do site da Microsoft, para Download, no seguinte endereço: <http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/msdn-files/027/000/976/msdncompositedoc.xml&frame=true>

Neste endereço temos a opção de baixar um arquivo único, com 126 MB ou dividir o download em diversas partes, conforme indicado na Figura 8.

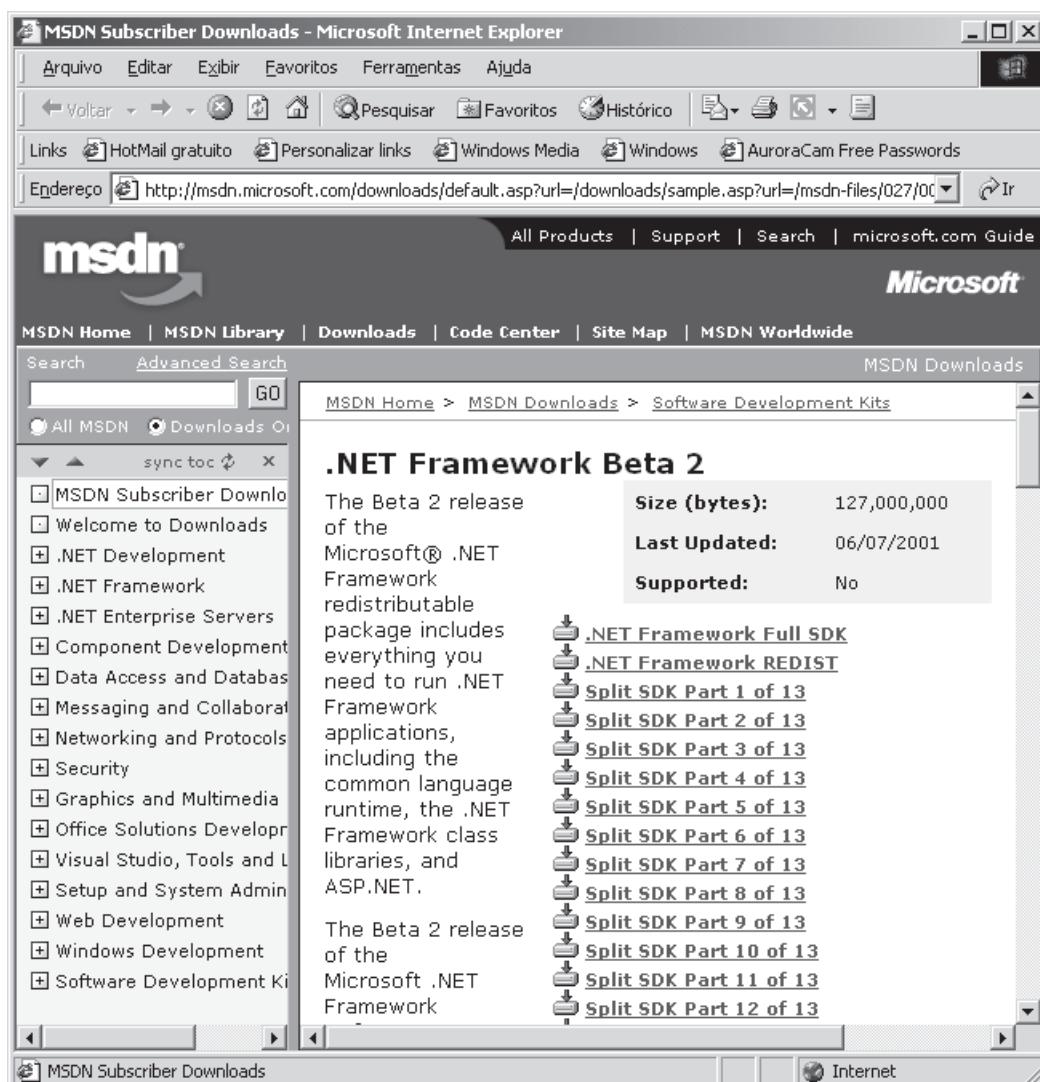


Figura 8: Página para download do Microsoft .NET Framework SDK.

Se você optar pelo download do arquivo completo, após concluir o download você terá um arquivo chamado Setup.exe de 126 MB. Este é o arquivo completo para a instalação do Framework .NET. Mais adiante veremos como proceder à instalação deste arquivo.

Para que o Microsoft .NET Framework SDK Beta 2 possa ser instalado com todas as suas funcionalidades, você também precisa instalar MDAC – Microsoft Data Access Components 2.7, Beta 2. Você pode fazer o download desta versão do MDAC, no seguinte endereço: http://download.microsoft.com/download/platformsdk/MDAC2.7/1.0/NT45XP/EN-US/MDAC_TYP_dnld.exe

O arquivo MDAC_TYP_dnld.exe possui cerca de 5MB. Antes de instalar o Framework .NET vamos instalar o MDAC 2.7 Beta 2.

Para instalar o MDAC 2.7 Beta 2 faça o seguinte:

1. Localize o arquivo MDAC_TYP_dnld.exe que você copiou anteriormente.
2. Dê um clique duplo no mesmo para iniciar a instalação do MDAC 2.7.
3. Surge uma tela com o contrato de licença de uso do MDAC 2.7. Dê um clique no botão Yes para aceitar o contrato.
4. Surge uma tela perguntando em que pasta temporária você deseja descompactar os arquivos. Digite C:\temp e dê um clique no botão OK. Os arquivos necessários serão descompactados na pasta C:\temp.
5. Vá para a pasta C:\temp e execute o arquivo MDAC_TYP.EXE.
6. Surge uma tela onde você precisa marcar uma caixa de controle para concordar com o contrato de licença. Marque esta caixa de controle, conforme indicado na Figura 9 e dê um clique no botão Avançar.

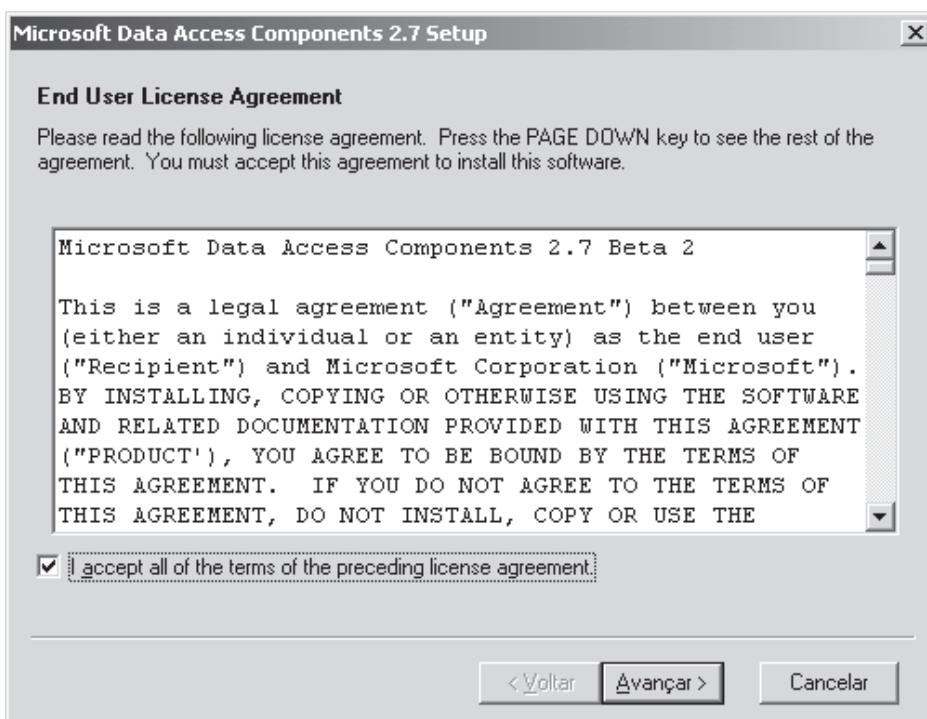


Figura 9: Contrato de licença para o MDAC 2.7.

7. Se existir algum serviço que será atualizado, em uso, o assistente de instalação avisa e pede para que o respectivo serviço seja parado. Para parar um serviço você pode utilizar o comando Iniciar -> Programas -> Ferramentas Administrativas -> Serviços.
8. Após a verificação dos serviços será exibida uma tela informando que o MDAC 2.7 será instalado. Dê um clique no botão Concluir e a instalação será iniciada.
9. Dentro de poucos minutos surge uma tela informando que o MDAC 2.7 foi instalado com sucesso. Dê um clique no botão Close, para fechar esta tela.

Após a instalação do MDAC 2.7, estamos prontos para iniciar a instalação do Microsoft .NET Framework SDK.

Para instalar o Microsoft .NET Framework SDK faça o seguinte:

1. Localize o arquivo Setup.exe que você copiou anteriormente.
2. Dê um clique duplo no mesmo para iniciar a instalação.
3. Surge uma tela perguntando se você deseja iniciar a instalação. Dê um clique no botão Sim para seguir adiante.
4. O programa de instalação iniciará a extração dos arquivos necessários à instalação. Este procedimento demora alguns minutos.
5. Encerrada a extração, surgirá a tela inicial do assistente de instalação. Dê um clique no botão Next para seguir para a próxima etapa.
6. Na segunda etapa você precisa marcar a opção “I accept the agreement”, para aceitar o contrato de licença do Framework .NET e seguir adiante.
7. Marque a opção “I accept the agreement” e dê um clique no botão Next para ir para a próxima etapa.
8. Nesta etapa temos a opção “the install the SDK (Software Development Kit) que é o Framework .NET propriamente dito, os exemplos. Observe que a opção para instalar a Documentação já vem marcada e não pode ser desmarcada.
9. Por padrão todas as opções já vêm marcadas. Vamos aceitar este padrão. Dê um clique no botão Next para ir para a próxima etapa.
10. Nesta etapa devemos definir a pasta de destino. Por padrão é selecionada a pasta: \Arquivos de programas\Microsoft.NET\FrameworkSDK\ do drive onde está instalado o Windows 2000. Vamos aceitar a pasta sugerida pelo Setup. Dê um clique no botão Next para ir para a próxima etapa.
11. Nesta etapa é iniciada a instalação dos diversos componentes. Este processo demora alguns minutos. À medida que os componentes vão sendo instalados, o andamento da instalação vai sendo exibido, conforme indicado na Figura 10.
12. No final surge uma mensagem dizendo que a instalação foi completada com sucesso. Dê um clique no botão OK para fechar esta mensagem.

Agora o Microsoft .NET Framework está instalado e pronto para ser utilizado. Com isso instalamos todo o Software necessário para acompanhar os exemplos deste livro.

Apenas para confirmar, podemos verificar se a instalação ocorreu com sucesso. Selecione o comando Iniciar -> Programas -> Microsoft .NET Framework SDK. Dentre as várias opções exibidas selecione a opção Documentation.

Esta opção nos dá acesso à Documentação completa sobre a plataforma .NET. A versão sempre atualizada desta documentação pode ser encontrada no site MSDN da Microsoft, no seguinte endereço: <http://msdn.microsoft.com/net>

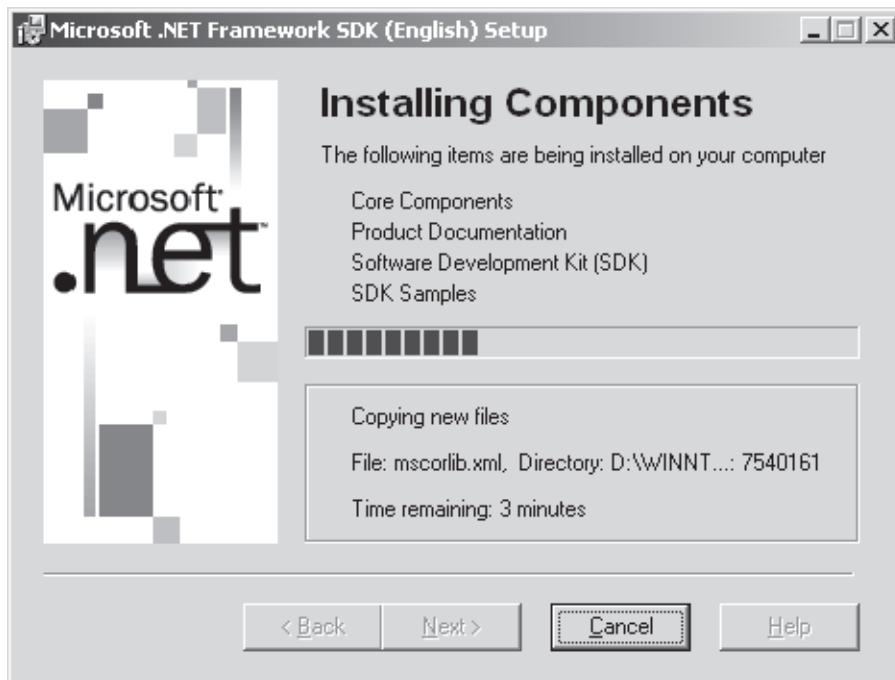


Figura 10: Andamento da instalação.

Configurações do Servidor Utilizado Neste Livro

Para criar os exemplos deste livro, utilizei um servidor com as seguintes configurações:

- ◆ Pentium 450
- ◆ 256 MB RAM
- ◆ Windows 2000 Server em Português
- ◆ IIS 5.0
- ◆ Microsoft .NET Framework SDK
- ◆ Nome do servidor: Servidor
- ◆ Endereço IP: 10.204.123.1
- ◆ Máscara de sub-rede: 255.255.255.0
- ◆ Domínio DNS: groza.com

O endereço para acessar a página inicial do servidor é o seguinte: <http://www.groza.com> ou <http://localhost>. Sempre que for feita referência a um destes dois endereços, você deve substituir pelo endereço do equipamento que você está utilizando para acompanhar os exemplos deste livro.

Uma Visão Geral do Conteúdo do Livro

O livro está dividido em três partes, conforme descrito a seguir:

- ◆ **Parte 1 – Capítulos 1 a 5:** Nesta parte trataremos sobre o Framework .NET, o que o mesmo apresenta de novo, quais os seus principais componentes e como o mesmo pode tornar mais fácil, rápido e eficiente o desenvolvimento de aplicações. Também veremos os fundamentos básicos da nova linguagem C# (leia-se C Sharp).
- ◆ **Parte 2 – Capítulos 6 a 12:** Veremos o que é o ASP.NET, quais as novas características e como criar páginas ASP.NET. Aprenderemos a conectar páginas com Bancos de Dados, utilizando ADO.NET e também a hierarquia de classes do Framework .NET.
- ◆ **Parte 3 – Capítulo 13 em diante:** Nesta parte final do livro veremos alguns exemplos de aplicativos utilizando ASP.NET bem como diversas técnicas e dicas úteis para a criação de aplicativos com ASP.NET. Vamos aprender a instalar o Visual Studio .NET e utilizá-lo para a criação e publicação de páginas ASP.NET. Veremos o que são e como criar Web Services. Também trataremos sobre segurança.

É Hora de Começar

Agora que já temos o nosso computador preparado para acompanhar os exemplos deste livro, é hora de iniciarmos o nosso aprendizado. Conforme veremos, o .NET representa uma mudança radical em relação aos modelos de programação anteriores. Aprender um novo modelo é uma tarefa desafiadora. Lembram do programador COBOL para o Mainframe tendo que aprender programação estruturada e, depois, tendo que passar de programação estruturada para orientação a objetos?

Se você está justamente querendo aprender os conceitos deste novo modelo, então este é o livro certo. Apresentaremos as principais dúvidas para aqueles que estão iniciando com o .NET e com ASP.NET.

O código-fonte para os exemplos em C# e para as páginas ASP.NET dos capítulos deste livro estão disponíveis para download no site da editora Axcel Books em www.axcel.com.br.

Caso você tenha sugestões sobre tópicos que gostaria de ver incluídos em futuras edições deste livro, ou queira relatar algum erro encontrado no livro, basta entrar em contato através do e-mail: livroaspnet@juliobattisti.com.br.

Desejo a todos uma boa leitura e tenho certeza de que este trabalho irá ajudá-los no entendimento deste novo paradigma de programação, bem como irá ajudá-los a, rapidamente, criar aplicações Web baseadas em ASP.NET.

P A R T E

1

O Framework .NET e a Linguagem C#

CAPÍTULO

1

Entendendo a Nova Proposta da Microsoft – o Framework .NET

Neste capítulo apresentaremos o Framework .NET. Veremos qual a proposta da Microsoft com esta iniciativa.

Iniciaremos o capítulo com uma visão geral, de alto nível, sobre as propostas do Framework .NET. Em outras palavras: “A que veio e a que se propõe o .NET?” Vamos mostrar quais os principais componentes e quais soluções pode nos oferecer o Framework .NET. Antes de entrarmos nos detalhes sobre cada elemento e das funções de cada elemento, é importante entendermos como este novo modelo poderá ajudar no desenvolvimento de Software e, em consequência, ajudar as empresas a terem sistemas melhores, mais rápidos, mais econômicos e mais sintonizados com as reais necessidades de cada empresa.

Uma vez devidamente apresentado o Framework .NET, falaremos sobre os modelos atuais de desenvolvimento e utilização de Software. Veremos quais os problemas do modelo atual e quais as soluções propostas para estes problemas, pelo Framework .NET.

Um novo paradigma de desenvolvimento tem que mostrar que é capaz de resolver, se não todos, pelo menos a grande maioria dos problemas dos modelos existentes. Já passou o tempo, se é que um dia esta prática foi aceitável, de mudar de tecnologia somente para estar atualizado com o mercado. Aprender novas técnicas e ferramentas envolve tempo, um longo aprendizado e, principalmente, pesados investimentos financeiros. Para que uma empresa esteja disposta a fazer todos estes investimentos, o Framework .NET terá que demonstrar que realmente é capaz de apresentar soluções para os grandes problemas dos modelos atuais de desenvolvimento de software.

Em seguida passaremos a apresentar os principais elementos que compõem o Framework .NET:

- ◆ CLR – Common Language Runtime
- ◆ Metadata
- ◆ Assemblies
- ◆ Linguagens habilitadas ao .NET
- ◆ Common Type System
- ◆ Interoperabilidade entre diferentes linguagens
- ◆ Web Services
- ◆ Os servidores .NET
- ◆ ADO.NET

Veremos, de uma maneira genérica, o papel de cada um destes componentes no Framework .NET. Estaremos estudando, detalhadamente, cada um destes itens nos demais capítulos deste livro. Apresentamos uma visão geral de cada um deles, para que o leitor já possa ter uma idéia do Framework .NET e de seus principais componentes, bem como da função de cada um.

Este capítulo, embora bastante teórico, forma a base para o entendimento dos conceitos apresentados ao longo de todo o livro. Por isso é importante que você entenda os conceitos apresentados. A documentação atualizada, sobre os conceitos apresentados, pode ser consultada no seguinte endereço: <http://msdn.microsoft.com/net>.

O Primeiro Contato com o Framework .NET

Definir exatamente o que é o Framework .NET não é uma tarefa das mais simples. Como não sou muito afeito a definições formais, vou mostrar qual a proposta do Framework .NET, quais os componentes e qual a função de cada componente.

Em primeiro lugar podemos afirmar que, com a iniciativa .NET, a Microsoft está mudando radicalmente o modelo de desenvolvimento e utilização de software. No livro “Introducing .NET”, da editora Wrox (www.wrox.com), encontramos duas afirmações interessantes sobre o Framework .NET:

- ◆ É uma mudança tão grande ou até maior do que a mudança do DOS para o Windows;
- ◆ O Framework .NET foi todo projetado já tendo a Internet como objetivo, diferente de outras plataformas que foram adaptadas para a Internet, à medida que a rede mundial crescia de importância.

Apresentando o Conceito de Serviços – Web Services

Realmente a mudança é bastante grande. Programadores, analistas e gerentes de projeto precisarão de muito estudo e tempo para absorver os conceitos desta nova plataforma. Também é verdade que, desde a sua concepção, o .NET foi projetado para a Internet. Em muitas publicações especializadas aparece a seguinte afirmação: “A iniciativa .NET é a visão da Microsoft de um mundo onde o Software se transforma em serviços, na verdade pequenos componentes que podem ser utilizados por qualquer aplicação.”

Um software de uma empresa brasileira pode utilizar um serviço que está residente em um servidor de uma empresa do Japão, desde que tenha permissões para isso. Este serviço pode oferecer, por exemplo, a funcionalidade para validação de uma transação via cartão de crédito. Um dispositivo móvel, como um celular WAP ou um Handled, pode utilizar um serviço de cotação de ações de um servidor da bolsa de valores de Londres. E como todos estes componentes fazem para se comunicar? Evidentemente que através da Internet. Por isso a Internet como ponto principal do projeto .NET.

Conforme veremos mais adiante, estes serviços, que podem ser acessados via Internet ou através de qualquer Intranet, são chamados de “Web Services”.

Ao invés de programas monolíticos, em que toda a funcionalidade necessária faz parte do próprio programa, construiremos programas como se fosse um jogo de montar. As diversas funcionalidades necessárias ao programa podem ser oferecidas através do acesso a serviços já implementados. Na Figura 1.1 temos uma pequena ilustração deste conceito.

No site da empresa [www\[minhaempresa.com\]](http://www[minhaempresa.com]), criamos, por exemplo, uma página ASP.NET para venda de livros. O preço dos livros está em dólares e deve ser convertido para a moeda do país do cliente, no momento da compra. No exemplo da Figura 1.1, a página ASP.NET utiliza um Web Service do servidor [www\[cotacoes.com\]](http://www[cotacoes.com]), para obter a cotação atualizada do dólar em relação à moeda do país do cliente. A página ASP.NET recebe esta informação, faz os cálculos necessários e exibe para o cliente. O próximo passo é efetivar a venda. Agora o cliente digita o número do seu cartão de crédito. Para validar o número do cartão de crédito do cliente, é acessado um outro Web Service, o qual está instalado no servidor [www\[validacao.com\]](http://www[validacao.com]).

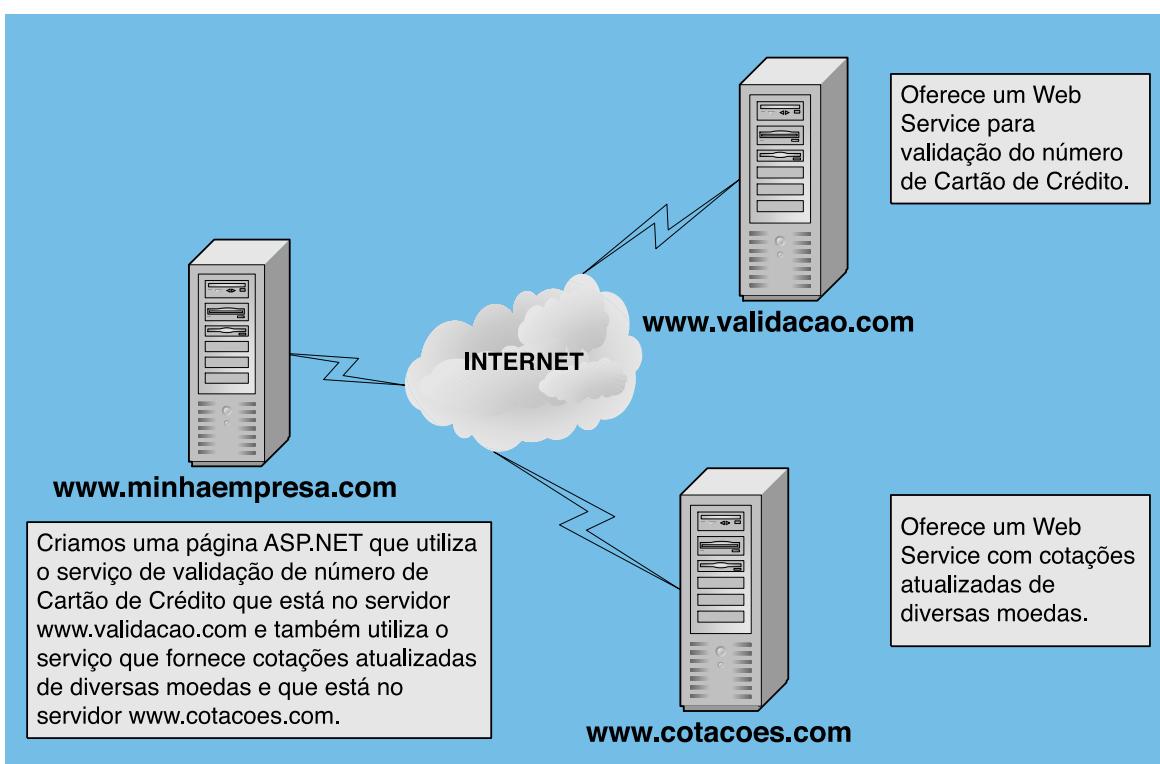


Figura 1.1: Um programa que utiliza diversos serviços.

Com esta arquitetura, a criação de software fica bastante simplificada, uma vez que podemos agregar ao nosso programa funcionalidades disponibilizadas através de Web Services que estão instalados em qualquer servidor da Internet. Desta maneira somente precisamos criar o que é específico do nosso programa. Com isso estamos reaproveitando código ou como preferem os puristas: reutilizando código.

Como diferentes Web Services, criados por diferentes empresas, poderão se comunicar e trocar informações? Esta comunicação é possível, porque todo programa criado para a plataforma .NET utiliza um padrão para troca de informações. No caso temos dois padrões:

- ◆ Para o formato dos dados, o Framework .NET utiliza XML – Extensible Markup Language. Conforme veremos mais adiante, o XML vem ganhando força como um padrão para troca de informações na Internet, principalmente para troca de informações entre empresas em sites de Comércio Eletrônico.
- ◆ Como protocolo de transporte, o Framework .NET utiliza o SOAP – Single Object Access Protocol. Com SOAP podemos fazer com que toda a comunicação entre diferentes Web Services e demais componentes de um programa seja feita através do protocolo padrão da Internet que é o HTTP. O SOAP não é um protocolo criado pela Microsoft para ser utilizado pelo .NET. O SOAP é um padrão da indústria, sendo utilizado por produtos de diversas empresas, como IBM, Sun e ORACLE.

Pode parecer que não existe nada de novo nesta abordagem, pois tecnologias para criar programas a partir de componentes prontos já existem há um bom tempo: COM+, CORBA, etc. Porém a grande vantagem dos Web Services, no Framework .NET, é que os mesmos podem ser acessados, facilmente, através da utilização de um protocolo padrão: SOAP e trocando informações em um formato padrão: XML. Esta abordagem torna a criação e utilização dos Web Services uma tarefa muito mais simples do que, por exemplo, a criação de componentes COM+ ou CORBA, os quais utilizam formatos de dados e protocolos de comunicação proprietários.

Falaremos mais sobre Web Services e também aprenderemos a criá-los, mais adiante neste livro.

Apresentando o CLR – Common Language Runtime

O CLR é um ambiente de execução, e poderíamos até dizer que é o “Coração do .NET”, o qual dá suporte a todas as linguagens de programação habilitadas para o .NET. Ao instalarmos o Microsoft .NET Framework SDK, temos disponíveis as seguintes linguagens:

- ◆ VB.NET (Visual Basic .NET)
- ◆ C# (leia-se C Sharp)
- ◆ ASP.NET
- ◆ Jscript.NET

O Runtime (ambiente de execução) é o ambiente que dá suporte à execução das aplicações .NET. Quando um programa .NET é executado, todo o controle do mesmo é feito através do CLR.

Para aplicações anteriores, desenvolvidas utilizando COM/COM+, o programador era responsável por inserir no código do programa uma série de funções necessárias ao correto funcionamento do mesmo, como por exemplo o Gerenciamento de memória, criação e destruição de objetos. A codificação destas funções não era uma tarefa fácil, o que exigia muito tempo do programador, além de conhecimentos avançados. Com aplicações .NET, todas estas funções são executadas pelo CLR, ou seja, o programador não precisa preocupar-se com as mesmas. Desta forma somente precisamos nos preocupar com a funcionalidade do nosso programa, o que poupa tempo e agiliza o processo de desenvolvimento.

As aplicações criadas em uma das linguagens habilitadas para o .NET (como VB.NET, C# ou ASP.NET), ao serem compiladas, geram um código intermediário conhecido como MSIL – Microsoft Intermediate Language, o qual é abreviado simplesmente como IL – Intermediate Language. Este código é que é executado pelo CLR. Vamos analisar o diagrama apresentado na Figura 1.2:

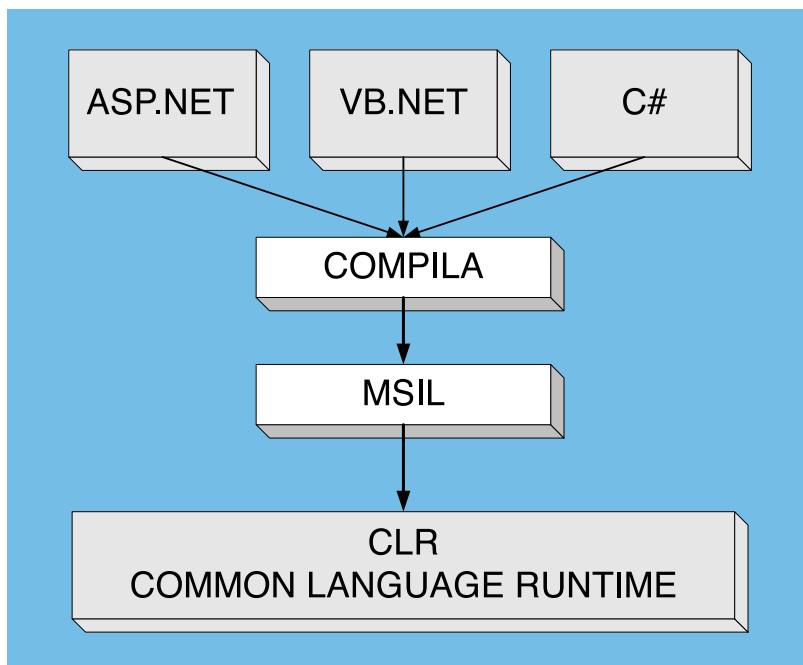


Figura 1.2: O ambiente de execução do CLR.

A partir da Figura 1.2 temos algumas observações importantes:

- ◆ Para que uma linguagem seja habilitada ao .NET, o seu compilador deve ser capaz de gerar código MSIL.
- ◆ O Código MSIL gerado é executado pelo CLR.

A própria Microsoft não nega que a idéia é bastante semelhante ao ambiente de execução para aplicações desenvolvidas em Java, onde temos um ambiente de execução comum – a Máquina Virtual Java, a qual executa byte code, que é o código gerado pelas aplicações Java.

Todas as linguagens habilitadas ao .NET têm a sua disposição um rico conjunto de classes e objetos, os quais fornecem desde conexão com banco de dados até funções mais específicas, como funções gráficas, de escrita em disco, etc. Este conjunto de classes e objetos é conhecido como “.NET Framework class library”.

Todo código habilitado a rodar no CLR, ou seja, que pode ser compilado para MSIL é conhecido como “managed code” ou código gerenciado. Código antigo, como por exemplo COM ou COM+, os quais não estão habilitados para rodar sob controle do CLR são chamados de “unmanaged code” ou código não gerenciado. Veremos mais detalhes a respeito destas definições no Capítulo 2, onde trataremos o CLR em mais detalhes.

.NET Framework Class Library

Este é o segundo elemento fundamental do Framework .NET. A .NET Framework class library (biblioteca de classes do Framework .NET), como o próprio nome sugere, é uma coleção de classes ou tipos completamente integrada com o ambiente de execução – CLR. Quando falamos em um conjunto de Classes, estamos utilizando o conceito originado no modelo de Programação Orientado a Objetos. Conforme veremos no decorrer deste livro, o Framework .NET é fortemente baseado nos conceitos de orientação a objetos, principalmente nos conceitos de Classes, Herança e Polimorfismo.

Os programas criados em qualquer linguagem habilitada ao .NET podem utilizar este conjunto de tipos e classes. Por exemplo, existe uma classe chamada System.Data, a qual oferece uma série de objetos e métodos para acesso às mais variadas fontes de dados. Vamos a um exemplo mais específico: existe uma classe chamada System.SqlClient (na versão Beta 1 existia uma classe chamada System.SQL, que foi descontinuada na versão Beta 2), a qual fornece uma série de métodos para acesso nativo aos dados de um servidor Microsoft SQL Server. Podemos utilizar a classe System.SqlClient em um programa feito em VB.NET, C#, em uma página ASP.NET ou qualquer linguagem que venha a ser habilitada para o .NET.

Ao fornecer um conjunto de classes e tipos, estamos facilitando a vida do programador, uma vez que grande parte da funcionalidade necessária é fornecida diretamente pelo Framework .NET e, o principal, é utilizada de uma maneira padronizada, pois a maneira de utilizar uma classe da biblioteca de classes do .NET é a mesma, independente da linguagem.

São muitas as funções disponibilizadas pela biblioteca de classes do .NET, conforme veremos no decorrer deste livro. Apenas a título de exemplo, vamos citar algumas funções disponibilizadas:

NOTA: Já existem diversos fabricantes trabalhando para habilitar suas linguagens de desenvolvimento para o Framework .NET.

- ◆ Manipulação de String
- ◆ Conectividade com banco de dados
- ◆ Acesso a arquivos
- ◆ Segurança
- ◆ Manipulação de dados

Na Figura 1.3, temos uma visão geral dos principais elementos que formam o Framework .NET.

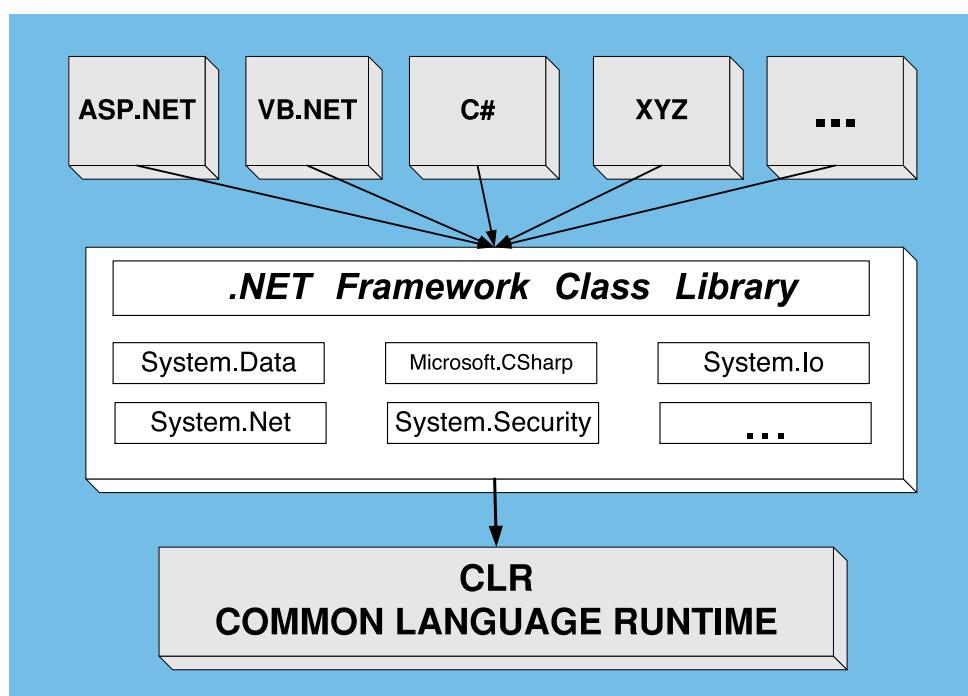


Figura 1.3: Principais elementos do Framework .NET.

Um Pequeno Parênteses Para “falar mal” dos Modelos Anteriores

Vamos falar um pouco sobre o modelo de desenvolvimento atual, mais especificamente sobre o modelo baseado em COM/COM+ da Microsoft. Para falar sobre este assunto vamos fazer um histórico sobre o desenvolvimento, desde os bons e velhos tempos do MS-DOS.

Ai que Saudade do MS-DOS???

Não, não é saudade da época do MS-DOS, apenas uma breve recapitulação. Para desenvolver aplicações para o ambiente MS-DOS, utilizamos diversas linguagens de programação, tais como:

- ◆ Clipper
- ◆ Pascal – Turbo Pascal
- ◆ Basic
- ◆ C – Turbo C
- ◆ C++

Um programa para o ambiente MS-DOS, na grande maioria das vezes, era composto por um arquivo executável (.exe). Os demais arquivos necessários ao funcionamento do programa, tais como imagens gráficas ou arquivos de dados, eram salvos, normalmente, no mesmo diretório (na época do MS-DOS não falávamos em Pastas) do arquivo executável e tudo funcionava perfeitamente bem.

Para instalar o programa em outro computador, bastava copiar o diretório do programa e pronto, nada mais precisava ser feito. Podemos ver que o processo de instalação era bastante simplificado, mas em contrapartida o desenvolvimento era todo manual e as funcionalidades bastante limitadas. As redes locais ainda não eram realidade e a grande maioria dos programas era feito para trabalhar em um único computador, acessando dados locais.

Comunicação entre programas, reaproveitamento de código e acesso via rede eram coisas raras ou inexistentes. Para as necessidades da época, era um modelo plenamente satisfatório. A maior prova disso é que, ainda hoje, facilmente encontramos programas feitos em Clipper, para o ambiente MS-DOS, rodando em pequenos estabelecimentos, dando suporte a todas as operações diárias. Tente você propor que estas pessoas substituam seus programas em Clipper, que atendam perfeitamente às necessidades destes pequenos estabelecimentos, por programas para Windows com interface gráfica. Com certeza você receberá um sonoro “NÃO”.

Porém logo as redes começariam a invadir as empresas, os programas a tornar-se mais complexos. Com o advento das redes e do sucesso dos PCs, mais e mais pessoas começaram a utilizar computadores. A velha interface a caracter do MS-DOS não atendia mais as necessidades. Neste momento é que o Windows começa a surgir no mercado.

Prazer. Eu Sou o Windows!

Com a chegada do Windows tivemos a popularização das interfaces gráficas e de termos como: mouse, ícone, atalho e janelas. As ferramentas para desenvolvimento no ambiente Windows custaram a chegar. No início, a programação tinha que ser feita em linguagem C, utilizando as APIs (Application Program Interface) do Windows.

Com o lançamento do Visual Basic 1.0 teve início a era das ferramentas gráficas para desenvolvimento de programas para o Ambiente Windows.

Com o Windows já passou a existir o conceito de Instalar o programa. A instalação do programa não se limitava a uma simples cópia de arquivos. Ao instalar um programa no Windows, o mesmo gravava uma série de informações de configuração: no Windows 3.x em arquivos .ini e no Windows 9x na Registry do Sistema Operacional. Além destas informações de configuração, partes do programa eram disponibilizadas no formato de arquivos .DLL (Dynamic Link Library). Falando assim parecia um modelo, digamos, bastante elegante. Um local centralizado para informações sobre configuração e o programa dividido em partes menores, que no conjunto forneciam a funcionalidade do programa.

Porém o modelo de programação para o Windows começou a apresentar uma série de inconvenientes. Vamos falar destes inconvenientes, através de exemplos:

- ◆ Se alguma das configurações, necessárias ao funcionamento do programa, fosse alterada, o programa deixava de funcionar. Isto gerava uma chamada ao pessoal de suporte que, na maioria das vezes, somente conseguia resolver a situação reinstalando o programa.
- ◆ Os arquivos .DLL poderiam ser utilizados por mais do que um programa. Pode acontecer uma situação em que um programa que esteja sendo instalado substitua uma determinada .DLL por uma versão mais nova do que a versão da .DLL atualmente existente no sistema. O problema é que podem existir programas que dependam da versão mais antiga. Nesta situação os programas que dependem da versão mais antiga simplesmente deixarão de funcionar. Pode também acontecer o contrário, ou seja, um programa que está sendo instalado substitui uma .DLL por uma versão mais antiga, fazendo com que outros programas deixem de funcionar. Em situações mais críticas poderia acontecer de o programa que está sendo instalado substituir uma .DLL vital para o Windows. Nestas situações todo o sistema deixaria de funcionar e, em alguns casos, somente uma reinstalação poderia resolver o problema.

Vejam que o que parecia uma boa solução acabou se mostrando um verdadeiro pesadelo para gerenciar e manter em funcionamento. Nesta época surge, inclusive, a expressão “DLL Hell”, que poderíamos traduzir por: “O Inferno das DLLs.”

- ◆ Cada vez que um programa fosse alterado, o mesmo precisaria ser reinstalado em todos os computadores onde fosse necessária a nova versão. E se, ao fazer a atualização para a nova versão, fosse substituída alguma .DLL necessária ao funcionamento de algum outro programa? Novos problemas para o pessoal de suporte. Vejam que este modelo gera uma grande carga de suporte, o que encarece muito a manutenção em funcionamento de uma estação de trabalho da empresa.

Redes e Internet – Mais Problemas (ou Soluções) à Vista!

Com o advento das redes como uma realidade nas empresas e com a explosão da Internet como uma plataforma viável para fazer negócios, os modelos de desenvolvimento de aplicações sofreram profundas mudanças.

Primeiro, com as redes, foi a época da “febre” em descentralizar as estruturas de TI e migrar para o modelo Cliente/Servidor, baseado em redes locais. Neste modelo, também conhecido como modelo de duas camadas, temos um ou mais equipamentos de maior capacidade de processamento, atuando como Servidores. Nas estações de trabalho dos usuários, conhecidas como clientes, são instalados programas, que fazem acesso a recursos residentes nos servidores. O exemplo mais típico de aplicação Cliente/Servidor seria uma aplicação desenvolvida em Visual Basic ou Delphi, a qual acessa dados de um servidor SQL Server 2000, instalado em um servidor da rede. Na Figura 1.4, temos uma visão geral do modelo Cliente/Servidor.

Vamos falar um pouco mais sobre o modelo de duas camadas.

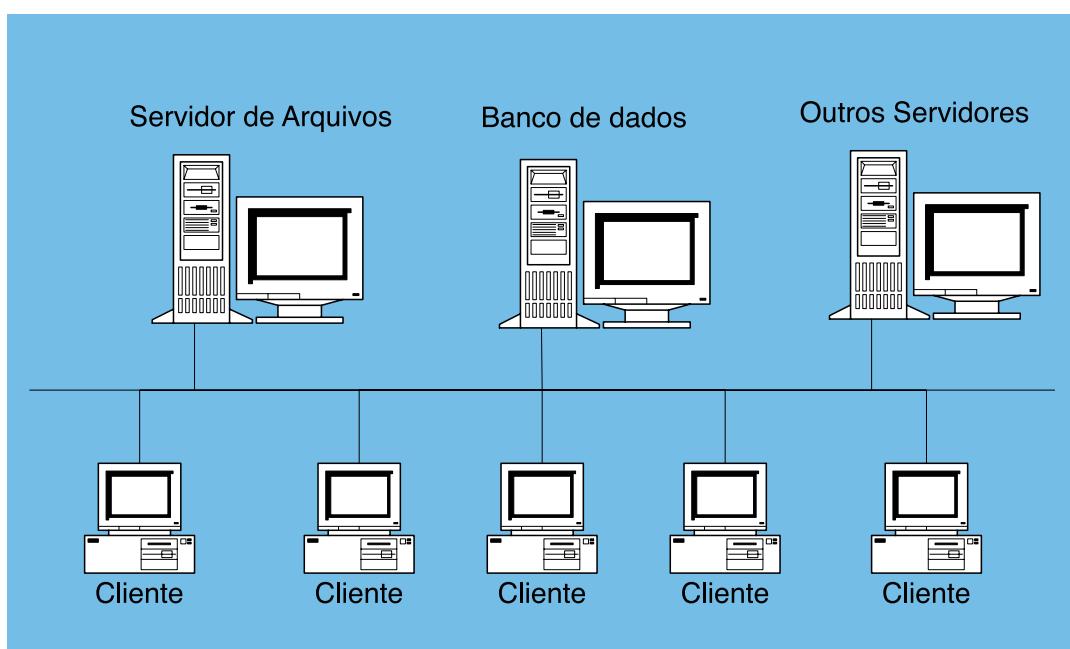
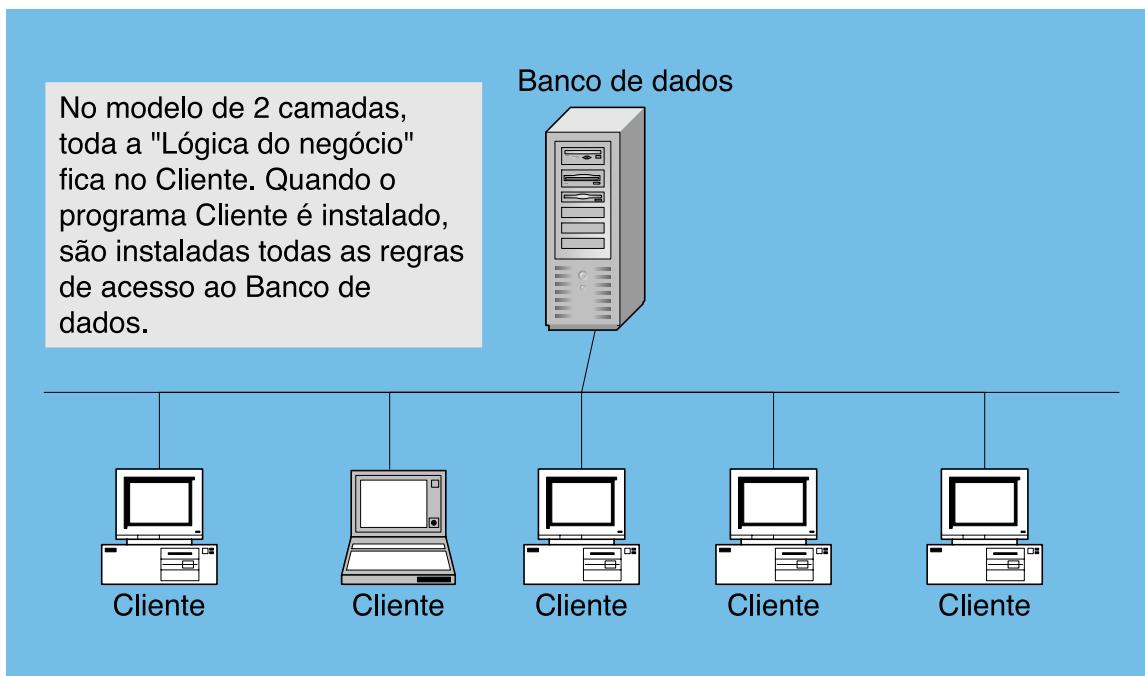


Figura 1.4: O modelo Cliente/Servidor.

Modelo em 2 Camadas

No início da utilização do modelo Cliente/Servidor, as aplicações foram desenvolvidas utilizando-se um modelo de desenvolvimento em duas camadas. Neste modelo, um programa, normalmente desenvolvido em um ambiente de desenvolvimento, como o Visual Basic, Delphi ou Power Builder, é instalado em cada estação de trabalho – Cliente. Este programa acessa dados em um servidor de banco de dados, conforme ilustrado na Figura 1.5.

Neste modelo, temos um programa que é instalado no Cliente. Programa esse que faz acesso ao banco de dados que fica residente no Servidor de Banco de dados. Na maioria dos casos, a máquina do cliente é um PC rodando Windows, e a aplicação Cliente é desenvolvida utilizando-se um dos ambientes conhecidos, conforme citado anteriormente.

**Figura 1.5: O Modelo de desenvolvimento em duas camadas.**

Sendo a aplicação cliente um programa para Windows (na grande maioria dos casos), a mesma deve ser instalada em cada uma das estações de trabalho da rede. É o processo de instalação normal, para qualquer aplicação Windows. No modelo de 2 camadas, a aplicação Cliente é responsável pelas seguintes funções:

- ◆ **Apresentação:** O código que gera a interface visível do programa faz parte da aplicação cliente. Todos os formulários, menus e demais elementos visuais estão contidos no código da aplicação cliente. Caso sejam necessárias alterações na interface do programa, faz-se necessária a geração de uma nova versão do programa, e todas as estações de trabalho que possuem a versão anterior devem receber a nova versão, para que o usuário possa ter acesso às alterações da interface. Aí que começam a surgir os problemas no modelo em 2 camadas: uma simples alteração de interface é suficiente para gerar a necessidade de atualizar a aplicação, em centenas ou milhares de estações de trabalho. O gerenciamento desta tarefa é algo extremamente complexo e oneroso financeiramente.
- ◆ **Lógica do Negócio:** As regras que definem a maneira como os dados serão acessados e processados são conhecidas como "Lógica do Negócio". Fazem parte das Regras do Negócio, desde funções simples de validação da entrada de dados, como o cálculo do dígito verificador de um CPF, até funções mais complexas, como descontos escalonados para os maiores clientes, de acordo com o volume da compra. Questões relativas a legislação fiscal e escrita contábil também fazem parte da Lógica do Negócio. Por exemplo, um programa para gerência de Recursos Humanos, desenvolvido para a legislação dos EUA, não pode ser utilizado, sem modificações, por uma empresa brasileira. Isso acontece porque a legislação dos EUA é diferente da legislação brasileira. Em síntese, as regras para o sistema de recursos humanos são diferentes. Alterações nas regras do negócio são bastante freqüentes, ainda mais com as repetidas mudanças na legislação do nosso país. Com isso, faz-se necessária a geração de uma nova versão do programa, cada vez que uma determinada regra muda, ou quando regras forem acrescentadas ou retiradas. Desta forma, todas as estações de trabalho que possuem a versão anterior devem receber a nova versão, para que o usuário possa ter acesso às alterações. Agora temos mais um sério problema no modelo de 2 camadas: qualquer alteração nas regras do negócio é suficiente para

gerar a necessidade de atualizar a aplicação, em centenas ou milhares de computadores. O que já era complicado piorou um pouco mais.

Com a evolução do mercado e as alterações da legislação, mudanças nas regras do negócio são bastante freqüentes. Com isso, o modelo de duas camadas demonstrou-se de difícil manutenção e gerenciamento, além de apresentar um TCO – Total Cost Ownership (Custo Total de Propriedade) bastante elevado.

A outra camada, no modelo de 2 camadas, é o Banco de dados, o qual fica armazenado no Servidor de banco de dados.

Sempre que um determinado modelo apresenta problemas, aparentemente intransponíveis, a indústria de informática parte para a criação de novos modelos. Em busca de soluções para os problemas do modelo de duas camadas é que surgiu a proposta do modelo de 3 camadas, conforme analisaremos a seguir.

Aplicações em 3 Camadas

Como uma evolução do modelo de 2 camadas, surge o modelo de três camadas. A idéia básica do modelo de 3 camadas é retirar as Regras do Negócio, da aplicação Cliente e centralizá-las em um determinado ponto, o qual é chamado de Servidor de Aplicações. O acesso ao banco de dados é feito através das regras contidas no Servidor de Aplicações. Ao centralizar as Regras do Negócio em um único ponto, fica muito mais fácil a atualização das mesmas. A Figura 1.6 nos dá uma visão geral do modelo em 3 camadas:

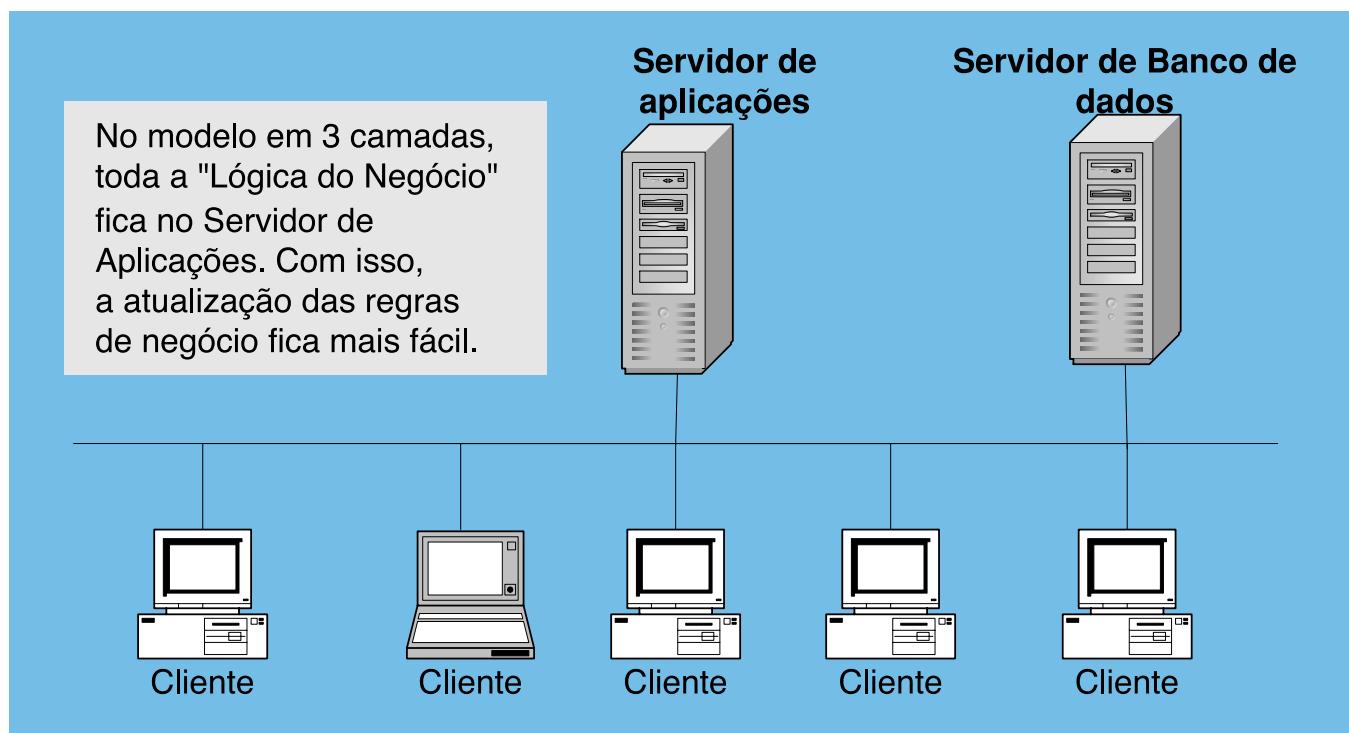


Figura 1.6: O modelo de desenvolvimento em três camadas.

Todo o acesso do cliente, aos dados do servidor de Banco de dados, é feito de acordo com as regras contidas no Servidor de Aplicações. O cliente não tem acesso aos dados do servidor de Banco de dados, sem antes passar pelo servidor de aplicações. Com isso, as três camadas são as seguintes:

- ◆ **Apresentação:** Continua no programa instalado no cliente. Alterações na Interface do programa ainda irão gerar a necessidade de atualizar a aplicação em todas as estações de trabalho, onde a aplicação estiver sendo utilizada. Porém cabe ressaltar que alterações na interface são menos freqüentes do que alterações nas regras do negócio.
- ◆ **Lógica:** São as regras do negócio, as quais determinam de que maneira os dados serão utilizados e manipulados. Esta camada foi deslocada para o Servidor de Aplicações. Desta maneira, quando uma regra do negócio for alterada, basta atualizá-la no Servidor de Aplicações. Após a atualização, todos os usuários passarão a ter acesso à nova versão, sem que seja necessário reinstalar o programa em cada um dos computadores da rede. Vejam que, ao centralizar as regras do negócio em um Servidor de Aplicações, estamos facilitando a tarefa de manter a aplicação atualizada. As coisas estão começando a melhorar.
- ◆ **Dados:** Nesta camada temos o servidor de Banco de dados, no qual reside toda a informação necessária para o funcionamento da aplicação. Cabe reforçar que os dados somente são acessados através do Servidor de Aplicação, e não diretamente pela aplicação cliente.

Com a introdução da camada de Lógica, resolvemos o problema de termos que atualizar a aplicação, em centenas ou milhares de estações de trabalho, toda vez que uma regra do negócio for alterada. Porém continuamos com o problema de atualização da interface da aplicação, cada vez que sejam necessárias mudanças na Interface. Por isso que surgiram os modelos de n-camadas.

No próximo tópico, iremos falar um pouco sobre o modelo de 4 camadas.

Aplicações em Quatro Camadas

Como uma evolução do modelo de três camadas, surge o modelo de quatro camadas. A idéia básica do modelo de 4 camadas é retirar a apresentação do cliente e centralizá-la em um determinado ponto, o qual na maioria dos casos é um servidor Web. Com isso o próprio Cliente deixa de existir como um programa que precisa ser instalado em cada computador da rede. O acesso à aplicação, é feito através de um Navegador, como, por exemplo, o Internet Explorer ou o Netscape Navigator. A Figura 1.7 nos dá uma visão geral do modelo em quatro camadas.

Para acessar a aplicação, o cliente acessa o endereço da aplicação, utilizando o seu navegador, como no exemplo: <http://intranet.minhaempresa.com/sistemas/vendas.aspx>.

Todo o acesso do cliente ao Banco de dados é feito de acordo com as regras contidas no Servidor de aplicações. O cliente não tem acesso ao Banco de dados, sem antes passar pelo Servidor de aplicações. Com isso temos as seguintes camadas:

- ◆ **Cliente:** Neste caso o Cliente é o Navegador utilizado pelo usuário, quer seja o Internet Explorer, quer seja o Netscape Navigator, ou outro navegador qualquer.
- ◆ **Apresentação:** Passa para o Servidor Web. A interface pode ser composta de páginas HTML, ASP, PHP, Flash ou qualquer outra tecnologia capaz de gerar conteúdo para o navegador. Com isso alterações na interface da aplicação são feitas diretamente no servidor Web, sendo que estas alterações estarão, automaticamente, disponíveis para todos os Clientes. Com este modelo não existe a necessidade de reinstalar a aplicação em

todos os computadores da rede. Fica muito mais fácil garantir que todos estão tendo acesso à versão mais atualizada da aplicação. A única coisa que o cliente precisa ter instalado na sua máquina é o navegador.

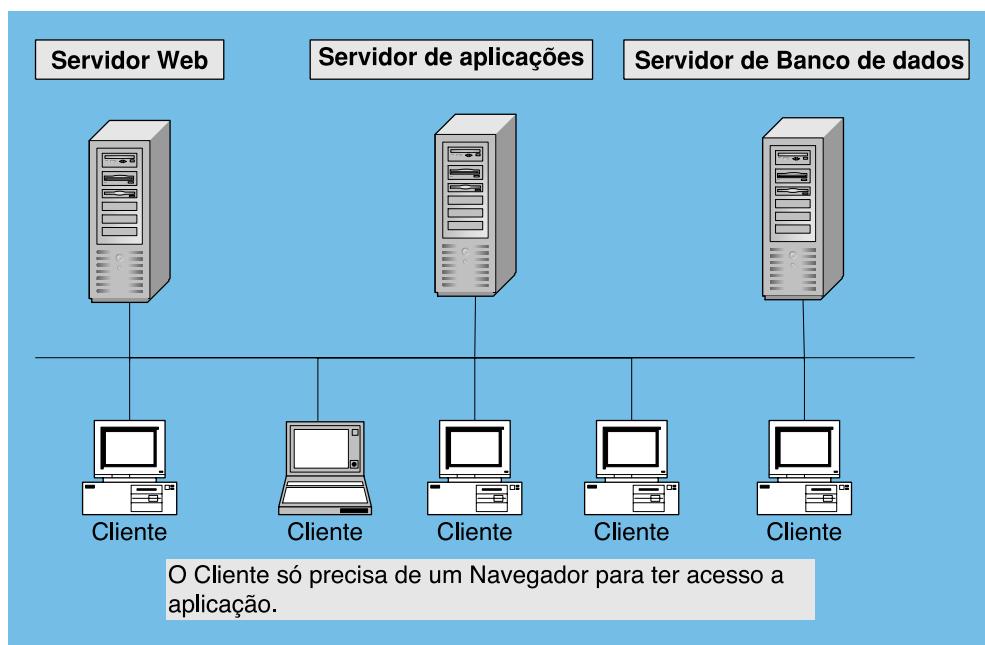


Figura 1.7: O modelo de desenvolvimento em quatro camadas.

- ◆ **Lógica:** São as regras do negócio, as quais determinam de que maneira os dados serão utilizados. Esta camada está no Servidor de Aplicações. Desta maneira, quando uma regra do negócio for alterada, basta atualizá-la no Servidor de Aplicações. Após a atualização, todos os usuários passarão a ter acesso à nova versão, sem que seja necessário reinstalar o programa em cada estação de trabalho da rede. Vejam que, ao centralizar as regras do negócio em um Servidor de Aplicações, estamos facilitando a tarefa de manter a aplicação atualizada.
- ◆ **Dados:** Nesta camada temos o servidor de Banco de dados, no qual reside toda a informação necessária para o funcionamento da aplicação.

Com o deslocamento da camada de apresentação para um Servidor Web, resolvemos o problema de termos que atualizar a aplicação, em centenas ou milhares de computadores, cada vez que uma interface precisar de alterações. Neste ponto a atualização das aplicações é uma tarefa mais gerenciável, muito diferente do que acontecia no caso do modelo em 2 camadas.

Os servidores de Aplicação, Web e banco de dados não precisam necessariamente ser servidores separados, isto é, uma máquina para fazer o papel de cada um dos servidores. O conceito de servidor de Aplicação, servidor Web ou servidor de Banco de dados é um conceito relacionado com a função que o servidor desempenha. Podemos ter, em um mesmo equipamento, um Servidor de aplicações, um servidor Web e um servidor de Banco de dados. Claro que questões de desempenho devem ser levadas em consideração.

Também podemos ter a funcionalidade do Servidor de Aplicações distribuída através de vários servidores, com cada servidor tendo alguns componentes que formam parte das funcionalidades da aplicação. Este modelo onde temos

componentes em diversos equipamentos é conhecido como Modelo de Aplicações Distribuídas. Também podemos colocar os componentes em mais do que um servidor para obtermos um melhor desempenho, ou redundância, no caso de um servidor falhar.

Questões a Considerarmos nos Modelos de 3 ou Mais Camadas Utilizados Atualmente

Muitas são as vantagens dos modelos de 3 ou mais camadas, em relação à facilidade de gerenciamento e atualização das aplicações. Mas, se tudo funciona tão bem, por que precisamos de um novo modelo de programação – leia-se Framework .NET?

Para responder a pergunta anterior, vamos continuar colocando alguns problemas com o modelo de desenvolvimento em uso atualmente, que é o modelo de n-camadas, com aplicações que utilizam componentes distribuídos através de diversos servidores: Servidor Web, de aplicações, de Banco de dados, etc.

O modelo de programação atual, para o ambiente Windows, é fortemente baseado no conceito de componentes. No exemplo do modelo de 4 camadas, quando uma aplicação cliente acessa uma regra de negócio, esta regra de negócio é implementada na forma de um componente COM/COM+. A regra de negócio pode utilizar um outro componente para fazer a conexão com o Banco de dados e retornar os dados solicitados pela aplicação. Para que os diversos componentes possam comunicar-se e trocar informações, os mesmos precisam de um padrão para a troca de mensagens. O padrão determina a estrutura interna do componente e a maneira como cada componente expõe suas funcionalidades. O padrão para o ambiente Windows é chamado, a partir do Windows 2000, de COM+. Para o Windows NT 4.0 e versões anteriores, tínhamos o COM/DCOM trabalhando em conjunto com o MTS – Microsoft Transaction Server.

Mas se existe um padrão para troca de mensagens qual é o problema? Acontece que a implementação de componentes utilizando COM/COM+ não é das tarefas mais simples. Se utilizarmos linguagens como o Visual C++, a criação e disponibilização de componentes é uma tarefa que exige programadores altamente especializados. Já com ferramentas como o Visual Basic e Delphi, a criação de componentes para o padrão COM+ é um pouco mais fácil. Porém a grande dificuldade é fazer com que componentes implementados em diferentes linguagens sejam capazes de trabalhar em conjunto e trocar mensagens entre si.

Agora imagine as dificuldades em um ambiente em que devemos criar aplicações para a Internet, onde deve existir uma maneira padronizada para que os diversos componentes sejam capazes de se comunicar. A Microsoft, com a sua iniciativa DNA, procurou disponibilizar informações para que seja possível tirar o máximo do modelo COM+, na criação de aplicações em n-camadas para a Internet. Porém a própria Microsoft reconheceu as limitações e dificuldades deste modelo, ao propor uma nova revolução nos métodos e práticas de desenvolvimento, revolução essa que atende pelo singelo nome de .NET.

Outro fator importante a considerar é que o padrão COM/COM+ é um padrão proprietário, desenvolvido pela Microsoft. Como fazer com que um padrão proprietário possa ser utilizado, sem maiores problemas para aplicações distribuídas na Internet? Fica muito difícil, para não dizer impraticável. Já com o .NET, conforme comentamos anteriormente, utiliza-se padrões não proprietários, como XML para os dados e SOAP sobre HTTP como protocolo de transporte. Desta forma, a comunicação entre Web Services acontece de uma maneira fácil, sem maiores problemas.

Para que um componente COM/COM+ possa ser utilizado, o mesmo precisa ser registrado no servidor que irá disponibilizar o componente para isso. Ao registrar o componente, são gravadas informações sobre o mesmo, na registry do Sistema Operacional. O programador precisa preocupar-se em garantir que o componente seja registrado corretamente, pois, caso contrário, o mesmo não poderá ser acessado. Já os serviços do .NET não necessitam de registro, sendo que toda a informação necessária para que os mesmos funcionem está contida no próprio serviço, no formato de metadados – Metadata. Mais adiante falaremos um pouco mais sobre Metadata.

Com componentes COM/COM+, o programador precisa preocupar-se em carregar o componente na memória, retirar o componente da memória quando o mesmo não for mais utilizado e uma série de outras funções necessárias ao funcionamento do componente. Com o .NET, todas estas “preocupações” foram transferidas para o Framework .NET. Isto faz com que o programador somente tenha que codificar a funcionalidade do serviço que está sendo desenvolvido, a parte mais, digamos assim, “chata”, será de responsabilidade do Framework .NET, mais especificamente do CLR. Isso aumenta a produtividade do programador e evita erros mais graves, os quais normalmente fazem com que o componente não funcione corretamente.

Já apresentamos os principais problemas do modelo atual; no restante deste capítulo veremos os demais elementos que compõem o Framework .NET e como cada um destes elementos procura solucionar problemas que os modelos anteriores não foram capazes de resolver.

De Volta ao Framework .NET: os Demais Elementos

Agora vamos falar um pouco mais sobre os diversos aspectos do Framework .NET e como cada um destes aspectos se propõe a solucionar problemas do modelo atual de desenvolvimento.

Linguagens Habilitadas ao .NET

O que significa uma linguagem ser “habilitada ao .NET”?

Significa que a linguagem é capaz de gerar, como resultado da compilação, código MSIL. Uma vez gerado o código MSIL, o mesmo é executado sob o controle do CLR. Ou seja, para o Framework .NET, uma vez gerado o código MSIL, não importa a partir de qual linguagem o mesmo foi gerado, pois a maneira como o mesmo é executado é sempre a mesma.

Juntamente com o Framework .NET, a Microsoft disponibiliza as seguintes linguagens:

- ◆ Visual Basic .NET (VB.NET)
- ◆ C++
- ◆ C#

Podemos criar nossos programas em qualquer uma destas linguagens. Ao compilarmos o programa, o resultado será código MSIL, o qual passa a ser executado pelo CLR. Quando a Inprise lançar o Delphi habilitado ao .NET, teremos

a opção de compilar um programa Delphi para gerar código MSIL. Uma vez gerado o código MSIL, para o CLR não importa em qual linguagem o programa foi codificado.

Mais adiante, nos capítulos 3, 4 e 5 estaremos apresentando uma introdução aos principais elementos da linguagem C#. Nos capítulos restantes do livro, estaremos utilizando a linguagem C# para a criação de páginas ASP.NET.

Um Rápida Apresentação do VB.NET

Com exceção do C# que é uma nova linguagem, o VB.NET é a linguagem que mais sofreu modificações em relação às versões anteriores. Foram introduzidas algumas características há muito tempo esperadas, tais como:

- ◆ Foram adicionadas mais característica de orientação a objetos, tais como herança verdadeira, construtores parametrizados e overriding de métodos e propriedades.
- ◆ Mais facilidades para o desenvolvimento de aplicações Internet. Por exemplo, para criar páginas ASP.NET, ao invés de VBScript utilizamos diretamente o VB.NET ou o C#.
- ◆ Herança visual: Esta é uma características que os desenvolvedores do Delphi já tinham há algum tempo. Com o VB.NET podemos criar um formulário com as características básicas para todos os formulários da aplicação. Quando precisarmos de um formulário com características adicionais, podemos construir este novo formulário baseado, isto é, herdando as características do formulário básico; depois é só adicionar os elementos necessários. Ao criarmos o nosso formulário, herdando do formulário básico, o formulário que está sendo criado terá, automaticamente, todos os elementos do formulário básico. Além dos elementos gráficos, o código para tratamento de eventos também é herdado. O melhor de tudo é que, ao alterarmos o formulário básico, todas as alterações são refletidas nos formulários herdados. A herança visual facilita, enormemente, a criação de aplicações que tenham uma interface consistente e de fácil alteração.
- ◆ Tratamento estruturado de exceções: com o VB.NET, temos muito mais recursos para o tratamento de erros e exceções do que o bom e velho On Error do Visual Basic. Isso faz com que possamos criar programas mais confiáveis.

Falar sobre as novidades do VB.NET é assunto para um capítulo inteiro. Ensinar o VB.NET: assunto para um livro inteiro. Em um trabalho futuro estaremos tratando desta nova e excitante versão do VB.

Uma Rápida Apresentação do C#

O C# é a nova linguagem da Microsoft, apresentada juntamente com o Framework .NET. O C# foi construído com base nos conceitos de Orientação a objetos. As diretivas básicas para a criação do C# foram as seguintes:

- ◆ Uma linguagem orientada a objetos e tão poderosa quanto o C++.
- ◆ Uma linguagem tão fácil de utilizar quanto o VB.

O pessoal da Microsoft realmente conseguiu aliar estas duas características em uma só linguagem: Poder e Simplicidade. Conforme veremos nos capítulos 3, 4 e 5, o C# é de fácil aprendizagem e utilização, ao mesmo tempo que nos fornece poderosos mecanismos, antes só encontrados no C++.

Como não poderia deixar de ser, vamos dar o tradicional exemplo do Hello World !!!, utilizando o C#. Apresentarei o código e os passos para compilar e rodar o nosso primeiro programa em C#. Nos demais capítulos deste livro aprenderemos mais sobre esta linguagem. Para criar o nosso primeiro programa, utilizaremos o Bloco de Notas do Windows.

Abra o Bloco de Notas e digite o texto indicado na Listagem 1.1.

Listagem 1.1 – Hello World !

```
using System;
class primeiroprograma
{
    // Meu primeiro programa em C#
    // O tradicional Hello World !!

    public static void Main()
    {
        string umamensagem = "Hello World !!!";
        Console.WriteLine(umamensagem);
    }
}
```

Salve este programa no disco com o nome de primeiroprograma.cs. Agora vamos compilar o programa e executá-lo.

Para compilar o programa, abra um Prompt de Comando (Iniciar -> Programas -> Acessórios -> Prompt de comando). Na janela do prompt vá para a pasta onde você salvou o arquivo hello.cs e execute o seguinte comando:

```
csc primeiroprograma.cs
```

Com este comando estamos compilando o arquivo-fonte primeiroprograma.cs, para gerar um executável .exe. O resultado deste comando é indicado na Figura 1.8, onde utilizamos o comando dir, para mostrar que a compilação gerou o executável primeiroprograma.exe.

Para executar o nosso programa, basta digitar primeiroprograma e pressionar Enter. O programa será executado, conforme indicado na Figura 1.9.

Bem, por enquanto é isto, fizemos apenas uma pequena apresentação da linguagem C#. Teremos três capítulos deste livro (3, 4 e 5), para tratar desta linguagem. Apenas um último detalhe para o qual gostaria de chamar a atenção: Observe, para aqueles que conhecem Java ou C++, a semelhança entre a estrutura de um programa em C# e um programa em uma destas linguagens.

```
C:\Exemplos CSharp\Capitulo01>csc primeiroprograma.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\Exemplos CSharp\Capitulo01>dir
O volume na unidade C é PRINCIPAL
O número de série do volume é 2629-07E6

Pasta de C:\Exemplos CSharp\Capitulo01
07/07/2001  17:09      <DIR>          .
07/07/2001  17:09      <DIR>          ..
07/07/2001  17:17              223 Hello.cs
07/07/2001  17:19              223 primeiroprogramma.cs
07/07/2001  17:19            3.072 primeiroprogramma.exe
                           3 arquivo(s)    3.518 bytes
                           2 pasta(s)  1.531.035.648 bytes disponíveis

C:\Exemplos CSharp\Capitulo01>_
```

Figura 1.8: Compilando o primeiro programa em C#.

```
C:\Exemplos CSharp\Capitulo01>primeiroprogramma
Hello World !!!
C:\Exemplos CSharp\Capitulo01>_
```

Figura 1.9: Executando o primeiro programa em C#.

Common Type System

O Framework .NET disponibiliza, na forma de objetos, um conjunto de tipos de dados comuns, os quais podem ser utilizados por todas as linguagens habilitadas ao .NET. Isso significa que uma variável do tipo Long terá a mesma estrutura e ocupará o mesmo número de bytes, quer seja no VB.NET, no C#, no C++ ou em qualquer outra linguagem habilitada ao .NET. Este conjunto de tipos comuns, que pode ser utilizado por qualquer linguagem, é chamado de Common Type System, que a partir de agora abreviaremos por CTS.

Dentre outras coisas, é o CTS que facilita a integração entre os programas e serviços criados, utilizando-se de diferentes linguagens do .NET. No modelo antigo, uma das dificuldades de fazer com que um Componente COM+ criado com o Visual C++ pudesse ser utilizado por um programa escrito em Visual Basic é que as linguagens possuíam um diferente conjunto de tipos básicos. Para que os componentes, escritos em diferentes linguagens, pudessem se comunicar,

o programador tinha que mapear os tipos de uma linguagem, para os tipos correspondentes em outra linguagem, fazendo as conversões necessárias. Vejam o quanto este procedimento é trabalhoso. Com o CTS do .NET simplesmente esta preocupação não existe, uma vez que todas as linguagens têm acesso a um conjunto de tipos comum a todas elas.

Conforme descrito na documentação do Framework .NET, são as seguintes as principais funções do CTS:

- ◆ Fornece uma estrutura que possibilita a integração entre diferentes linguagens habilitadas ao .NET, com uma execução mais rápida, uma vez que a sobrecarga para a conversão entre os diferentes tipos de diferentes linguagens deixa de existir.
- ◆ Fornece uma estrutura de tipos com base em um modelo orientado a objetos, o que facilita a criação de novas linguagens habilitadas ao .NET, favorecendo a utilização de boas práticas de programação, como por exemplo a herança. Uma vez que os tipos são objetos, podemos criar tipos derivados dos objetos básicos, os quais herdam todas as características dos objetos básicos.
- ◆ O CTS define algumas regras que toda linguagem deve seguir, para ser habilitada ao .NET. Por seguirem um conjunto de regras comum, a interação entre programas escritos em diferentes linguagens fica bem mais fácil.

No Capítulo 2, quando detalharemos o CLR, vamos detalhar um pouco mais sobre o CTS.

Metadata

Na seção “Questões a considerarmos nos modelos de 3 ou mais camadas utilizados atualmente”, descrevemos, como uma das inconveniências do modelo de componentes baseados em COM/COM+, o fato de que os mesmos, para poderem ser utilizados, precisam ser registrados. Ao registrarmos um componente COM/COM+, uma série de informações sobre o mesmo são gravadas na Registry do sistema. Estas informações são utilizadas pelos programas que precisam acessar o componente. Se alguma destas informações estiver errada ou tiver sido alterada, o componente não poderá ser acessado e os programas que utilizam o componente deixarão de funcionar corretamente.

No Framework .NET, para utilizarmos os componentes .NET (.NET Componentes), não é necessário que os mesmos sejam registrados. O próprio componente .NET possui todas as informações necessárias ao seu funcionamento, bem como as informações necessárias para que outros aplicativos possam utilizá-los. Estas informações, que fazem parte do próprio componente .NET ficam gravadas no arquivo que compõe componente, na forma de Metadata. Uma tradução “popularmente conhecida” para Metadata seria: dados sobre dados. No nosso caso, Metadata seriam as informações que o componente .NET possui a respeito de si mesmo, informações estas que podem ser utilizadas por outros componentes e serviços, para acessar o componente em questão.

Além de fazer com que não seja necessário o registro do componente, as informações de Metadata facilitam a interoperabilidade entre diferentes componentes, mesmo entre componentes escritos em diferentes linguagens. Estas informações são geradas, automaticamente, no momento da compilação do componente e são gravadas no arquivo .DLL ou .EXE do componente. São muitas as informações que podem ser inseridas no componente, na forma de Metadata, tais como:

- ◆ Nome e versão do componente.

- ◆ Uma chave pública para verificação da origem e da autenticidade do componente.
- ◆ Informações sobre todas as classes ou componentes, dos quais o componente depende para funcionar corretamente.
- ◆ Tipos disponibilizados (exportados) pelo componente.
- ◆ Permissões de segurança para o acesso ao componente, seus métodos e propriedades.
- ◆ Classes básicas e interfaces do Framework .NET, utilizadas pelo componente.
- ◆ Atributos personalizados, implementados no componente.
- ◆ Membros do componente (métodos, campos, propriedades, eventos, etc).

Quando o componente é acessado, o CLR carrega os metadados do componente na memória e faz referência a estes metadados, para obter informações sobre as classes, membros, herança e dependências do componente.

São diversos os benefícios do uso de metadados, dentre os quais podemos destacar os seguintes:

- ◆ Módulos de código auto descriptivos: O próprio componente contém toda a informação necessária para interagir com outros componentes. Desta forma o componente pode ser implementado como um único arquivo, o qual contém a sua definição (na forma de metadados) e a sua implementação, o código do componente.
- ◆ Nunca é demais repetir: A utilização de metadados facilita, enormemente, a interoperabilidade entre componentes criados usando diferentes linguagens.

Assemblies

Uma aplicação .NET é constituída de um conjunto de “blocos” chamados Assembly. Através dos Assemblies é que podemos controlar a distribuição de uma aplicação, fazer o controle de versões, além de definir as configurações de segurança. Um assembly é uma coleção de tipos e recursos que foram construídos para trabalharem juntos, formando, com isso, uma unidade com funcionalidade e escopos bem definidos. Um assembly fornece ao CLR importantes informações sobre a implementação de tipos da aplicação. Para o CLR, um tipo somente existe no contexto de um assembly. De uma maneira mais simples, poderíamos dizer que um assembly é o mecanismo utilizado pelo .NET, para “empacotar” todos os elementos e informações necessárias ao funcionamento de uma aplicação ou componente. Vamos simplificar mais ainda: o assembly é uma maneira de juntar e organizar os diversos elementos que formam uma aplicação ou componente.

Os assemblies foram criados para simplificar a distribuição de aplicações e resolver o problema de “versões”, existentes em aplicações baseadas em componentes. Lembram que no início deste capítulo falamos sobre o termo “DLL Hell”? Com este termo estávamos nos referindo ao problema de um programa, ao ser instalado, substituir uma DLL por uma versão mais nova ou mais antiga, fazendo com que programas que dependiam da versão anterior da DLL deixassem de funcionar. Através do uso de assemblies e dos metadados contidos em cada componente, é possível que diferentes versões, do mesmo componente, estejam disponíveis, ao mesmo tempo, em um computador. Desta forma, cada programa utiliza a versão do componente para o qual o programa foi criado. Ao instalarmos uma nova versão do componente, o qual vem embutido em um assembly, as versões anteriores serão mantidas, se as mesmas estiverem sendo utilizados por outros programas. Isso faz com que o inferno das DLLs (DLL Hell) seja coisa do passado.

Para resolver o problema de versões e evitar o inferno das DLLs, o CLR utiliza assemblies da seguinte maneira:

- ◆ Permite que o desenvolvedor defina regras sobre o uso de diferentes versões entre diferentes componentes .NET.
- ◆ Fornece a infraestrutura necessária para que as regras de versão definidas pelo desenvolvedor sejam respeitadas.
- ◆ Fornece a infraestrutura necessária, para que diferentes versões de um mesmo componente de software possam rodar, simultaneamente. Esta execução simultânea é conhecida como “syde-by-syde execution.”

Um assembly é composto de dois elementos básicos:

- ◆ Manifesto.
- ◆ Um conjunto de módulos.

Interfaces com o Usuário

NOTA: Veremos mais detalhes sobre assemblies e seus elementos no Capítulo 2, onde estaremos detalhando o CLR – Common Language Runtime.

No Framework .NET a nomenclatura utilizada para representar os elementos que compõem uma aplicação Web são diferentes dos termos utilizados para representar uma aplicação tradicional para o Windows, também conhecidas como aplicações Win32. Aliás, este é um dos pontos que gostaria de destacar; o Framework .NET não foi concebido apenas para a criação de aplicações Web. Com Framework .NET, podemos criar qualquer tipo de aplicação, desde aplicações Web, passando por aplicações Win32 tradicionais, até aplicações de Console, também conhecidas como aplicações de linha de comando.

Em termos de interface com o usuário, temos dois elementos a considerar:

- ◆ Windows Forms
- ◆ Web Forms

Windows Forms

Que o Framework .NET foi todo projetado tendo em vista a Internet nos já sabemos. Porém nem todas as aplicações são ou serão desenvolvidas para a Web. A utilização de Windows Forms (Win Forms ou, se preferirem, Formulários do Windows) é o mecanismo que nos permite criar as tradicionais aplicações para Windows.

Win Forms é o novo mecanismo para construção de aplicações Windows, baseadas no Framework .NET.

Um Windows Form é bastante semelhante ao conceito de formulário utilizado pelas versões atuais do VB e do Delphi. O formulário é o elemento básico, sobre o qual adicionamos controles e código para determinados eventos associados com o formulário e seus controles. O Windows Form é tudo isso, porém com a diferença de poder utilizar todos os mecanismos do Framework .NET. Dentre os principais mecanismos disponíveis, destaca-se o mecanismo de herança, o qual é chamado, para o caso dos Win Forms, de herança visual. Anteriormente havíamos comentado sobre herança visual.

Um Win Form, como tudo no Framework .NET, é um objeto, o qual é obtido a partir da instanciação de uma classe básica. Todos os formulários no Framework .NET são baseados em uma das seguintes classes:

- ◆ System.Windows.Forms.
- ◆ São baseadas em um formulário padrão criado pelo usuário, através do mecanismo de herança.

Em resumo, o Win Form é o elemento básico de interação com o usuário; em outras palavras, o Win Form é o elemento visual das aplicações, elemento este com o qual o usuário irá trabalhar.

Na Figura 1.10, temos uma visão geral dos elementos que compõem uma aplicação Win32 típica, construída com o Framework .NET.

Web Forms

Para aplicações Win32 a interface com o usuário são os Windows Forms, vistos no item anterior. Para aplicações Web, criadas com ASP.NET, as páginas são construídas utilizando-se Web Forms.

Um dos objetivos da utilização de Web Forms é trazer para o desenvolvimento Web as facilidades de “arrastar e soltar”, existentes no desenvolvimento de aplicações tradicionais. Com isso poderemos, com o uso do Visual Studio .NET, criar páginas Web, simplesmente arrastando componentes sobre um Web Form.

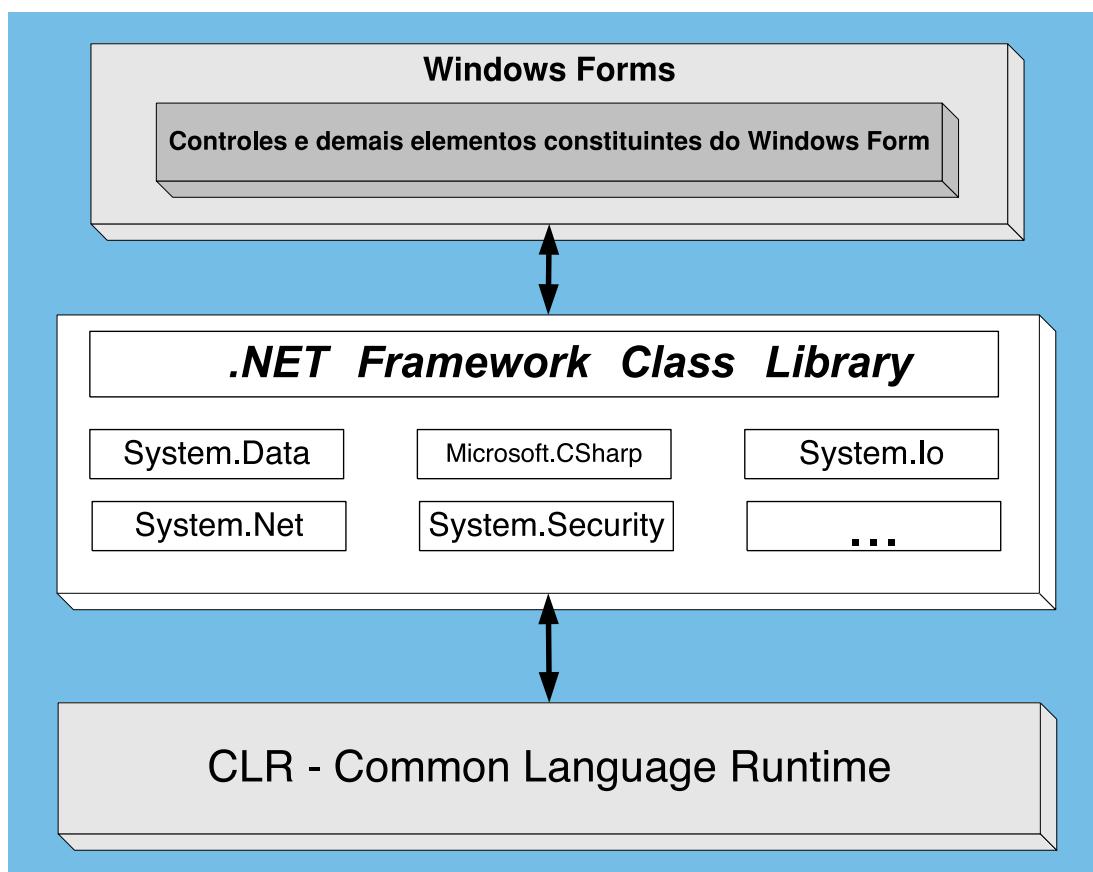


Figura 1.10: Uma aplicação Win32 com o Framework .NET.

Com o uso de Web Forms, uma página ASP.NET é dividida em dois componentes básicos:

- ◆ Um modelo (template), o qual contém os elementos visuais e de formatação da página, normalmente conteúdo baseado em HTML, ou HTML gerado pelo código ASP.NET.
- ◆ Uma seção de código, a qual é responsável por todo o processamento lógico dos elementos da página. Por exemplo, na seção de código podemos colocar os comandos necessários para estabelecer uma conexão com um banco de dados e um comando para retornar os dados que serão utilizados pela página. Também podemos colocar comandos que atualizam os dados no banco de dados, de acordo com as informações digitadas na página.

Observe que com estes dois elementos: template e seção de código, temos uma separação entre o código de processamento e o conteúdo da página propriamente dito. Isso evita a criação de códigos do tipo “macarrão”, onde temos seções de código alternadas com seções de HTML, sucessivamente. O código “macarrão” é bastante comum nas versões anteriores do ASP (ASP 2.0 e ASP 3.0).

Com o Framework .NET são disponibilizados uma série de novos controles para utilização em páginas ASP.NET. Estes novos controles apresentam novas características e propriedades que facilitam a criação de páginas, digamos, “mais inteligentes”, onde podemos inserir código em resposta a uma série de eventos. O melhor disso tudo é que o funcionamento destes controles é independente do Navegador utilizado pelo cliente, uma vez que os mesmos são processados no servidor Web.

A seção de código de um Web Form pode ser criada utilizando qualquer linguagem do Framework .NET, como por exemplo: VB.NET, C# ou Jscript.NET.

Com a utilização de Web Forms é bem mais simples manter o “estado” de uma página, entre diferentes requisições do usuário, isso tudo sem a necessidade de implementar a manutenção de estado utilizando os objetos Session e Application.

NOTA: Veremos mais detalhes sobre a criação, vantagens e os elementos de Web Forms no Capítulo 6 – Uma Introdução ao ASP.NET.

Além disso, com o uso de Web Forms podemos construir aplicações Web com interfaces bastante sofisticadas e funcionais, de uma maneira fácil através da utilização de ferramentas de desenvolvimento como o Visual Studio .NET e outras que deverão surgir para o desenvolvimento para o Framework .NET.

Na Figura 1.11, temos uma visão geral dos elementos que compõem uma aplicação Web típica, criada com ASP.NET, com a utilização de Web Forms.

Os Servidores .NET

Além do Framework .NET, a Microsoft vem disponibilizando novas versões dos servidores da família Back Office. Estas novas versões vêm sendo adaptadas para dar suporte ao modelo de desenvolvimento .NET. Por exemplo, os servidores estão sendo adaptados para o padrão XML, que é o padrão de dados para troca de informações entre componentes e serviços do Framework .NET. Outra preocupação para esta linha de servidores é com a “escalabilidade” dos mesmos. Estes servidores foram projetados para atender um grande volume de usuários, além de terem a capacidade

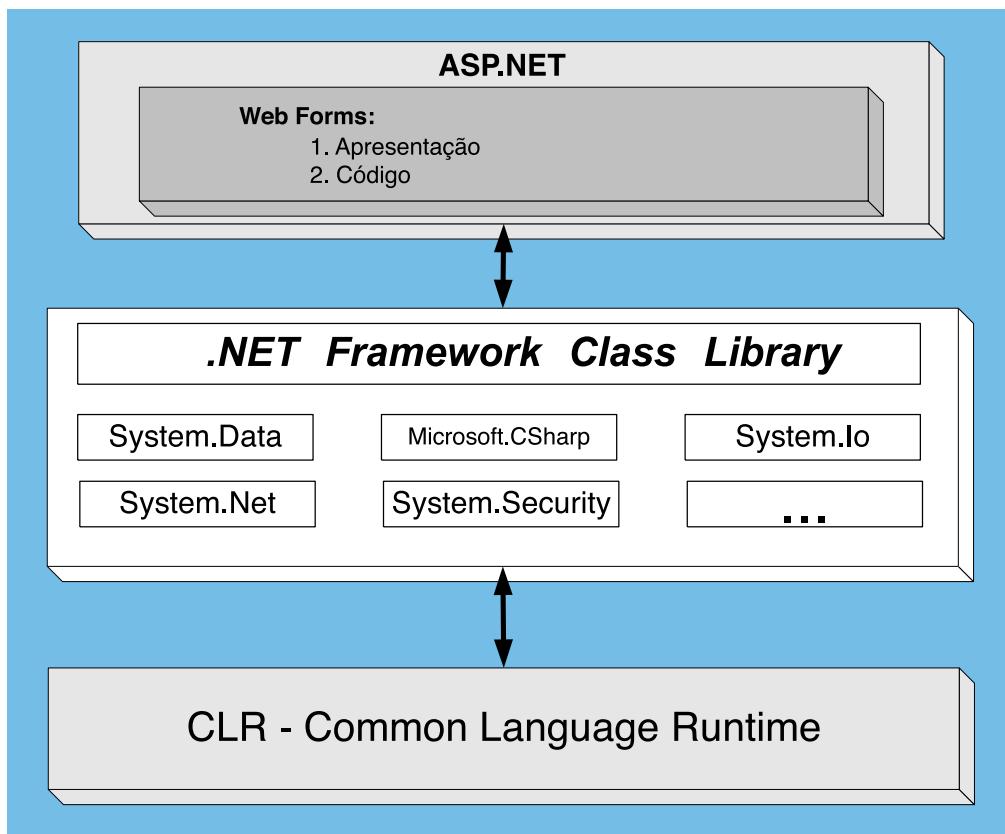


Figura 1.11: Uma aplicação Web com ASP.NET e Web Forms.

de utilizar a tecnologia de Cluster e Balanceamento de Cargas do Windows 2000. Outras diretrivas básicas para estes servidores são as seguintes:

- ◆ **Interoperabilidade:** Ao contrário do que muitos anunciam, os Mainframes continuam vivos. Além disso, o ambiente de TI das empresas é bastante heterogêneo, onde convivem diferentes sistemas operacionais e diferentes fontes e formatos de dados. O Framework .NET e os servidores .NET facilitam a tarefa de integrar dados das mais variadas fontes, desde o tradicional Mainframe, passando por fontes de dados estruturados como o SQL Server 2000 ou ORACLE e chegando a fontes não estruturadas, como mensagens de correio eletrônico, arquivos de vídeo e imagens gráficas.

NOTA: Nota: Para maiores informações sobre MMC e Snap-Ins, consulte a Unidade IV do livro: "Série Curso Básico & Rápido: Microsoft Windows 2000 Server", de minha autoria, publicado pela editora Axcel Books (www.axcel.com.br).

- ◆ **Facilidade de gerenciamento:** Dentro da filosofia da Microsoft de construir interfaces de administração intuitivas e fáceis de utilizar. Todos os servidores utilizam interfaces padronizadas, através do uso do MMC – Microsoft Management Console e Snap-Ins específicos para a administração de cada servidor.

Neste tópico descreveremos, rapidamente, quais os servidores .NET e qual a função de cada um no ambiente empresarial. Veremos uma breve descrição dos seguintes servidores:

- ◆ SQL Server 2000
- ◆ Exchange Server 2000
- ◆ BizTalk Server 2000
- ◆ Commerce Server 2000

- ◆ Application Center 2000
- ◆ Host Integration Server 2000
- ◆ Internet Security and Acceleration Server 2000
- ◆ Mobile Information 2001 Server

SQL Server 2000

O SQL Server 2000 é o servidor de banco de dados relacionais da Microsoft. Oferece funcionalidades avançadas como replicação, stored procedures, acesso a diferentes fontes de dados, múltiplas instâncias em um único servidor, mecanismo de segurança refinado e integrado com o Windows 2000, transações distribuídas, etc. Podemos acessar os dados de um servidor SQL Server 2000, no formato XML, utilizando um navegador, através do protocolo HTTP.

Para maiores informações sobre o SQL Server 2000, consulte as seguintes fontes:

- ◆ <http://www.microsoft.com/sql>
- ◆ Livro: “[SQL Server 2000 Administração e Desenvolvimento: Curso Completo](#)”, de minha autoria, publicado pela editora Axcel Books (www.axcel.com.br).

Exchange Server 2000

É um servidor de mensagens e correio eletrônico, além de uma plataforma para desenvolvimento de aplicações do Workflow. Cada vez mais o Exchange vem ganhando mercado de concorrentes como o Lotus Notes da IBM e o Novel Groupwise da Novel. O Exchange 2000 é completamente integrado com o Active Directory do 2000, o que facilita a criação e manutenção de contas de usuários. O suporte ao XML também foi introduzido nesta versão do Exchange. Maiores informações sobre Exchange podem ser encontradas nos seguintes endereços:

- ◆ <http://www.microsoft.com/exchange>
- ◆ <http://www.swynk.com>

BizTalk Server 2000

Com a consolidação do comércio eletrônico, principalmente do chamado B2B – Business to Business, que é o comércio entre empresas, cada vez faz-se mais necessária a integração entre sistemas de informação de diferentes empresas. Um dos maiores problemas é que estes diferentes sistemas de informação utilizam diferentes formatos de dados. Durante muito tempo, uma das soluções adotadas foi o EDI – Exchange Data Interchange. Porém o EDI apresenta algumas limitações, além de um custo elevado. Com o advento da Internet e do padrão XML, a troca de informações entre empresas tem migrado para soluções onde o XML é o formato universalmente aceito, o que facilita a troca de informações. O Biztalk Server 2000 é a solução da Microsoft que facilita a criação, desde o modelo conceitual até a implementação, de aplicações baseadas em XML, para troca de informações entre diferentes empresas. Maiores informações e uma versão de avaliação para download podem ser encontradas no seguinte endereço: <http://www.microsoft.com/biztalk>

Commerce Server 2000

O Commerce Server trabalha em conjunto com o IIS. Na verdade o Commerce Server facilita a criação de um site para comércio eletrônico, quer seja B2C – Business to Consumer, quer seja B2B – Business to Business. Através de uma série de modelos prontos e através da utilização de assistentes, podemos rapidamente criar um site para comércio eletrônico. Após a criação, é possível personalizar o site de acordo com as necessidades da empresa. Pode trabalhar integrado com os demais servidores .NET. Por exemplo, pode utilizar o SQL Server 2000 para armazenar informações sobre o catálogo de produtos, preços e estoque. Maiores informações e uma versão de avaliação para download podem ser encontradas no seguinte endereço: <http://www.microsoft.com/commerceserver>

Application Center 2000

O Application Center 2000 é a ferramenta da Microsoft para a implementação e gerenciamento de Web sites que deverão suportar uma elevada carga de acesso, com um grande número de acessos simultâneos. Também oferece ferramentas para a distribuição de um site entre diversos servidores, com o objetivo de distribuir a carga entre diversos equipamentos. Com o uso do Application Center fica mais fácil realizar tarefas como por exemplo manter sincronizado o conteúdo dos diversos servidores, bem como fazer o gerenciamento e a distribuição de cargas.

Host Integration Server 2000

Esta é a nova versão do antigo SNA Server da Microsoft, só que com o nome alterado. O Host Integration Server possibilita a integração de redes Windows com outros ambientes, como por exemplo, Mainframes baseados na arquitetura SNA da IBM. Maiores informações e uma versão de avaliação para download podem ser encontradas no seguinte endereço: <http://www.microsoft.com/hiserver>

Internet Security and Acceleration Server 2000

De certa maneira é o sucessor do Proxy Server 2.0 da Microsoft, com algumas melhorias. É utilizado para conectar a rede local da empresa, de uma maneira segura, à Internet. Suas funções básicas são as seguintes:

- ◆ Firewall
- ◆ Cache de páginas

Maiores informações e uma versão de avaliação para download podem ser encontradas no seguinte endereço: <http://www.microsoft.com/isaserver>

Mobile Information 2001 Server

O Framework .NET não foi concebido apenas para o desenvolvimento de aplicações que serão acessadas através de PCs ligados em rede ou computadores tradicionais. Com o Framework .NET, a Microsoft pretende fornecer uma sólida plataforma de desenvolvimento, também para os diversos dispositivos móveis existentes, tais como telefones celulares, assistentes pessoais, Palm Pilots, etc. Dentro desta estratégia, o Mobile Information 2001 Server desempenha um papel fundamental, fornecendo suporte ao protocolo WAP 1.1. Usando o Mobile Information 2001 Server será

possível, por exemplo, fazer com que as suas mensagens do Exchange sejam convertidas para o formato em que possam ser lidas por um celular ou qualquer outro dispositivo habilitado ao WAP.

Maiores informações e uma versão de avaliação para download podem ser encontradas no seguinte endereço: <http://www.microsoft.com/servers/miserver/default.htm>

ADO.NET

Com o ASP 3.0, a tecnologia preferida para o acesso a diferentes fontes de dados é a tecnologia ADO – Activex Data Objects, trabalhando em conjunto com OLE-DB, conforme ilustrado na Figura 1.12

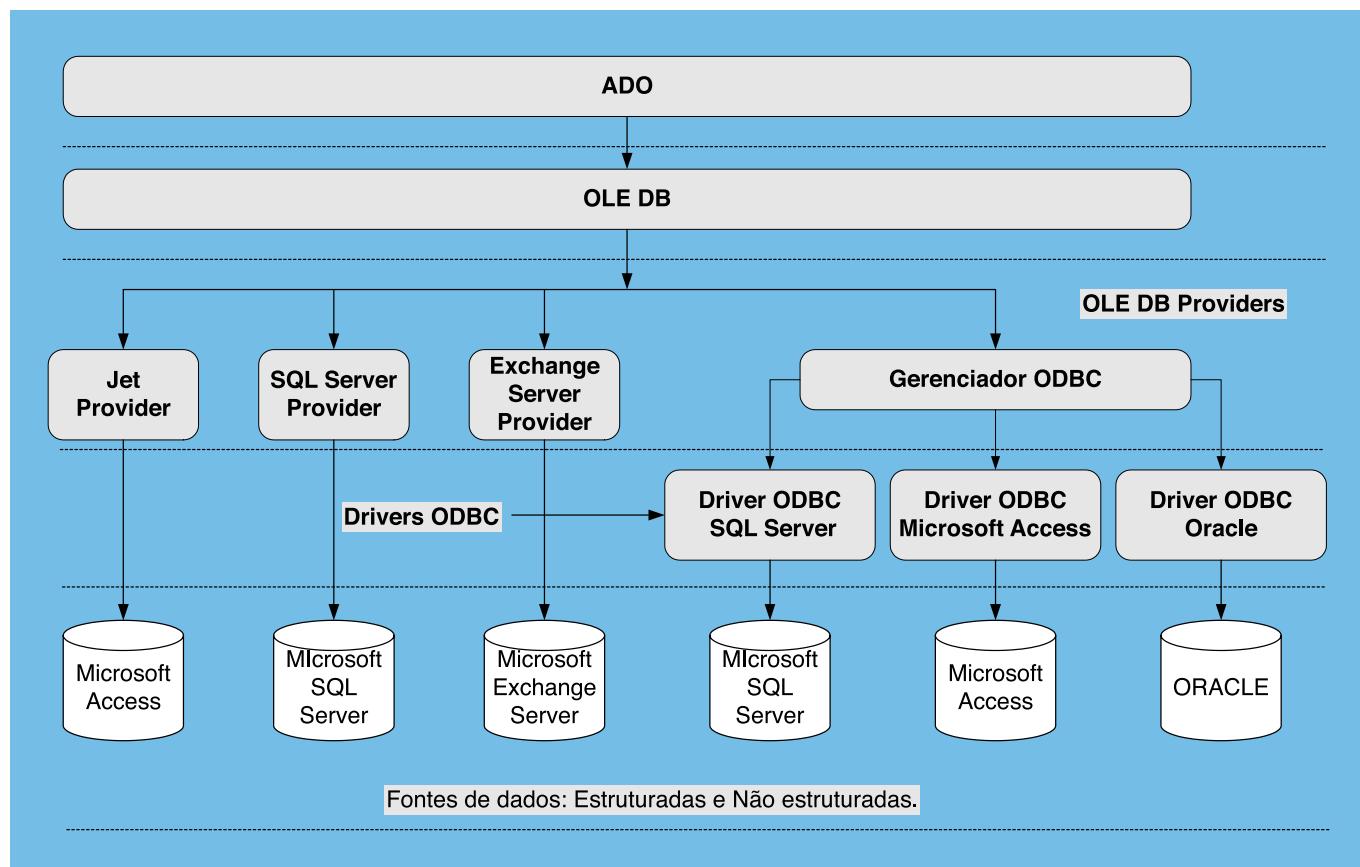


Figura 1.12: Acessando dados com ASP 3.0.

Activex Data Objects para o Framework .NET (ADO.NET) é um conjunto de classes que expõem serviços para acesso a diversas fontes e formatos de dados, para os programadores, utilizando qualquer linguagem habilitada ao .NET. Cabe ressaltar que ADO.NET não é um substituto para o ADO, pois as duas versões de componentes para acesso a dados podem continuar existindo em um mesmo servidor Web, por exemplo. Ao instalarmos o Framework .NET, o mesmo instala, além do suporte a páginas ASP.NET, todo o conjunto de classes do Framework .NET. Porém o acesso a páginas ASP tradicionais continua disponível, o que significa que, mesmo após instalar o Framework .NET, todas as aplicações criadas com ASP 3.0 ou versões anteriores do ASP continuarão funcionando sem problemas, até que você possa migrá-las para ASP.NET. Então em um mesmo servidor Web podemos ter uma página ASP, que acessa dados de

um arquivo .mdb do Access utilizando ADO, convivendo com um aplicativo criado com ASP.NET, o qual acessa informações de uma caixa de correio do Exchange, utilizando ADO.NET.

ADO.NET foi projetado pensando nos desafios de criar programas baseados no modelo de 3 camadas, para a Internet, os quais precisam acessar dados de diferentes fontes. Além disso, as aplicações Internet atuais estão bem mais sofisticadas, onde é necessário manter o conceito de estado, o que é difícil em um mundo como o da Internet, conhecido como Connection Less. Na Internet o usuário faz a conexão com o servidor, requisita a página, recebe o resultado e desfaz a conexão. Para manter o estado entre uma conexão e outra, no ASP 3.0, utilizávamos os objetos Session e Application. Conforme veremos nos exemplos práticos, a partir do Capítulo 6, manter o estado ficou muito mais simples com ASP.NET e ADO.NET.

Outro fator importante a ser considerado é que, cada vez mais, as aplicações estão utilizando dados no padrão XML. ADO.NET fornece meios de utilizar dados no padrão XML.

Conforme veremos, os objetos e métodos de ADO.NET são bastante diferentes dos utilizados com ADO. Um dos primeiros objetos de que “sentiremos falta” é o tradicional Recordset. Na Figura 1.13 temos uma visão geral da arquitetura do ADO.NET.

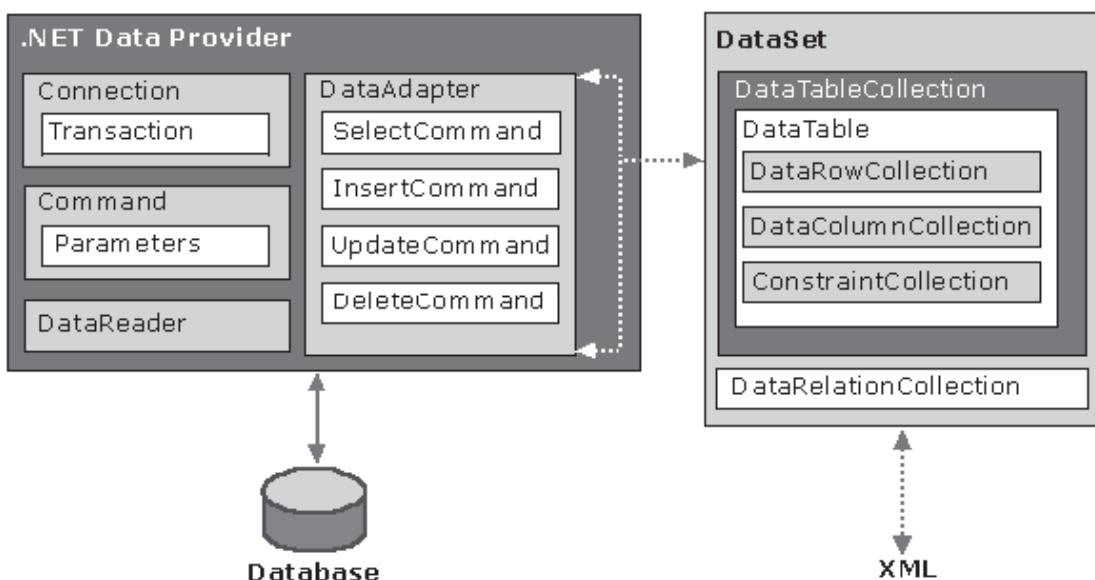


Figura 1.13: Uma visão geral da arquitetura do ADO.NET.

NOTA: Esta figura foi
retirada da documentação
do Framework .NET.

Observe que não temos o tradicional objeto Recordset.

Para trabalharmos com ADO.NET, utilizaremos as seguintes classes da biblioteca de classes do Framework .NET (.NET Framework Class Library).

Ao longo deste livro, principalmente nos Capítulos 10 e 11, estaremos utilizando ADO.NET em diversos exemplos de conexão de páginas ASP.NET com fontes diversas de dados.

Conclusão

Neste capítulo “apresentamos” o .NET ao amigo leitor.

Iniciamos o capítulo apresentando uma visão geral e uma definição genérica para a iniciativa .NET da Microsoft. Tratamos de conceitos como Web Services, CLR – Common Language Runtime e .NET Framework Class Library. Vimos que o .NET é a visão da Microsoft em um mundo onde o Software passa a ser desenvolvido e comercializado na forma de serviços. Os programas são construídos a partir de unidades menores, os assemblies. Dentro de um assembly temos todos os elementos que garantem uma funcionalidade específica, para a qual o assembly foi projetado. Dentro deste contexto podemos concluir que, mais do que uma atualização tecnológica, o .NET é uma mudança de Paradigma.

Destacamos os dois principais componentes do Framework .NET. Em primeiro lugar uma ambiente de execução comum a todas as linguagens habilitadas ao .NET – CLR – Common Language Runtime. Qualquer linguagem habilitada ao .NET, ao ser compilada, gera código MSIL – Microsoft Intermediate Language, sendo que é o código MSIL que é executado pelo CLR.

Além disso, é fornecida uma biblioteca de classes que pode ser utilizada por qualquer linguagem habilitada ao .NET – .NET é uma mudança de Paradigma. Ao colocar a disposição do programador um conjunto de classes padronizadas, o Framework .NET está incentivando boas práticas de programação, como a herança e a consequente reutilização de código.

Em seguida apresentamos os problemas com o modelo de desenvolvimento atual, baseado em componentes que seguem o padrão COM/COM+. Vimos que, para que um componente possa ser utilizado, o mesmo precisa ser registrado e que diferentes versões de um mesmo componente não podem coexistir em um mesmo computador, o que pode fazer com que, ao instalarmos um programa, o mesmo substitua a versão atual do componente por uma versão diferente, a qual não pode ser utilizada por outros programas que dependem da versão anteriormente instalada. Isto faz com que o programa que depende da versão substituída deixe de funcionar. Esta situação também é conhecida como “DLL Hell.”

Também fizemos uma revisão rápida dos modelos de desenvolvimento em 2, 3 ou n camadas. Apresentamos as deficiências e vantagens de cada modelo.

Em seguida voltamos aos componentes do Framework .NET apresentando, em primeiro lugar, as linguagens que fazem parte do Framework .NET, com destaque para a facilidade em fazer com que programas escritos em diferentes linguagens possam se comunicar e trocar informações. Fizemos uma descrição breve do VB.NET e da nova linguagem C#. Neste livro estaremos utilizando o C# para a criação das páginas ASP.NET de exemplo.

Continuando nossa jornada pela Framework .NET apresentamos o CTS – Common Type System. Vimos a importância de todas as linguagens utilizarem um conjunto padronizado de dados, principalmente para garantir a interoperabilidade entre programas criados em diferentes linguagens.

Também tratamos do conceito de metadados (Metadata), o qual faz com que cada componente contenha todas as informações necessárias ao seu funcionamento, o que evita que o mesmo tenha que ser registrado, como acontecia

com os componentes COM/COM+. Além disso os metadados é que possibilitam que diferentes versões de um componente coexistam em um computador, podendo inclusive as diferentes versões serem executadas simultaneamente.

Vimos que a unidade básica de “empacotamento” é chamada de assembly. Um assembly contém todos os elementos necessários ao funcionamento de um componente. Falaremos mais sobre assemblies no próximo capítulo.

IMPORTANTE: Para a documentação sempre atualizada sobre o .NET, não deixe de consultar o endereço: <http://msdn.microsoft.com/net>.

NOTA: Devido à natureza dinâmica da Internet, alguns dos links apresentados podem ter sido alterados e, portanto, deixado de funcionar.

Seguindo, passamos a tratar das interfaces com o usuário, onde vimos os conceitos de Windows Forms e Web Forms, onde foram apresentados diagramas ilustrativos da utilização destes elementos. Voltaremos ao assunto Web Forms nos demais capítulos deste livro.

Também não poderíamos deixar de fora uma breve apresentação dos servidores .NET. Uma linha de servidores da Microsoft, para facilitar o desenvolvimento de aplicações .NET, os quais fornecem uma série de funcionalidades, desde um servidor de Banco de Dados Relacionais (SQL Server 2000), até um servidor para garantir a segurança na Internet – Internet Security and Acceleration Server 2000.

Para encerrar fizemos uma breve apresentação do ADO.NET. Voltaremos a tratar de ADO.NET, principalmente nos capítulos que tratam de conexão de páginas ASP.NET com Bancos de dados e outras fontes de dados.

Introdução

Antes de iniciarmos o nosso aprendizado de C# (Capítulos 3, 4 e 5) e de ASP.NET (demais capítulos deste livro), vamos apresentar mais alguns detalhes sobre o CLR – Common Language Runtime. O CLR, conforme descrevemos no Capítulo 1, é o componente principal do Framework .NET.

Iniciaremos o capítulo apresentando mais alguns detalhes sobre a estrutura de uma aplicação .NET e a maneira como os diversos elementos são “empacotados”, através do uso de assemblies. Vamos apresentar informações detalhadas sobre o uso de assemblies, os benefícios da utilização dos mesmos, questões de segurança, questões sobre versão e utilização de um cache de assemblies para acelerar a execução dos programas .NET.

Falaremos um pouco sobre as linguagens habilitadas ao .NET, a geração do código MSIL, a compilação do código MSIL para código nativo – JIT Compile e a execução deste código. Estes aspectos são importantes para entendermos a maneira como as aplicações .NET são executadas e gerenciadas pelo CLR.

Detalharemos o conceito e a função dos assemblies. Veremos que os mesmos formam os blocos básicos para a construção de aplicações .NET. Também veremos quais os componentes de um assembly, com destaque para a importância do “manifesto” contido em um assembly.

Na seqüência apresentaremos mais detalhes sobre o CTS – Common Type System. Veremos quais os principais benefícios de termos um sistema de tipos comum a todas as linguagens habilitadas ao .NET. Também falaremos sobre as diferenças entre “value types” e “reference types”.

Também apresentaremos as principais classes e tipos da .NET Framework Class Library. Veremos os tipos básicos, bem como as principais classes disponíveis para utilização em nossos programas e páginas ASP.NET.

Na parte final do capítulo faremos uma revisão dos conceitos básicos de orientação a objetos. Uma vez que o Framework .NET é fortemente baseado em conceitos de orientação a objetos, também a linguagem C# que utilizaremos neste livro para a criação de páginas ASP.NET é baseada nos conceitos de orientação a objetos; é bastante oportuno que revisemos os seguintes conceitos:

- ◆ O que é um objeto?
- ◆ Métodos e propriedades
- ◆ O que são Classes?
- ◆ Herança

CAPÍTULO

2

Entendendo o CLR

- ◆ Polimorfismo e funções virtuais
- ◆ Encapsulamento e visibilidade

Todo o Framework .NET foi criado com base nos conceitos de Orientação a Objetos. A linguagem C# é totalmente orientada a objetos, onde tudo são classes. Claro que alguns “puristas” poderiam dizer que o C# não é completamente orientado a objetos porque não suporta herança múltipla. Na verdade veremos que o C# foi projetado para conciliar os benefícios da orientação a objetos e ao mesmo tempo ser simples e prático. Para isso foram eliminadas algumas características da orientação a objetos, características estas que mais causavam problemas do que propriamente forneciam soluções.

CLR – Common Language Runtime

No Capítulo 1 falamos rapidamente sobre o CLR. Agora chegou o momento de apresentarmos mais alguns detalhes sobre este, que, com certeza, é o principal elemento do Framework .NET.

O CLR é o ambiente de execução do Framework .NET. O CLR gerencia a execução de qualquer aplicativo .NET, além de fornecer uma série de serviços que facilitam e agilizam o desenvolvimento de aplicações. Conforme discutido no Capítulo 1, o maior benefício do CLR é que o mesmo fornece uma série de serviços e funcionalidades que nos modelos de desenvolvimento anteriores ao .NET tinham que ser codificados na própria aplicação. Ao fornecer este conjunto de serviços e funcionalidades, estamos evitando que o programador tenha que se preocupar como a maneira com a aplicação vai ser executada, com a alocação de memória, com liberação de memória e tantos outros aspectos que dificultavam o desenvolvimento de aplicações. Com o suporte fornecido pelo CLR, o programador somente precisa se preocupar com a lógica da sua aplicação e não com a infraestrutura necessária para a mesma funcionar.

O Código que é executado sob o controle do CLR é chamado de “Managed Code”; vamos arriscar uma tradução: Código Gerenciado. Este gerenciado significa: “Sob o controle do CLR”. Em contrapartida código que não for executado sob o controle do CLR, no caso de programas não .NET, é chamado de “non Managed Code”, que traduziremos por Código não gerenciado.

Os programas criados com linguagens habilitadas ao .NET e, portanto, executados sob o comando do CLR, podem se beneficiar das seguintes vantagens:

- ◆ Fácil integração e interoperabilidade entre programas criados em diferentes linguagens.
- ◆ Fácil implementação e controle da segurança da aplicação.
- ◆ Utilizar a biblioteca de classes do Framework .NET.
- ◆ Melhor gerenciamento das diferentes versões de um mesmo componente, inclusive com a possibilidade de execução simultânea de diferentes versões do mesmo componente, o que é conhecido por: “Side-by-side execution”.
- ◆ Gerenciamento automático da alocação e liberação da memória.

Compilar ou não Compilar, eis a Questão?

Outro ponto importante a esclarecer é se os programas .NET são compilados ou interpretados. Confesso que, na opinião deste autor, pouca importância tem se é compilado ou interpretado, desde que o desempenho final seja satisfatório. Mas como todo programador que se preza é um pouco teimoso, e apaixonado por uma discussão acalorada, vamos esclarecer este assunto antes que os ânimos se exalte.

Ao criarmos um programa, utilizando uma linguagem habilitada ao .NET, como por exemplo o VB.NET ou o C#, o código-fonte do programa, ao ser compilado, gera código MSIL – Microsoft Intermediate Language. Senhores! Com calma. Ainda não é hora de começar a malhar o .NET. O CLR, que será responsável pela execução e controle do código MSIL gerado, COMPILA o código MSIL para código nativo. Com isso, o código que é executado pelo CLR é código nativo da plataforma onde está instalado o Framework .NET. Até o momento, a única plataforma compatível é a dobradinha Windows/Intel. Porém a Microsoft está tornando disponível, publicamente, as especificações para CLR e MSIL, o que possibilita que outras empresas desenvolvam um Framework .NET para outras plataformas, como, por exemplo, o UNIX.

Nos veremos mais adiante, que mesmo uma página ASP.NET é compilada no servidor Web, antes de ser enviada para o Navegador do cliente. Em resumo, todo e qualquer código executado pelo CLR, é código compilado nativo. Agora sim. Fiquem à vontade para discutir, dizer que não é bem assim, que esse tal de MSIL não tem nada a ver, mais um chope e assim por diante.

O Papel dos Metadados (Metadata)

Para que o CLR possa fornecer uma série de serviços ao “managed code”, os compiladores das linguagens habilitadas ao .NET devem ser capazes de criar metadados. Os metadados contêm informações sobre os tipos, membros e referências contidas no código do programa. Os metadados são gravados com o próprio código do programa. Desta maneira toda a informação necessária para que o programa funcione está contida no seu próprio código, o que faz com que não seja necessário o registro do mesmo, diferente do que acontecia com os componentes COM/COM+. O CLR utiliza metadados para localizar e carregar programas ou componentes, organizar as várias instâncias de um mesmo componente na memória, resolver a chamada de métodos, gerar código nativo, garantir a segurança de acesso e definir os limites para o contexto de execução de um componente ou programa.

O CLR automaticamente gerencia a alocação de objetos na memória, bem como as referências aos objetos. Quando não houver mais nenhuma referência ao objeto, o que significa que o mesmo não está mais sendo utilizado, o CLR automaticamente remove o objeto, liberando mais memória para o sistema. Com isso, os tradicionais problemas de “memory leaks” – programas que alocam recursos de memória e não liberam, fazendo com que a quantidade de memória disponível fique reduzida, até o ponto de fazer com que o próprio sistema operacional fique instável e a máquina tenha que ser reinicializada – deixam de existir. Vejam o quanto fica simplificada a tarefa de desenvolvimento, uma vez que o programador não precisa preocupar-se com alocação e liberação de memória, apenas com a lógica do seu programa.

Integração Entre Diferentes Linguagens: Promessa ou Realidade?

Com o CLR fica fácil a criação de componentes e aplicações nas quais os objetos sejam capazes de interagir, mesmo que codificados em diferentes linguagens. Por exemplo, podemos criar uma Classe chamada Clientes, em VB.NET.

Podemos criar uma classe Clientes Especiais, em C#, herdada da classe Clientes. Com isso, a nossa classe Clientes Especiais irá herdar todas as propriedades e métodos da classe Clientes.

Essa integração entre diferentes linguagens é possível, pois todas as linguagens habilitadas ao .NET têm acesso a um sistema de tipos comuns CTS – Common Type System. Falaremos um pouco mais sobre o CTS mais adiante, neste capítulo. Além disso cada componente ou aplicativo possui informações, na forma de metadados, sobre a sua estrutura, sobre os métodos e propriedades que o mesmo possui e que podem ser acessados por outros componentes.

NOTA: No final do capítulo apresentaremos os principais conceitos de orientação a objetos, como por exemplo: classes e herança.

O Processo de Execução de Código do CLR

Vamos ver quais os passos envolvidos, desde a criação de uma aplicação utilizando uma das linguagens habilitadas ao .NET, até a execução da mesma, sob o controle do CLR.

Os passos para a criação e execução de uma aplicação .NET podem ser resumidos da seguinte maneira:

- ◆ Cria a aplicação ou componente utilizando uma linguagem habilitada ao .NET (VB.NET, C#, etc).
- ◆ Compila o código da sua aplicação ou componente para gerar código MSIL. O compilador da linguagem que você está utilizando compila o seu código-fonte para MSIL e acrescenta os meta dados necessários.
- ◆ Em tempo de execução, um compilador JIT (Just In Time) do CLR transforma o código MSIL em código nativo, COMPILADO.
- ◆ O código nativo é executado, utilizando toda a infraestrutura disponibilizada pelo CLR.

Na Figura 2.1, temos uma ilustração destes passos.

Conforme ilustrado na Figura 2.1, quando compilamos o código-fonte criado por uma linguagem habilitada ao .NET, estamos gerando MSIL, a qual é um conjunto de instruções, independente da CPU, conjunto este que pode ser convertido (pelo JIT) para código nativo. O código MSIL inclui instruções para carregar, armazenar, inicializar e chamar métodos, instruções para operações aritméticas e lógicas, controle de fluxo do programa, acesso direto a memória, tratamento de exceções e demais operações necessárias à execução do programa.

Antes que o código MSIL possa ser executado, o mesmo precisa ser convertido para código específico da CPU em questão. Isto é feito pelo JIT – Just in time compiler. O CLR disponibiliza compiladores JIT para cada arquitetura suportada pelo CLR; desta maneira o mesmo conjunto de instruções MSIL pode ser compilado e executado em qualquer arquitetura para a qual exista um compilador JIT.

Esta é a estratégia da Microsoft para fazer com que o código MSIL de aplicações .NET possa ser executado em diferentes plataformas, desde servidores Windows ou não Windows, até dispositivos móveis para os quais esteja disponível um compilador JIT. É claro que a implementação em outras plataformas que não o Windows, como por exemplo o UNIX, dependerá da implementação de terceiros. Por exemplo, a HP pode querer implementar um compilador

JIT que rode no HP-UX, fazendo com que aplicações .NET possam rodar nesta plataforma. Se surgirão implementações para outras plataformas é uma questão que somente o tempo e o mercado dirão.

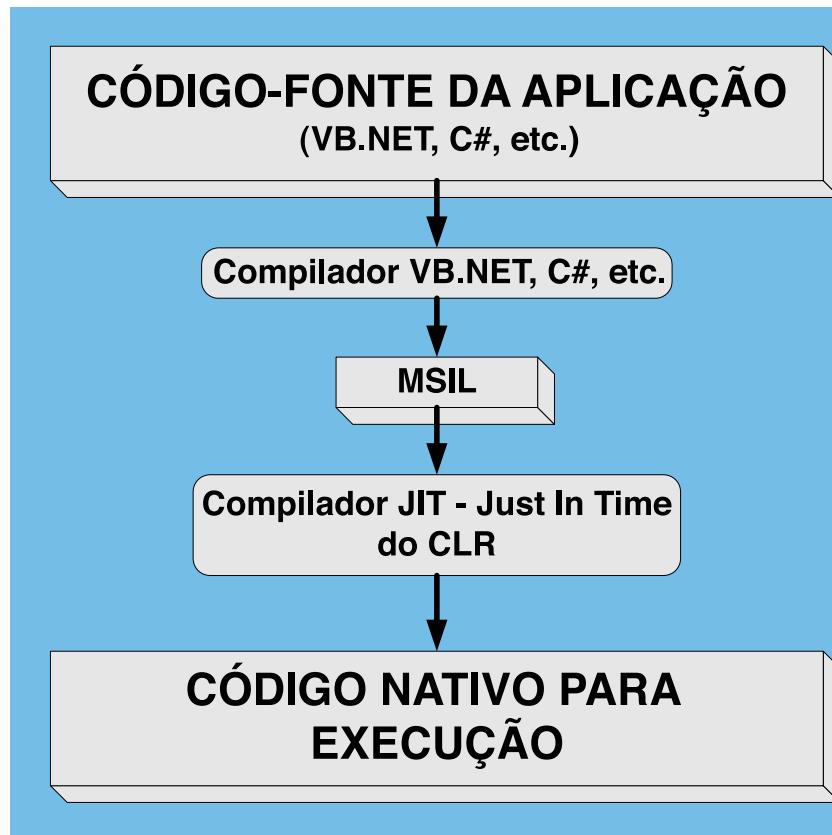


Figura 2.1: O processo de execução de código do CLR.

Quando o compilador de uma linguagem habilitada ao .NET produz o código MSIL, o mesmo também cria os metadados necessários. O código MSIL e os metadados estão contidos em um arquivo PE – portable executable. O portable significa que este arquivo pode, em tese, ser compilado e executado em qualquer plataforma para a qual existe um compilador JIT. A presença dos metadados, juntamente com o código MSIL, torna o código autodescritivo, o que significa que não temos mais a necessidade de “bibliotecas de tipo” (type libraries) ou IDL – Interface Definition Language, dispositivos que eram necessários ao funcionamento de componentes baseados no modelo COM/COM+. Voltamos a salientar este ponto, o componente criado para o .NET possui, na forma de metadados, toda a informação necessária ao seu funcionamento e necessária para que possa ser acessado por outros componentes. O CLR localiza e extrai os metadados do arquivo PE durante a execução, à medida que isso vai sendo necessário.

Mais Algumas Observações Sobre o JIT

O processo de compilação utilizado pelo JIT procura fazer uma série de otimizações, para tornar o código nativo gerado o mais enxuto e veloz possível.

O JIT detecta se algum método ou propriedade não será utilizado durante a execução do programa. Ao invés de converter todo o código MSIL, contido no arquivo PE, o JIT converte apenas o código que realmente será necessário

para a execução do programa. Isso faz com que a compilação seja mais rápida, com que sejam poupados recursos como por exemplo memória, e o código resultante seja otimizado, tanto em termos de tamanho quanto de velocidade de execução.

Como parte da compilação do MSIL para código nativo, o código deve passar por um processo de verificação, a não ser que tenha sido definida uma política de segurança que permite ao código passar pela verificação. O processo de verificação faz uma análise do código MSIL e dos metadados, para definir se o código pode ser definido como “type safe”. Para que o código possa ser definido como type safe, o mesmo somente deve acessar regiões de memória para as quais o mesmo tem autorização. Este processo é necessário para garantir que os objetos estejam isolados uns dos outros, de tal maneira que um objeto não tente alterar ou corromper dados de outros objetos. Na plataforma Windows 9x, quando um programa tenta acessar a área de memória utilizada por outro programa, é gerada uma “Falha Geral de Proteção”, com a famigerada mensagem “Você Executou uma Operação Ilegal e o seu programa será fechado”.

Para que o código passe no teste de “type safe”, o mesmo deve atender as seguintes condições:

- ◆ Todas as referências feitas a um determinado tipo são compatíveis com o tipo que está sendo referenciado.
- ◆ Somente operações devidamente definidas são invocadas pelo objeto.
- ◆ É possível verificar e confirmar a identidade do objeto. Isso evita que um componente, por exemplo um vírus, se faça passar por um componente válido, porém com um comportamento bem diferente do verdadeiro componente.

O processo de verificação também faz uma análise do código MSIL, para ver se o mesmo foi corretamente gerado. Se as políticas de segurança da nossa aplicação definem que o código deve passar na verificação de “type safe” e o código não passar nesta verificação, uma exceção será gerada quando o código for executado.

Um detalhe importante é que nem todo o código MSIL é compilado para código nativo, de uma só vez. A compilação acontece em nível de método. Quando um método é chamado, o código MSIL do mesmo é compilado, o código nativo é gerado pelo JIT e o método é executado. O código nativo gerado é mantido. Quando o método for chamado pela segunda vez, o código nativo gerado na primeira chamada será executado, evitando, com isso, que o código do método tenha que ser compilado a cada chamada do mesmo, o que acarretaria uma perda considerável de desempenho.

NOTA: O mecanismo de verificação do código é um exemplo típico de funcionalidade disponibilizada pelo Framework .NET, a qual, nos modelos anteriores, precisava ser desenvolvida pelo programador.

Assemblies

Conforme descrito no Capítulo 1, chegou o momento de apresentarmos mais alguns detalhes sobre Assemblies e sobre o formato (ou quem sabe anatomia) de uma aplicação .NET.

Uma Definição em Poucas Palavras

Para aplicações tradicionais do Windows, anteriores ao modelo .NET, a aplicação final é um arquivo executável ou um arquivo executável mais um conjunto de .DLL e componentes. Uma DLL pode conter um objeto COM/COM+ em uma biblioteca de tipos (typelib).

Para o Framework .NET, os diversos componentes de uma aplicação são “empacotados” através da utilização de assemblies. Um assembly contém todo o código MSIL gerado pelo compilador, os metadados e os demais arquivos necessários ao funcionamento da aplicação. Cada assembly também contém um “manifesto”. O manifesto contém as seguintes informações:

- ◆ Uma lista dos arquivos contidos no assembly.
- ◆ A definição de quais tipos e recursos do assembly podem ser acessados por outros componentes ou programas.
- ◆ Um mapeamento entre os tipos e recursos disponibilizados pelo assembly e os arquivos que contêm os tipos e recursos.
- ◆ Uma lista de outros assemblies, dos quais o assembly depende para o seu correto funcionamento.
- ◆ Informações sobre a identidade do assembly, incluindo o nome e a versão do assembly.
- ◆ Se o assembly for público, como por exemplo um Web Service, o manifesto também pode conter a chave pública do assembly. Este é um mecanismo semelhante, em funcionalidade, à utilização de um Certificado Digital, para identificar a origem de um controle ActiveX.

Um assembly contém toda a informação necessária ao seu funcionamento. Ao definirmos um assembly, o mesmo pode estar contido em um único arquivo ou em múltiplos arquivos. A vantagem de utilizar múltiplos arquivos é que as diferentes partes do assembly podem ser carregadas e executadas à medida que as mesmas forem sendo utilizadas.

Funções do Assembly

Pela descrição anterior podemos concluir que o assembly é um elemento fundamental na criação de aplicações .NET. Os assemblies foram projetados, principalmente, para simplificar o processo de distribuição de aplicações e para solucionar o problema do controle de versões existente no modelo COM/COM+. Um assembly é responsável pelas seguintes funções:

- ◆ Contém o código executado pelo CLR. O código MSIL contido em um arquivo PE (portable executable) não poderá ser executado se o mesmo não estiver associado a um manifesto.
- ◆ É a unidade básica para atribuição de permissões e configurações de segurança. Podemos definir, em nível de assembly, quem pode acessar o assembly e com que nível de acesso.
- ◆ É uma unidade de referência, pois as informações contidas no manifesto do assembly são utilizadas para resolver tipos e para atender requisições de outros assemblies. Estas informações definem quais os métodos e propriedades que são visíveis externamente. O manifesto também contém informações a respeito de quais assemblies o assembly em questão depende.
- ◆ É uma unidade para definição de versão. A atribuição de um número de versão é feita em nível de assembly.

- ◆ É uma unidade de distribuição das aplicações .NET. Quando uma aplicação é inicializada, somente o assembly chamado na inicialização da aplicação precisa estar disponível. Outros assemblies somente são chamados à medida que forem sendo utilizados. Essa é uma característica muito importante, principalmente para aplicações Web, onde a velocidade das conexões é sempre um fator muito importante a ser considerado.

Assemblies podem ser estáticos ou dinâmicos. Assemblies estáticos podem incluir tipos do Framework .NET (interfaces e classes), além dos recursos utilizados pelo assembly (bitmaps, .jpg files, arquivos de som ou imagem, etc.). Assemblies estáticos são gravados em disco em arquivos PE. Na verdade um arquivo PE pode ser gravado na forma de um EXE ou uma DLL.

Assemblies dinâmicos rodam diretamente da memória e não são salvos no disco antes de serem executados. Após a execução, podemos salvar assemblies dinâmicos em disco.

Componentes do Assembly

Um assembly estático é formado pelos seguintes componentes:

- ◆ Manifesto, o qual contém metadados do assembly.
- ◆ Metadados sobre os tipos do assembly (Type metadata).
- ◆ Código MSIL.
- ◆ Demais recursos necessários, como por exemplo arquivos gráficos, arquivos de vídeo, etc.

Podemos agrupar todos os elementos constituintes do assembly em um único arquivo, conforme ilustrado na Figura 2.2.



Figura 2.2: Assembly e seus componentes em um único arquivo.

Os componentes do assembly também podem estar divididos em diversos arquivos. Estes arquivos podem conter módulos de código compilado, recursos (como por exemplo arquivos gráficos ou de vídeo) ou qualquer outro arquivo necessário ao funcionamento da aplicação .NET.

Existem duas razões para a colocação dos componentes de um assembly em múltiplos arquivos:

- ◆ Quando precisamos combinar, na mesma aplicação, componentes ou módulos criados em diferentes linguagens.
- ◆ Para otimizar o download dos componentes de uma aplicação, de tal maneira que os componentes ou módulos menos utilizados somente sejam carregados quando forem necessários. Esta é uma alternativa importante, principalmente para aplicações Web.

Na Figura 2.3 temos o exemplo de um assembly composto de quatro arquivos:

- ◆ Um arquivo é o módulo principal. – criado em VB.NET.
- ◆ Outro arquivo é o módulo com funções (Criado em C#) usadas pelo assembly principal.
- ◆ Um arquivo .bmp.
- ◆ Um arquivo .jpg.

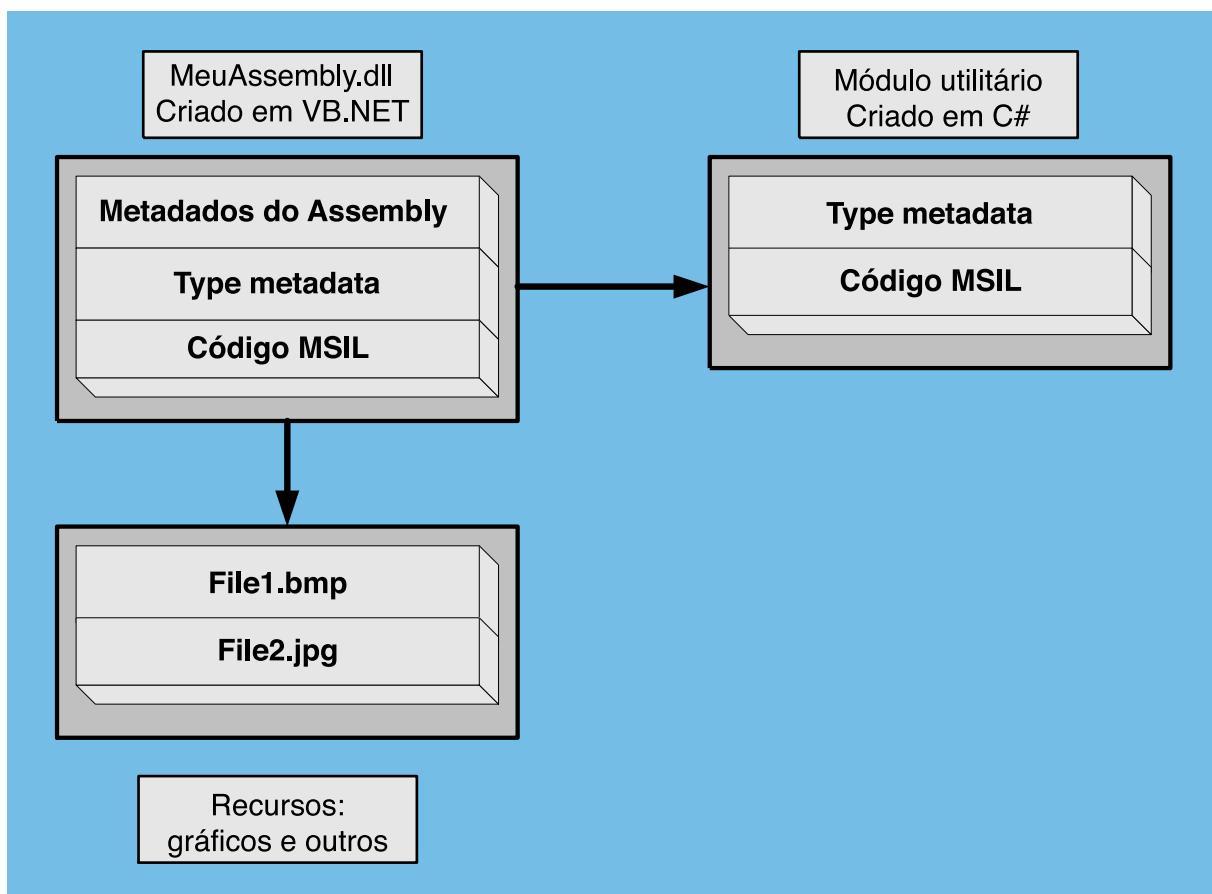


Figura 2.3: Exemplo de um assembly com vários arquivos.

O que Temos no “Manifesto”?

O Manifesto contém o seguinte conjunto de informações (metadados):

- ◆ Qual a relação entre os diferentes componentes do assembly.
- ◆ Informações sobre a versão do assembly.
- ◆ Escopo.
- ◆ Informações para resolver referências a outras classes e componentes.

O manifesto pode estar contido em um arquivo PE – Portable executable (pode ser um .exe ou uma .dll), juntamente com o código MSIL ou pode estar em um arquivo PE separado, o qual contém somente o manifesto. Esta última situação acontece quando temos um assembly formado de diversos arquivos.

As informações contidas no manifesto são responsáveis pelas seguintes funções:

- ◆ Relação dos arquivos que compõem o assembly.
- ◆ Gerencia o mapeamento entre os tipos e recursos disponibilizados pelo assembly e os arquivos onde estão contidas as declarações e implementações dos respectivos tipos e recursos.
- ◆ Relaciona os assemblies dos quais o assembly em questão é dependente.

Em resumo, reforçando o que já foi comentado no Capítulo 1, este conjunto de informações fornece uma autonomia ao assembly, de tal maneira que o mesmo contém toda a informação necessária ao seu funcionamento. Isso evita que o assembly tenha que ser registrado. Com estes recursos, o processo de instalar uma aplicação .NET está resumido a copiar os arquivos necessários. Como diriam os ainda apaixonados pelo DOS, voltamos ao estilo “xcopy” de instalação, em referência ao comando xcopy, o qual é utilizado para copiar arquivos ou pastas.

CTS – Common Type System

O CTS desempenha um papel fundamental para garantir e facilitar a interoperabilidade entre programas e componentes criados em diferentes linguagens. O CTS define a maneira como os tipos de dados são declarados, usados e gerenciados pelo CLR. Através do CTS temos a garantia de que, por exemplo, um inteiro terá as mesmas características, independente da linguagem na qual o mesmo estiver sendo utilizado.

Uma das maiores dificuldades, no modelo COM/COM+, em fazer com que componentes escritos em diferentes linguagens pudessem interagir é o fato de que cada linguagem possui o seu próprio conjunto de tipos, o qual não é compatível com o conjunto de tipos das demais linguagens. Colocando de uma maneira mais simples: o tamanho, forma de armazenamento e demais características de cada tipo é dependente da linguagem, no modelo COM/COM+.

O CTS é responsável pelas seguintes funções:

- ◆ Fornecer uma estrutura de dados e tipos padronizados, estrutura esta que torna a integração entre diferentes linguagens uma realidade, além de garantir uma execução mais rápida, uma vez que as conversões de tipo passam a ser coisa do passado.
- ◆ Fornece um modelo orientado a objetos, o qual é capaz de suportar qualquer linguagem habilitada ao .NET.

- ◆ Define algumas regras e padrões que as linguagens habilitadas ao .NET devem obedecer, o que ajuda a garantir que objetos escritos em diferentes linguagens serão capazes de interagir e trocar mensagens.

Classificação dos Tipos do CTS

Temos duas categorias principais de tipos no CTS:

- ◆ Value types: Toda linguagem de programação fornece um conjunto de tipos básicos de dados, como por exemplo: inteiros, caracteres, números reais, strings, etc. Estes tipos básicos são passados por valores. Em outras palavras, se tenho uma variável x cujo valor é 5, utilizo o seguinte comando:

y=x

neste caso estou passando o valor contido na variável x, para a variável y. Alterações feitas na variável y não afetarão o valor da variável x. No Framework .NET estes tipos básicos, passados por valor, são chamados de value types.

Estes tipos básicos fazem parte do conjunto de classes disponibilizado pelo Framework .NET. Por exemplo, um inteiro é acessado como System.Int32 e um valor booleano como System.Boolean. Estes tipos disponibilizados na própria biblioteca de classes do Framework .NET são chamados de “built-in value types”.

Para fornecer uma maior flexibilidade, o Framework .NET também permite que sejam criados os chamados “User-defined value types.” Com isso o programador pode definir tipos personalizados, os quais são derivados da classe básica System.ValueType. Com isso podemos criar tipos específicos, de acordo com as necessidades de cada aplicação. Por exemplo, para aplicações de engenharia, podemos criar um tipo para representar números complexos ou matrizes com características definidas.

- ◆ Reference types: Este tipo contém uma referência para a localização do valor do tipo. Em outras palavras, contém o endereço de memória onde está armazenado o valor. Vamos imaginar que x seja uma variável do tipo reference type. Agora vamos fazer uma outra variável y, também do tipo reference type, como sendo igual a x:

y=x

Neste caso, o endereço onde está armazenado o valor de x foi atribuído à variável x. Se alterarmos o valor de x, o que acontece com y? Como y contém o endereço da variável x, ao alterarmos x, y passará a enxergar o valor modificado. Por exemplo, se utilizarmos o comando Console.WriteLine(y), será exibido o novo valor da variável x.

NOTA: Nos Capítulos 3, 4 e 5 apresentaremos alguns exemplos que ilustram a diferença entre value types e reference types.

.NET Framework Class Library – Biblioteca de Classes do Framework .NET

O Framework .NET fornece uma biblioteca hierárquica de classes. Esta biblioteca pode ser acessada por qualquer linguagem habilitada ao .NET. Nesta biblioteca temos milhares de classes, interfaces e estruturas, que disponibilizam os mais variados tipos de serviços e funcionalidades, como por exemplo:

- ◆ Acesso a fontes variadas de dados.

- ◆ Configurações de segurança.
- ◆ Desenvolvimento de componentes e Web services.
- ◆ Manipulação de objetos gráficos.
- ◆ Leitura e escrita em disco.
- ◆ Serviço de Sockets, com a possibilidade do envio e recebimento de dados utilizando uma grande variedade de protocolos de rede.
- ◆ Criação de aplicações Web, com serviços de fila de mensagens, correio eletrônico, etc.
- ◆ Manipulação de dados e esquemas XML.
- ◆ Criar aplicações Win32 tradicionais através do uso de Win Forms.
- ◆ Criar aplicações ASP.NET utilizando Web Forms.
- ◆ Acessar as informações contidas nos assemblies, na forma de metadados.

As funcionalidades oferecidas pela biblioteca de classes do Framework .NET facilitam a criação de programas, uma vez que muitas funções já estão prontas para serem utilizadas, evitando que as mesmas tenham que ser implementadas em cada programa. A utilização de classes comuns a todas as linguagens do Framework .NET também facilita a interoperabilidade entre diferentes linguagens.

Podemos utilizar as classes diretamente em nossos programas ou podemos criar classes que herdam os métodos e propriedades de uma determinada classe e adicionar as modificações necessárias. Vejam que aqui estamos utilizando o conceito de herança.

Como são milhares de classes, cada uma com seus métodos e propriedades, o Framework .NET precisa organizar estas classes de uma maneira a evitar conflito de nomes. A maneira encontrada pelo Framework .NET é através da utilização de um espaço de nomes (namespace). Um espaço de nomes é simplesmente um agrupamento lógico das classes, estruturas e interfaces relacionadas. A maioria das classes está contida no espaço de nomes System. Diretamente ligado a system temos a definição de tipos básicos como por exemplo: Int32, String, etc. Um exemplo de classe de segundo nível, dentro do espaço de nomes System, é a classe System.Data, a qual fornece uma série de métodos para acesso às mais variadas fontes de dados. Temos também uma classe de segundo nível chamada System.Security, a qual fornece serviços para configurações da segurança de acesso às aplicações .NET. E assim vamos formando uma imensa hierarquia de classes. As classes de segundo nível, como System.Data e System.Security, herdam todos os métodos e propriedades da classe mãe System. Este é um dos princípios da orientação a objetos: Herança. No próximo item falaremos mais sobre a herança e os demais fundamentos da orientação a objetos.

Na Figura 2.4, temos uma representação parcial do espaço de nomes System, no qual representamos apenas alguns tipos básicos, ligados diretamente a System e às classes derivadas System.Data e System.Security.

Podemos continuar nos aprofundando na hierarquia de classes. Por exemplo, existem classes derivadas de System.Data. A classe System.Data.SqlClient contém toda a funcionalidade necessária para acessar dados de um servidor SQL Server 2000. Já a classe System.Data.OleDb fornece funcionalidades para conexão com qualquer fonte de dados, para a qual esteja disponível um OLE-DB Provider. Se acrescentássemos mais estas duas classes, o espaço de nomes System ficaria conforme indicado na Figura 2.5.

Vejam que a hierarquia de classes vai crescendo. No decorrer deste livro estaremos utilizando diversas classes do .NET Framework Class Library.

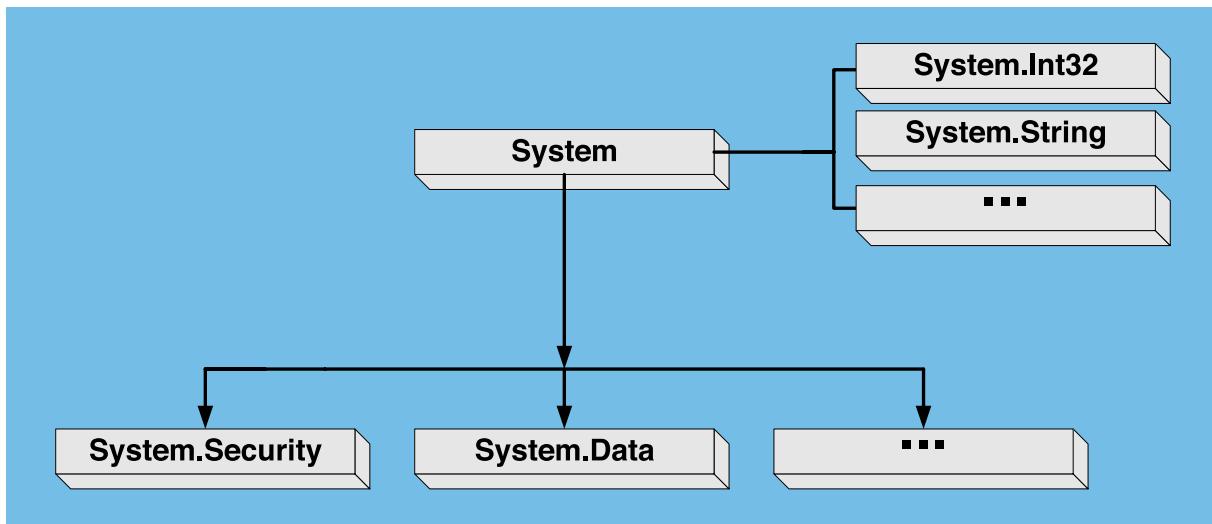


Figura 2.4: Uma representação parcial do espaço de nomes `System`.

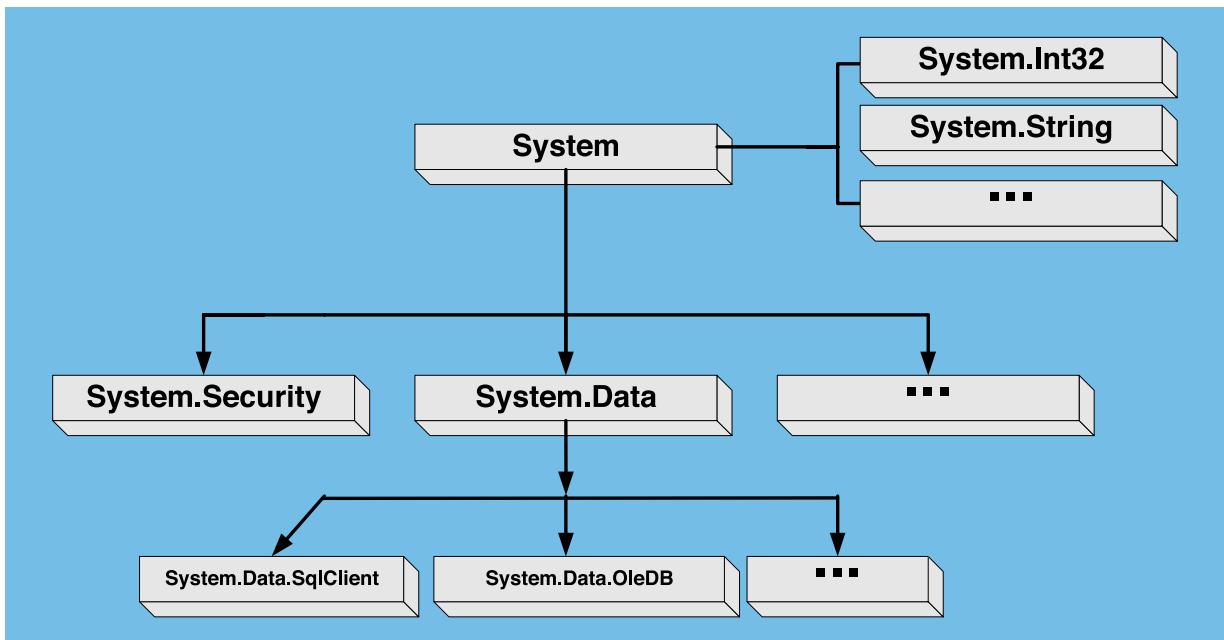


Figura 2.5: `System.SqlClient` e `System.OleDb`.

Vamos fazer uma breve descrição das principais classes do .NET Framework Class Library.

- ◆ **Microsoft.Csharp:** Suporte a linguagem C# do Framework .NET.
- ◆ **Microsoft.Jscript:** Suporte a linguagem JScript no Framework .NET.

NOTA: Para uma descrição completa, de todas as classes do Framework .NET, consulte o item ".NET Framework Class Library", dentro do tópico ".NET Framework Reference", na documentação do Framework .NET, conforme indicado na Figura 2.6. Lembrando que, uma vez instalado o Framework .NET, você pode acessar a documentação do produto utilizando o comando: Iniciar -> Programas -> Microsoft .NET Framework SDK -> Documentation.

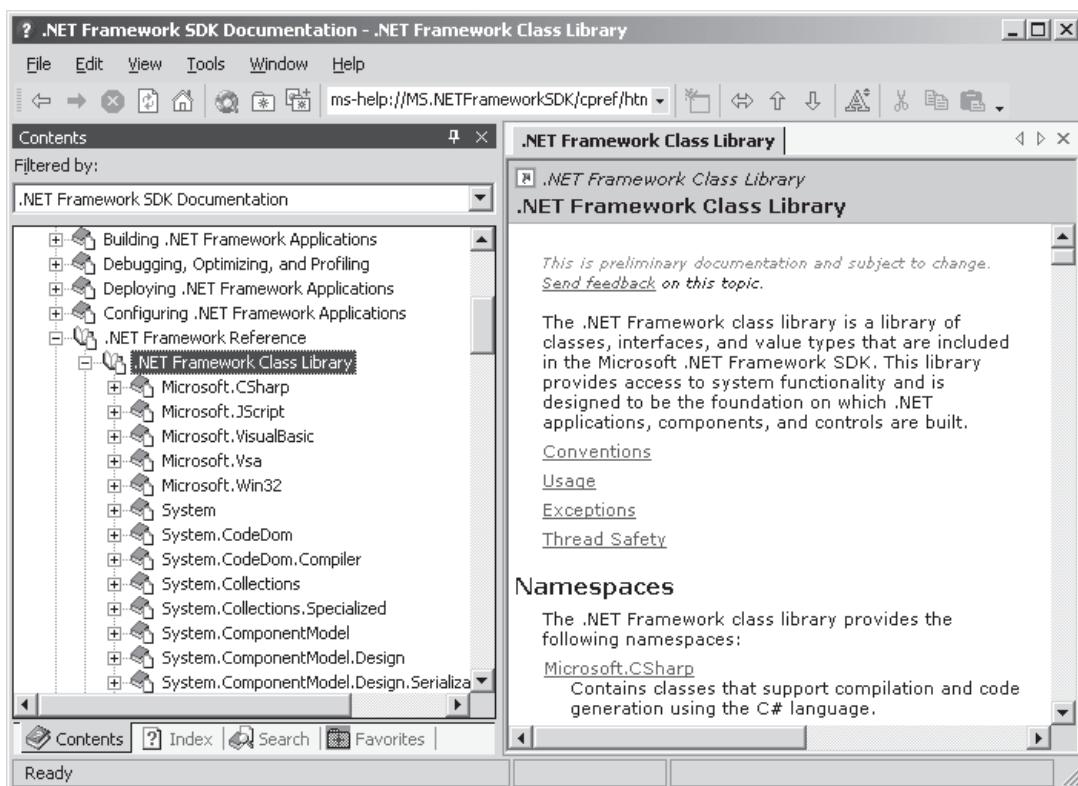


Figura 2.6: Documentação do Framework .NET.

- ◆ **Microsoft.VisualBasic:** Contém o Run Time para o VB.NET e fornece o suporte a linguagem no Framework .NET.
- ◆ **Microsoft.Win32:** Fornece dois tipos de classes: um que trata dos eventos gerados pelo sistema operacional e outro que fornece funções para acessar e gravar informações na registry do Sistema Operacional.
- ◆ **System:** É a classe principal para todo o espaço de nomes System. Contém todas as características comuns a todas as classes, uma vez que as classes derivadas de System herdam suas características. Também contém os tipos básicos, como por exemplo inteiros de 16, 32 ou 64 bits, String, byte, etc.
- ◆ **System.Collections:** Contém as interfaces e classes que definem várias coleções de objetos, como por exemplo listas, filas, arrays e dicionários.
- ◆ **System.Data:** Contém a maioria das classes que compõem a arquitetura do ADO.NET. Com ADO.NET, podemos construir componentes capazes de gerenciar, de uma maneira eficiente, dados de múltiplas fontes. Em um ambiente desconectado (connection less) como a Internet, ADO.NET disponibiliza uma série de ferramentas para requisitar, alterar e sincronizar dados em uma arquitetura de múltiplas camadas. O principal elemento do ADO.NET é uma classe chamada DataSet, a qual estudaremos em detalhes nos capítulos sobre ASP.NET.
- ◆ **System.Data.Common:** Contém as classes que são compartilhadas por todos os .NET data providers (provedores de dados para o .NET). Um .NET data provider é uma coleção de classes que fornece acesso a uma determinada fonte de dados, como por exemplo dados em um Mainframe ou em um servidor SQL Server 2000.
- ◆ **System.Data.OleDb:** Contém as classes que fornecem acesso a qualquer fonte de dados para a qual exista um OLE-DB Provider disponível.
- ◆ **System.Data.SqlClient:** Contém um conjunto de classes que fornece um acesso otimizado ao SQL Server 2000. Utiliza o driver nativo para o SQL Server 2000, ao invés de um OLE-DB Provider ou uma fonte ODBC. Por isso o acesso é mais rápido.

- ◆ **System.Diagnostics:** Um conjunto de classes com que permite que seja feita a depuração e acompanhamento da execução do código das aplicações .NET. Também fornece classes para leitura e escrita no log de eventos, para inicializar serviços do Sistema Operacional, para monitorar a performance do sistema através da utilização de contadores de desempenho.
- ◆ **System.DirectoryServices:** Um conjunto de classes que fornece, para as aplicações .NET, acesso ao Active Directory do Windows 2000. As classes de System.DirectoryServices podem ser utilizadas com qualquer Active Directory Service Provider disponível. Atualmente estão disponíveis os seguintes:
 1. IIS – Internet Information Server
 2. LDAP – Lightweight Directory Access Protocol
 3. NDS – Novel Directory Services
 4. WinNT – Para acesso ao diretório do Windows NT 4.0.
- ◆ **System.Drawing:** Um conjunto de classes com funções gráficas da biblioteca gráfica do Framework .NET, a qual é conhecida como GDI+ (bem que poderia ser GDI.NET). Fornece métodos para desenhar elementos básicos no vídeo, como por exemplo retângulos, círculos, uma linha reta, um ponto, etc.
- ◆ **System.Globalization:** Este namespace contém o conjunto de classes com as funcionalidades necessárias para a construção de aplicações com suporte a diferentes idiomas.
- ◆ **System.IO:** Um conjunto de classes com suporte a leitura e escrita, síncrona e assíncrona para stream de dados e arquivos em disco.
- ◆ **System.Messaging:** Contém um conjunto de classes para habilitar os programas .NET a trabalhar com filas, enviar mensagens para uma fila, ler mensagens de uma fila, etc. No Windows NT 4.0 tínhamos o MSMQ – Microsoft Message Queue que atuava como servidor, disponibilizando serviços de filas para aplicações COM.
- ◆ **System.Net:** Disponibiliza uma interface de programação bastante fácil de utilizar, a qual nos dá acesso à grande maioria dos protocolos disponíveis na Internet.
- ◆ **System.Security:** Um conjunto de classes com métodos para a definição das configurações básicas de segurança para aplicações .NET.
- ◆ **System.Security.Cryptography:** Disponibiliza serviços de criptografia, incluindo codificação e decodificação de dados, hashing, geração de números aleatórios e autenticação de mensagens.
- ◆ **System.Web:** Contém uma série de classes e interfaces para a comunicação browser/servidor web. Por exemplo, existe uma classe chamada HTTPRequest a qual disponibiliza uma série de informações a respeito da requisição HTTP feita pelo cliente. Existem outros namespaces, herdados de System.Web, como por exemplo: System.Web.Caching, System.Web.Configuration, System.Web.Hosting, System.Web.Mail, System.Web.SessionState, etc.
- ◆ **System.Windows.Forms:** Contém classes para a criação de aplicações Win32, as quais podem ter acesso a todos os elementos da interface do Windows, como por exemplo botões, menus, barras de rolagem, etc. Neste namespace encontraremos uma classe chamada Form além de muitos outros controles que podem ser utilizados para a criação da interface do usuário.

NOTA: Para que o programador possa utilizar de maneira eficiente estas classes, um bom conhecimento do diretório em questão é exigido.

- ◆ **System.XML:** É o namespace que contém as classes que dão suporte ao padrão XML.

Apresentamos apenas uma descrição básica dos principais namespaces do Framework .NET. Livros inteiros podem ser escritos sobre um único namespace como por exemplo System.Net ou System.Security. A melhor fonte de consulta para as classes, métodos e propriedades de um namespace é a própria documentação do Framework .NET. No decorrer deste livro estaremos utilizando algumas classes em nossas páginas ASP.NET. Uma das classes que mais utilizaremos é System.Data e suas classes derivadas como System.SqlClient e System.OleDb.

Agora vamos fazer uma revisão dos principais conceitos de Orientação a Objetos. É importante a revisão destes conceitos uma vez que a linguagem C# (assunto para os capítulos 3, 4 e 5) e todo o Framework .NET são baseados nos conceitos de Orientação a Objetos.

Conceitos Básicos de Orientação a Objetos

Vamos fazer uma revisão dos principais conceitos de orientação a objetos. A orientação a objetos, quer seja como metodologia de análise, projeto ou programação, foi criada com o objetivo de resolver problemas que a programação estruturada não foi capaz. A orientação a objetos possui os seguintes objetivos básicos:

- ◆ Produtividade.
- ◆ Incentivo a boas práticas de programação.
- ◆ Uma modelagem mais próxima do entendimento do usuário final.
- ◆ Reutilização de código.
- ◆ Facilidade de manutenção do código.

Embora a proposta de orientação a objetos seja bastante consistente, a mesma não teve o nível de implementação esperado. Não cabe aqui discutir os motivos que fizeram com que a velocidade de adoção ficasse abaixo da esperada. Este fato é ainda mais interessante uma vez que a grande maioria das empresas admite que a orientação a objetos oferece uma série de vantagens em relação ao modelo de programação estruturada. Porém a mudança para um novo modelo envolve uma nova maneira de pensar, uma necessidade de treinamento e, principalmente, uma mudança cultural. Esta última sem sombra de dúvida a mudança mais difícil.

Vamos revisar os seguintes conceitos básicos:

- ◆ O que é um objeto?
- ◆ Mensagens.
- ◆ Classes.
- ◆ Herança.
- ◆ Instâncias.
- ◆ Reutilização de código.

O que é um Objeto?

Na programação estruturada, nós temos uma separação entre procedimentos e os dados sobre os quais os mesmos atuam. Um procedimento pode ser uma função que retorna um valor ou uma procedure que realiza uma série de operações, podendo ou não retornar um valor. Porém este modelo apresenta sérias dificuldades do ponto de vista do usuário. Ao fazer a análise e projeto de um programa, o analista precisa traduzir as necessidades dos usuários em termos de procedimentos, funções, módulos, etc. Existe uma diferença muito grande entre a linguagem dos usuários e a linguagem dos analistas/programadores.

Um objeto é uma entidade que contém, além dos dados, todas as funções que atuam sobre estes dados. Ou seja, um objeto é composto dos dados que descrevem o objeto (propriedades) e das operações que podem ser realizadas sobre estes dados (métodos). Esta abordagem já é um pouco mais próxima da abordagem entendida pelos usuários finais do sistema. Vamos imaginar um sistema para o departamento de recursos humanos. Os usuários falam em termos de empregados, cargos, etc. Poderíamos ter um objeto chamado Funcionário. Este objeto poderia conter diversas propriedades, tais como:

- ◆ Matrícula.
- ◆ Nome.
- ◆ Endereço.
- ◆ Fone.
- ◆ Data de Admissão.
- ◆ Data de Aniversário.
- ◆ Nome do Pai.
- ◆ Nome da Mãe.
- ◆ Número da Identidade.
- ◆ Número do CPF.
- ◆ Cargo.
- ◆ Salário.

Com isso podemos observar que as propriedades descrevem as características de um determinado objeto. O conjunto de valores contidos nas propriedades de um determinado objeto define o seu estado atual.

Além das propriedades o objeto pode conter métodos. Os métodos descrevem ações que podem ser realizadas pelo objeto ou no objeto. Por exemplo, o nosso objeto funcionário poderia ter um método chamado pagamento, outro chamado transferência, mais um chamado promoção e assim por diante. Um método, na prática, é uma função ou procedimento que realiza uma série de ações. Os métodos de um objeto podem receber parâmetros e ter o seu comportamento alterado, dependendo do valor dos parâmetros. Por exemplo, o método Promoção de um objeto funcionário pode receber, como parâmetros, a Matrícula do funcionário, a data da promoção e o código do novo cargo. Dentro do método Promoção pode ser chamado um método AtualizaSalário, o qual atualiza o valor do salário do funcionário, de acordo com o novo cargo que o mesmo irá ocupar.

Na Figura 2.7 temos uma representação do objeto Funcionário.

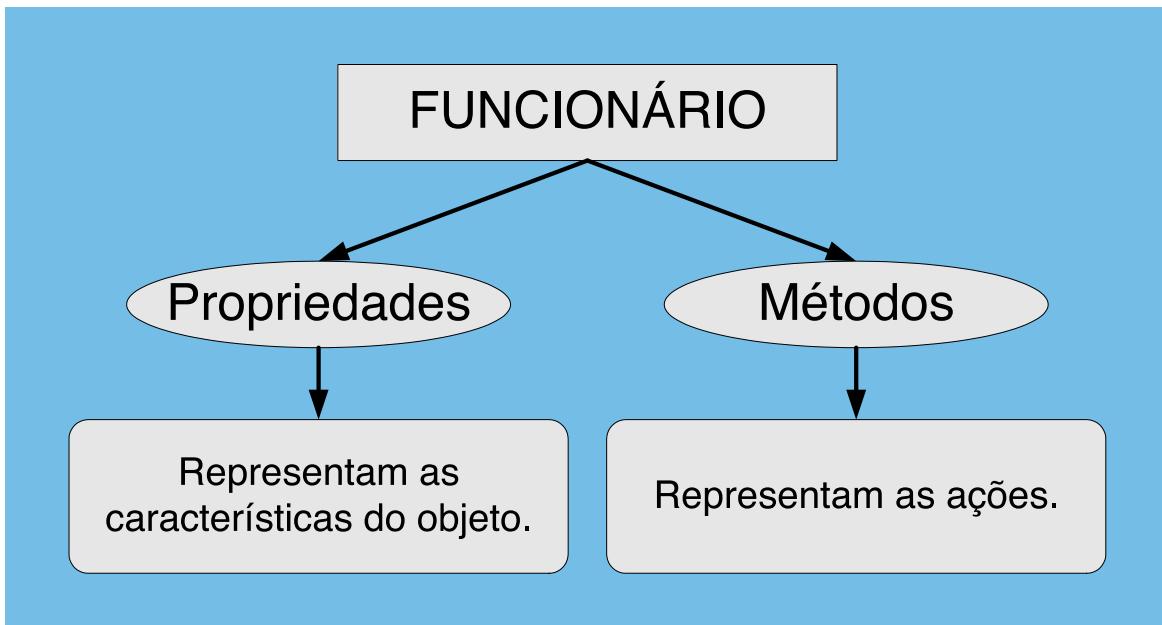


Figura 2.7: Objeto Funcionário – métodos e propriedades.

Mensagens

Os objetos se comunicam através de um mecanismo de troca de mensagens. Mensagens são mecanismos de comunicação entre objetos através do qual se desencadeia a execução de um método específico. Através dos mecanismos de troca de mensagens um objeto pode invocar métodos de outros objetos e receber os resultados da execução do método invocado.

No desenvolvimento baseado em padrões como COM/COM+ existem métodos proprietários para troca de mensagens. Além disso, cada linguagem de programação possui um conjunto distinto de tipos. Desta maneira a troca de mensagens entre objetos criados em diferentes linguagens não é uma tarefa simples. Já no .NET existe um sistema de tipos comuns a todas as linguagens: CTS – Common Type System. A existência do CTS facilita, enormemente, a interação entre objetos criados em diferentes linguagens. Outra vantagem do Framework .NET é que o mesmo utiliza um sistema para troca de mensagens, baseado no padrão XML e no protocolo SOAP, ambos adotados como um padrão de fato pela indústria de TI.

Outro termo relacionado com orientações a objetos é instância. Quando um programa carrega um determinado objeto na memória, dizemos que foi feita uma instância do objeto. Ao carregar na memória estamos reservando recursos para armazenar as características do objeto – através da definição dos valores de suas propriedades e também recursos para que os métodos do objeto possam ser executados.

Classes

Se pesquisarmos a bibliografia sobre orientação a objetos encontraremos um sem-fim de definições para classes. Vamos inicialmente apresentar algumas das definições formais encontradas na bibliografia. Depois vamos a uma, digamos, explicação mais light:

- ◆ Classes constituem modelos que são utilizados para a criação de objetos. Nas classes são descritas a estrutura de dados (através das propriedades) e o comportamento (através de seus métodos) de um ou mais objetos similares que possuem seus dados estruturados da mesma forma e são manipulados pelos mesmos métodos.
- ◆ Um objeto é uma instância de uma classe que é criada em tempo de execução. Classes são puramente uma descrição estática de um conjunto de possíveis objetos.

Na prática, o que significa, por exemplo, termos uma classe chamada Funcionários? Esta classe serve como modelo para a criação de objetos do tipo Funcionário. Na classe Funcionários estão as definições das propriedades e dos métodos para um objeto Funcionário. Ou seja, sempre que criarmos um objeto do tipo funcionário, o mesmo será criado com todas as propriedades e métodos da classe Funcionários.

Como eu sou apaixonado por definições simples e sem meias-palavras, adorei esta definição que encontrei na Internet: “Classe é uma forma para fazer objetos”. Acho que essa frase resume tudo.

Grande parte da funcionalidade do Framework .NET é fornecida por um grande número de classes, as quais fazem parte de “.NET Framework Class Library”, já descrita anteriormente. O Framework .NET agrupa as classes de acordo com suas funcionalidades. Um agrupamento de classes criadas para um determinado fim é também conhecido como um namespace (espaço de nomes). Por exemplo, temos o namespace System.Data. Dentro deste namespace existem várias classes que fornecem os métodos necessários para a conexão e manipulação de fontes variadas de dados. Existem classes para a conexão com o SQL Server 2000, outras para a conexão com fontes ODBC e assim por diante. A biblioteca de classes do Framework .NET é organizada de uma forma hierárquica, onde as classes de níveis inferiores herdam todas as características da classe mãe. Falaremos mais sobre herança no próximo item.

Nos Capítulos 3, 4 e 5 vamos estudar os fundamentos da linguagem C#. Veremos que tudo (ou quase tudo, como preferem alguns) no C# são classes. Até para criar um simples programa como o Hello World apresentado no Capítulo 1 e exibido novamente na Listagem 2.1, estamos criando uma classe. Observe a palavra “class” na segunda linha do exemplo.

Listagem 2.1 – Hello World !

```
using System;
class primeiroprograma
{
    // Meu primeiro programa em C#
    // O tradicional Hello World !!

    public static void Main()
    {
        string umamensagem = "Hello World !!!";
        Console.WriteLine(umamensagem);
    }
}
```

Herança

“É o mecanismo que permite definir uma nova classe a partir de uma classe já existente. A classe que está sendo criada é dita subclasse ou classe filha da classe já existente. Em contrapartida a classe já existente é chamada de superclasse da classe que está sendo criada. A subclasse herda a estrutura de dados e os métodos da superclasse, podendo adicionar variáveis na estrutura de dados herdada, bem como adicionar novos métodos e reescrever métodos herdados.

Uma classe pode possuir uma única superclasse – herança simples, ou pode conter mais do que uma superclasse – herança múltipla. A herança múltipla tem sido alvo de muitas discussões e controvérsias. A única linguagem, do Framework .NET, que implementa diretamente a herança múltipla é o C++. A grande discussão em torno da herança múltipla tem a ver com a relação custo x benefício, uma vez que a mesma é de difícil implementação e concepção, embora os benefícios nem sempre sejam os esperados.

Herança é um conceito fundamental para a orientação a objetos. Através do mecanismo de herança podemos criar uma classe baseada em outra já existente. A nova classe que está sendo criada “herda” todas as propriedades e métodos da classe base, também chamada de classe mãe ou superclasse conforme descrito anteriormente. A herança evita que as propriedades e métodos da classe mãe tenham que ser redefinidos na classe filho, embora a classe filho ou subclasse possa redefinir os métodos da classe mãe, através de um mecanismo conhecido como Overwrite.

Para ilustrar o mecanismo de herança vamos a um exemplo prático. Vamos imaginar que você esteja projetando um programa – baseado na orientação a objetos, evidentemente, para um banco. Uma das prováveis classes seria a classe Clientes. Nesta classe poderíamos definir as características e métodos básicos para um cliente típico.

Para a classe Clientes poderíamos definir as seguintes propriedades:

- ◆ Nome
- ◆ CPF
- ◆ RG
- ◆ Identidade
- ◆ Endereço
- ◆ Cidade
- ◆ CEP
- ◆ Fone
- ◆ e-mail

Para a classe Clientes poderíamos definir os seguintes métodos:

- ◆ Cadastrar
- ◆ Excluir
- ◆ Atualizar

Observe que a nossa classe Clientes possui apenas as propriedades e métodos comuns a qualquer cliente, quer seja um grande cliente ou um pequeno correntista. Este é um dos princípios da utilização de classes: nas classes de primeiro nível definimos apenas as propriedades e métodos comuns, os quais deverão ser utilizados pelas classes dos demais níveis.

Na Figura 2.8 temos uma visão geral da classe Clientes.

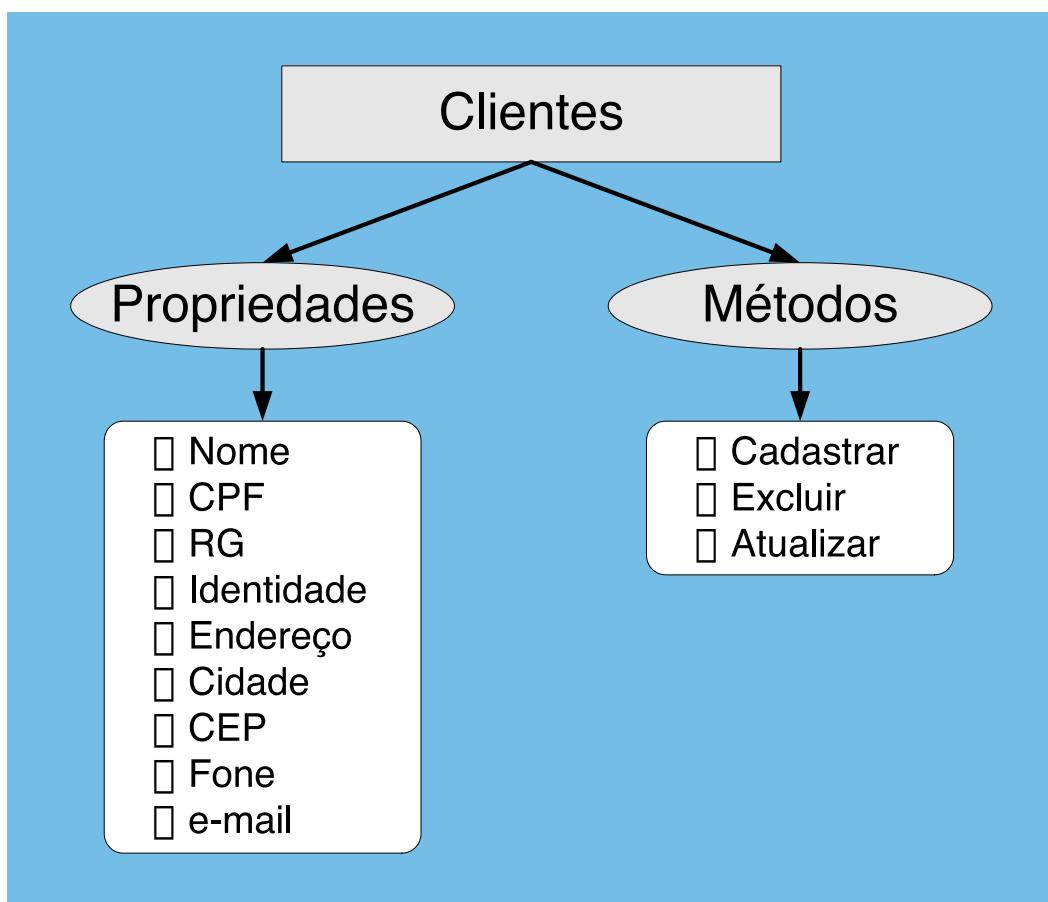


Figura 2.8: A classe Clientes.

Continuando o nosso exemplo, o banco tem clientes com características diferenciadas. Por exemplo:

- ◆ Correntistas
- ◆ Contas de poupança
- ◆ Empréstimos pessoais
- ◆ Financiamentos habitacionais

Cada um destes tipos de clientes possui propriedades e relações diferenciadas com o banco. Por isso precisamos de objetos que representem estas diferenças. A primeira sugestão seria criar uma classe para cada tipo de cliente. A título de exemplo poderíamos criar as seguintes classes:

- ◆ ClienteCorrentista
- ◆ ClientePoupança
- ◆ ClienteEmpréstimo
- ◆ ClienteHabitacional

Ao criar a classe ClienteCorrentista podemos criá-la como uma subclasse da classe Clientes. Com isso, a classe ClienteCorrentista herda todas as propriedades e métodos da classe pai – Clientes. Somente precisaremos implementar as propriedades e métodos que são específicas da subclasse ClienteCorrentista.

A título de exemplo poderíamos definir algumas propriedades e métodos da classe ClienteCorrentista.

Para a classe ClienteCorrentista poderíamos definir as seguintes propriedades:

- ◆ NumConta
- ◆ NumAgência
- ◆ Limite
- ◆ Categoria

Para a classe ClienteCorrentista poderíamos definir os seguintes métodos:

- ◆ DefinirLimite
- ◆ AumentarLimite
- ◆ BloquearConta
- ◆ LiberarConta

Juntando as propriedades e classes que foram herdadas da classe pai – Clientes, a classe ClienteCorrentista teria as seguintes propriedades:

- ◆ Nome (herdada da classe pai)
- ◆ CPF (herdada da classe pai)
- ◆ RG (herdada da classe pai)
- ◆ Identidade (herdada da classe pai)
- ◆ Endereço (herdada da classe pai)
- ◆ Cidade (herdada da classe pai)
- ◆ CEP (herdada da classe pai)
- ◆ Fone (herdada da classe pai)
- ◆ e-mail (herdada da classe pai)
- ◆ NumConta
- ◆ NumAgência
- ◆ Limite
- ◆ Categoria

Juntando as propriedades e classes que foram herdadas da classe pai – Clientes, a classe ClienteCorrentista teria as seguintes propriedades:

- ◆ Cadastrar (herdada da classe pai)

- ◆ Excluir (herdada da classe pai)
- ◆ Atualizar (herdada da classe pai)
- ◆ DefinirLimite
- ◆ AumentarLimite
- ◆ BloquearConta
- ◆ LiberarConta

Na Figura 2.9 temos uma visão geral da classe ClienteCorrentista, já incluindo as propriedades e métodos herdados da classe pai – Clientes.

O mesmo raciocínio também é válido para a criação das classes ClientePoupança, ClienteEmpréstimo e ClienteHabitacional, como subclasses da classe pai Clientes.

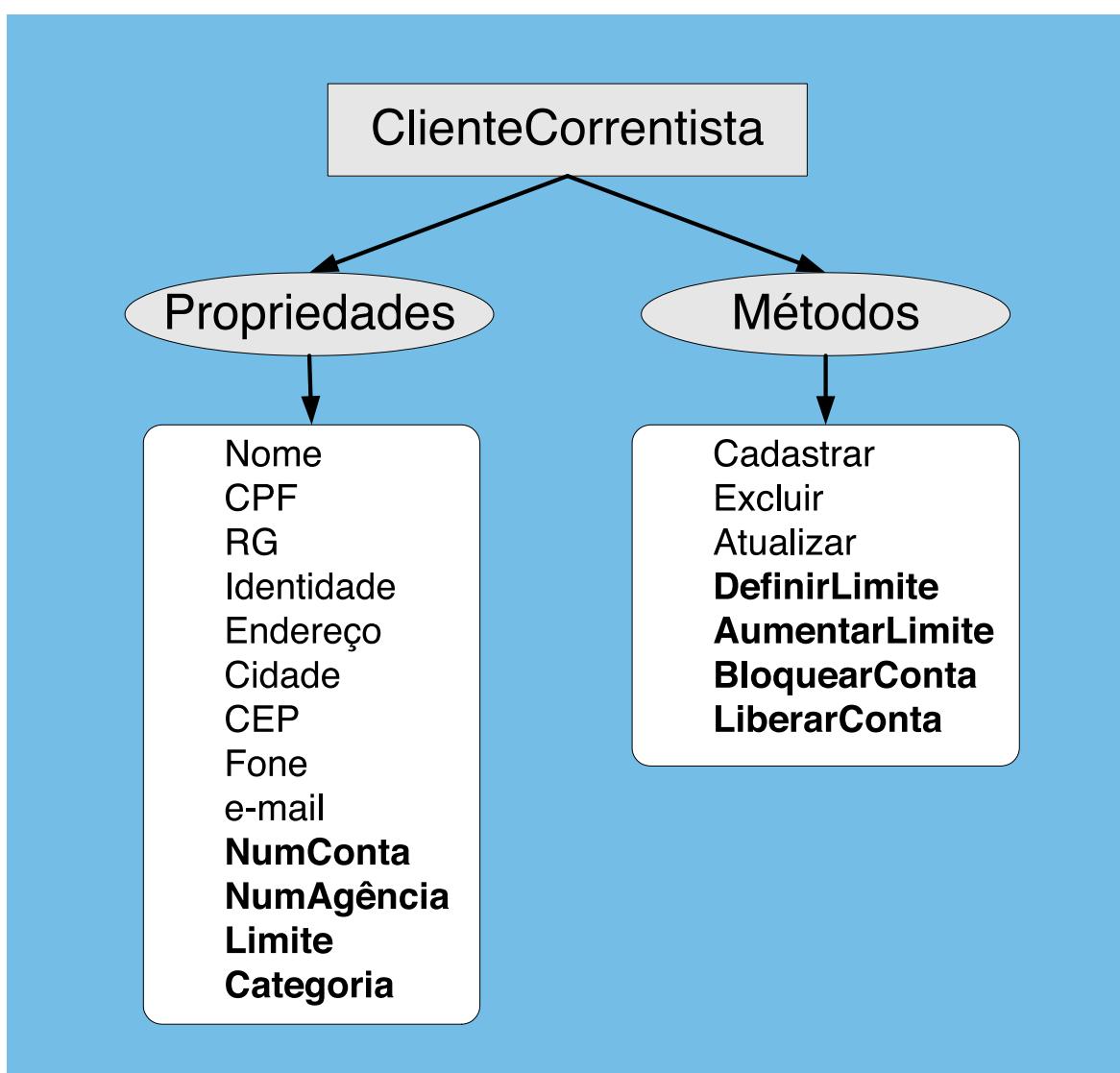


Figura 2.9: A classe Clientes.

Caso haja necessidade, uma subclasse pode sobrepor um método herdado da classe pai. Por exemplo, a subclasse ClienteCorrentista poderia sobrepor o método Cadastrar da classe pai – Clientes. Isso pode ser feito para que sejam atendidas necessidades específicas do cadastramento de um cliente correntista. As linguagens do Framework .NET suportam esta técnica, a qual também é conhecida como Overwrite.

Após termos criado as demais classes do nosso exemplo, estaremos com a hierarquia de classes ilustrada na Figura 2.10.

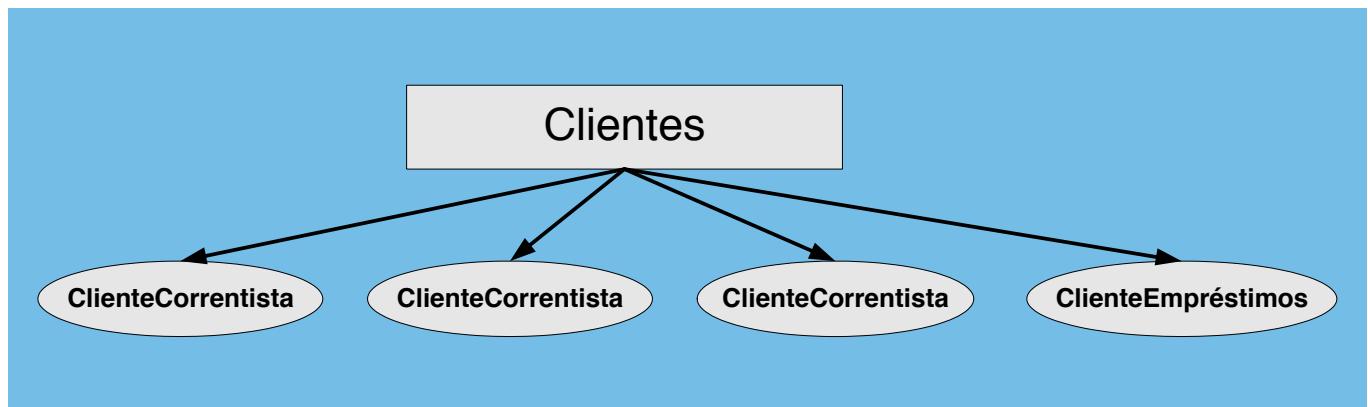


Figura 2.10: Uma hierarquia de classes.

Conforme descrevemos anteriormente, uma hierarquia de classes também pode ser chamada de namespace ou “espaço de nomes”. Também já descrevemos antes que “.NET Framework Class Library” é, na verdade, um grande número de namespaces, cada um com funcionalidades específicas, funcionalidades estas disponibilizadas por um grande número de classes dentro de cada um dos namespaces. Também gostaria de reforçar que este é um dos grandes atrativos do .NET, ou seja, oferecer uma infinidade de classes, as quais facilitam muito o desenvolvimento das aplicações e, ainda por cima, classes estas que podem ser acessadas a partir de qualquer linguagem do Framework .NET.

Reutilização de Código

Ao utilizar uma das classes do Framework .NET qual o princípio de orientação a objetos que estamos utilizando? Herança. Certo. Mas não era bem deste princípio que eu gostaria de tratar. Então vamos colocar as coisas de uma maneira um pouco diferente: “Ao utilizarmos a herança, qual o princípio de orientação a objetos que estamos utilizando? Reutilização ou reaproveitamento de código ou, de uma maneira mais “chique”: Reusabilidade.

Ao invés de termos que implementar a funcionalidade de determinada classe, em cada programa onde a funcionalidade for necessária, simplesmente criamos uma classe que herda estas funcionalidades, já prontas, de uma classe do Framework .NET. Com isso estamos reutilizando o código já desenvolvido. Isso poupa trabalho e facilita o desenvolvimento das aplicações. Melhor ainda. Vamos imaginar que tenha sido descoberto um pequeno “bug” em um método de uma classe. Ao corrigirmos este “bug” na classe em questão, todos os programas que utilizam a classe passarão a ter acesso às modificações, ou seja, irão herdar as alterações. Isso facilita, enormemente, a manutenção e alteração das aplicações. Mas era de esperarmos tal comportamento, uma vez que reutilização de código, herança e facilidade de manutenção dos programas são alguns dos princípios básicos da orientação a objetos.

Em uma empresa, a equipe de desenvolvimento pode criar classes básicas, as quais podem ser utilizadas em diversos programas da empresa. Sempre que forem feitas alterações nas classes básicas, as alterações serão herdadas por todos os programas que utilizam as classes básicas. Vejam que este cenário é muito diferente do que acontece, na prática, hoje. Muitas vezes cada projeto de desenvolvimento parte do zero, sem que nada seja reaproveitado de projetos anteriores. Com isso, uma série de funcionalidades básicas precisam ser reescritas a cada novo programa. Desta forma temos projetos mais longos, com maior custo e que muito raramente são concluídos no prazo.

Mais Alguns Detalhes Antes de Iniciarmos a Parte Prática

Conforme já foi descrito anteriormente, nos próximos três capítulos estaremos tratando da linguagem C#. Nos capítulos restantes do livro, estaremos utilizando o C# para a criação de páginas ASP.NET.

Para que você possa acompanhar os exemplos apresentados nos capítulos 3, 4 e 5 é necessário que o Framework .NET tenha sido instalado com sucesso e que o compilador do C# esteja disponível.

A seguir coloco os passos necessários para que você verifique se está tudo OK com o computador que você utilizará para acompanhar os exemplos dos capítulos 3, 4 e 5.

Para verificar se o Framework .NET foi instalado com sucesso:

1. Verifique se a opção “Microsoft .NET Framework SDK” foi adicionada ao menu Programas (Iniciar -> Programas -> Microsoft .NET Framework SDK).
2. Verifique se a pasta “\Arquivos de programas\Microsoft.NET\FrameworkSDK” foi criada no drive onde está instalado o Windows 2000.

NOTA: Para orientações sobre o download e a instalação do Framework .NET consulte a Introdução deste livro.

Para verificar se o compilador C3 está instalado e funcionando:

1. Abra o Bloco de Notas (Iniciar -> Programas -> Acessórios -> Bloco de Notas).
2. Digite o código indicado na Listagem 2.2:

Listagem 2.2 – Hello World !

```
using System;
class teste
{
    // Aprendendo C#.
    // Um exemplo simples.

    public static void Main()
    {
        Console.WriteLine("Testando o compilador !!!");
    }
}
```

NOTA: Se você está utilizando o Windows 2000 em Inglês, ao invés de Arquivo de Programas, procure na pasta Program Files.

3. Selecione o comando Arquivo -> Salvar como...
4. Na lista Salvar em, navegue até a pasta C:\Meus documentos.
5. Na lista “Salvar como tipo:”, selecione a opção Todos os arquivos.
6. No campo “Nome do arquivo:”, digite teste.cs
7. Sua janela deve estar conforme indicado na Figura 2.11.
8. Dê um clique no botão Salvar.
9. Feche o Bloco de Notas.
10. Abra um Prompt de comando (Iniciar -> Programas -> Acessórios -> Prompt de comando).
11. Navegue até a pasta Meus documentos.
12. Por exemplo, se a pasta Meus documentos está no drive C: utilize os seguintes comandos para navegar até a pasta Meus documentos:

C:

cd "Meus documentos"

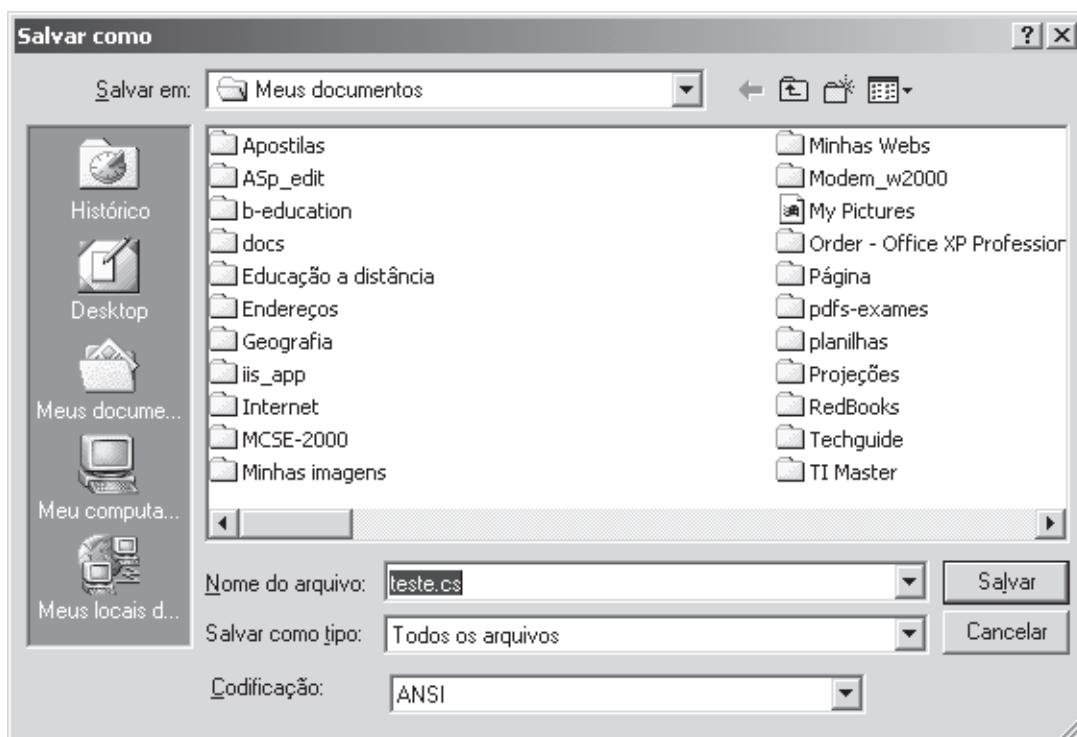


Figura 2.11: Salvando o arquivo teste.cs

13. Observe que o nome da pasta Meus documentos vem entre aspas porque a mesma possui espaço em branco no nome. Se não colocarmos aspas o comando cd (change directory) irá gerar um erro.
14. O prompt deve indicar que você está na pasta Meus documentos, conforme indicado na Figura 2.12.

```
D:\>c:
C:\>cd "Meus documentos"
C:\Meus documentos>_
```

Figura 2.12: Navegando até a pasta Meus documentos.

15. Execute o seguinte comando:

```
csc teste.cs
```

16. Se o compilador C# estiver corretamente instalado, o programa teste.cs será compilado e será gerado um arquivo chamado teste.exe. Para executar este arquivo basta digitar teste e pressionar Enter. Você obterá os resultados indicados na Figura 2.13.

```
C:\Meus documentos>csc teste.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\Meus documentos>dir teste.exe
 0 volume na unidade C é PRINCIPAL
 0 número de série do volume é 2629-07E6

 Pasta de C:\Meus documentos

12/07/2001  22:20           3.072 teste.exe
              1 arquivo(s)      3.072 bytes
              0 pasta(s)  1.577.115.648 bytes disponíveis

C:\Meus documentos>teste
Testando o compilador !!!
C:\Meus documentos>_
```

Figura 2.13: Executando o programa teste.exe.

Observe a saída do programa que é simplesmente a mensagem: Testando o compilador !!! Isto comprova que o compilador C# está instalado e funcionando.

Conclusão

Já temos todas as condições para iniciar a parte prática. Já foram apresentados os aspectos teóricos necessários ao entendimento do .NET. Agora é hora de começar a trabalhar.

Então, nada mais a concluir. Já conversei bastante. Vamos à prática.

Introdução

Neste capítulo apresentaremos a linguagem C# – lê-se C Sharp. Vamos explicar o porquê de mais uma linguagem de programação, sendo que já temos tantas. Também falaremos sobre as características que tornarão o C# uma alternativa atraente para o desenvolvimento de soluções corporativas.

Em seguida veremos “a cara” de um programa escrito em C#. Vamos detalhar qual a estrutura de um programa, quais os elementos obrigatórios. Vamos apresentar os elementos que compõem um programa C#. Também falaremos dos comandos básicos para ler e escrever na tela, de tal forma que possamos apresentar alguns exemplos simples.

Vamos entender como fazer uso das Classes do .NET Framework Class Library. Veremos que temos duas opções para usar métodos e propriedades das classes do .NET Framework Class Library. Explicaremos o uso da instrução Using e veremos como os comandos de um programa são diferentes dependendo de utilizarmos a instrução Using ou não utilizarmos.

Vamos falar sobre os tipos básicos disponíveis no C#. Veremos que os tipos são disponibilizados pelo Framework .NET, no que é conhecido como CTS – Common Type System. O CTS disponibiliza um conjunto de tipos padronizados, o qual está disponível para qualquer linguagem do Framework .NET. Isso faz com que um tipo Int32, por exemplo, tenha a mesma estrutura e ocupe a mesma quantidade de memória, quer seja em um programa escrito em C# ou VB.NET ou qualquer outra linguagem do Framework .NET. A disponibilização de um sistema de tipos padronizado facilita a integração entre programas escritos em diferentes linguagens.

NOTA: Para maiores detalhes sobre o CTS consulte os Capítulos 1 e 2.

Na seqüência apresentaremos as estruturas da linguagem C#, para o controle do fluxo do programa. Veremos que estas estruturas são muito semelhantes às encontradas na linguagem C++, com algumas pequenas diferenças.

Também apresentaremos os principais operadores matemáticos, lógicos e de atribuição. Novamente veremos uma certa semelhança com a linguagem C++.

Juntamente com os conceitos apresentados estaremos mostrando a aplicação dos mesmos, através do uso de exemplos simples e didáticos. Se você já programa em C++ ou Java e tem um bom entendimento dos conceitos de orientação a objetos, verá que aprender C# será uma tarefa muito simples, rápida e agradável. Se você

CAPÍTULO 3

Apresentando o C#

for programador iniciante, não tem problema. Os exemplos apresentados irão ajudá-lo a entender a estrutura e os comandos da linguagem.

O código-fonte com os exemplos dos capítulos 3, 4 e 5 estão disponíveis para download no site da editora Axcel Books: www.axcel.com.br.

Antes de começarmos faça o seguinte:

Crie uma pasta chamada ExCSharp no drive C, ou em outro drive qualquer do computador que você está utilizando para acompanhar os exemplos deste livro. Na pasta ExCSharp crie uma subpasta chamada Cap3, outra chamada Cap4 e mais uma chamada Cap5. Colocarei, em cada uma das subpastas, os exemplos dos capítulos respectivos. Dentro destas pastas ficarão gravados dois arquivos para cada exemplo. Um arquivo com a extensão .cs, o qual é o arquivo com o código-fonte, e um arquivo .exe que é o resultado da compilação do arquivo .cs.

Mais uma Linguagem de Programação?

Sim, mais uma.

Para resumir em poucas palavras quais os objetivos da Microsoft com a criação do C#, eu diria o seguinte: “Uma nova linguagem, criada para ter a simplicidade e facilidade de desenvolvimento do Visual Basic, aliadas com o poder do C++”. Vamos resumir um pouco mais:

- ◆ Simples
- ◆ Poderosa
- ◆ Fácil

Além de poder e simplicidade, sendo uma linguagem do Framework .NET, o C# pode se beneficiar de toda a funcionalidade disponibilizada pelo mesmo.

Além disso podemos utilizar o C# para qualquer tipo de desenvolvimento, desde aplicações de linha de comando, também conhecidas como aplicações de Console, passando por aplicações gráficas Win32 tradicionais, até aplicações Web com ASP.NET. Aliás é justamente de criação de aplicações Web com a dobradinha ASP.NET/C# que estaremos tratando do Capítulo 5 até o final deste livro.

Nas versões anteriores, para a criação de páginas ASP nós tínhamos basicamente duas opções: VBScript ou JScript. Para a criação de aplicações WIn32 tínhamos várias opções: VB, C++, etc.

Para aplicações do tipo console, basicamente o C++. Agora, com o Framework .NET podemos criar qualquer tipo de aplicação com qualquer linguagem. Se um dia tivermos um Cobol habilitado ao .NET, poderemos criar aplicações Web com a dobradinha ASP.NET/Cobol. Parece estranho? Que tal um FORTRAN habilitado ao .NET?

O C# foi a linguagem utilizada para criar a maioria dos componentes do Framework .NET, o que nos leva a crer que o C# será a indicação da Microsoft, como linguagem para o desenvolvimento de aplicativos empresariais, baseados no .NET.

Também poderíamos questionar se C# é realmente uma nova linguagem ou um “C++ remodelado com cara de Java”. As interpretações podem variar de acordo com a opinião pessoal de cada um, mas o fato é que o C# é muito semelhante ao C++, com exceção de alguns recursos e comandos que foram retirados.

Uma das diferenças tem a ver com a utilização de “ponteiros”. No C# não temos ponteiros. Para os programadores C/C++ a não existência de ponteiros soa como uma “heresia”. Porém com o Framework .NET, muitas das funcionalidades que tinham que ser codificadas no próprio programa foram passadas para o controle do Framework .NET. Muitas das operações avançadas que eram implementadas com a utilização de ponteiros passaram a ser automaticamente gerenciadas pelo Framework .NET. O exemplo mais típico é a alocação e liberação de memória pelo programa. Estas funções são automaticamente tratadas pelo Framework .NET, liberando o programa para tratar apenas da sua própria funcionalidade e não de toda a infraestrutura necessária ao seu correto funcionamento. Lembrando do Capítulo 1, que o código das aplicações .NET é chamado de “managed code”, onde operações inseguras como o acesso direto à memória não são permitidas. Estas operações estão a cargo do Framework .NET.

Que aprender C++ não é uma tarefa simples nós já sabemos. E aprender C#? É uma tarefa simples. Você poderá constatar isso na prática, à medida que formos avançando neste livro.

A seguir listo alguns recursos do C++ que foram modificados no C#:

- ◆ Não existem ponteiros no C#. Gerenciamento de memória é “coisa” do Framework .NET.
- ◆ Uma série de operadores do C++ como por exemplo: ::, . e -> foram substituídos por um único operador no C#: . (ponto)

Outra melhoria significativa do C# em relação ao C++ foi o gerenciamento de exceções. Veremos mais detalhes nos próximos capítulos.

Como é “a cara” de um Programa Escrito em C#?

Vamos apresentar o todo e depois explicar as partes. Em outras palavras: Vamos apresentar um programa completo, porém bastante simples, em C#. Além do código-fonte iremos compilar e executar o programa. O nosso programa de exemplo solicita que sejam digitados dois números inteiros e exibe o resultado da adição e da multiplicação entre os números digitados. Após criarmos e testarmos o programa, iremos às devidas explicações. Vamos salvar o código-fonte como ex1cap3.cs. Ao compilarmos o código-fonte será gerado um arquivo chamado ex1cap3.exe.

Criando o programa ex1cap3.cs:

1. Abra o Bloco de Notas (Iniciar -> Programas -> Acessórios -> Bloco de Notas).
2. Digite o código indicado na Listagem 3.1:

Listagem 3.1 – Um exemplo simples – ex1cap3.cs

```
using System;

class ex1cap3
{
    // Exemplo1 - Capítulo 3.
    // Entrada e saída com C#
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {
        // Declaração das variáveis.

        Int32 Adicao;
        Int32 Produto;

        // Entrada dos valores de x e y

        Console.Write("Digite o primeiro valor inteiro ->");
        String Aux1=Console.ReadLine();

        Console.Write("Digite o segundo valor inteiro ->");
        String Aux2=Console.ReadLine();

        // Cálculo da adição e do produto.

        Adicao = Convert.ToInt32(Aux1) + Convert.ToInt32(Aux2);
        Produto = Convert.ToInt32(Aux1) * Convert.ToInt32(Aux2);

        // Exibição dos resultados.

        Console.WriteLine("O valor da soma é -> {0}",Adicao);
        Console.WriteLine("O valor da Produto é -> {0}",Produto);
    }
}
```

3. Selecione o comando Arquivo -> Salvar como...
4. Na lista Salvar em, navegue até a pasta C:\ExCsharp\Cap3.
5. Na lista “Salvar como tipo:”, selecione a opção Todos os arquivos.
6. No campo “Nome do arquivo:”, digite ex1cap3.cs.
7. Dê um clique no botão Salvar.
8. Feche o Bloco de Notas.
9. Abra um Prompt de comando (Iniciar -> Programas -> Acessórios -> Prompt de comando).
10. Navegue até a pasta C:\ExCsharp\Cap3.
11. Execute o seguinte comando: csc ex1cap3.cs

NOTA: Os passos 9, 10 e 11 são utilizados para a compilação de um programa C#. Podemos resumir estes passos da seguinte maneira:
Navegar até a pasta onde está o arquivo fonte (.cs) e executar o comando csc nome_do_arquivo.cs.
Nos próximos exemplos não irei repetir estes três passos detalhadamente. Irei resumir os três passos em um único: Navegue para a pasta onde está o arquivo .cs e execute o comando csc nome_do_arquivo.cs.

12. Se o compilador C# estiver corretamente instalado, o programa teste.cs será compilado e será gerado um arquivo chamado ex1cap3.exe.
13. Agora vamos testar o nosso exemplo. Para executar o nosso exemplo basta digitar ex1cap3 e pressionar Enter.
14. Surge uma mensagem pedindo que você digite o primeiro valor. Digite 10 e pressione Enter.
15. Surge uma mensagem pedindo que você digite o segundo valor. Digite 20 e pressione Enter.
16. O programa exibe o resultado da soma e da adição dos valores digitados. Você obterá os resultados indicados na Figura 3.1.

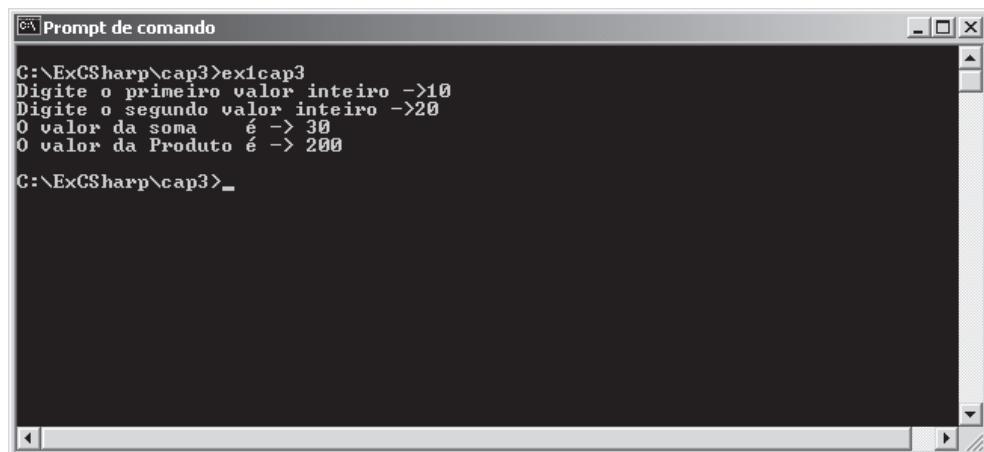


Figura 3.1: Execução do programa ex1cap3.exe.

IMPORTANTE: O C# é “case sensitive”, ou seja, o mesmo faz diferença entre letras maiúsculas e minúsculas. Por exemplo, se você declarar uma variável i minúscula e, por engano, utilizar em uma expressão I maiúsculo será gerado um erro de compilação. Outro exemplo, o comando Console.WriteLine deve ser escrito desta maneira; se você escrever, por exemplo, Console.writeline será gerado um erro de compilação.

Agora vamos às devidas explicações sobre o nosso primeiro exemplo.

Vamos analisar o seguinte trecho de código:

using System;

A cláusula using é utilizada para que possamos utilizar as classes, métodos e propriedades de um determinado namespace. No nosso exemplo estamos fazendo referência ao namespace System. Ao fazermos referência ao namespace System, passamos a ter acesso a todos os seus métodos, classes e propriedades. Por exemplo, abaixo de System, existe uma classe chamada Console – System.Console. Na Classe System.Console temos um método chamado WriteLine – System.Console.WriteLine. Observe que estamos utilizando este método em nosso programa. Porém, ao invés de digitar System.Console.WriteLine, utilizamos somente Console.WriteLine. Isso é possível pois estamos fazendo referência ao namespace System, através da cláusula using. Se não fizéssemos referência ao namespace System, teríamos que utilizar a nomenclatura completa para o método WriteLine – System.Console.WriteLine. Observe que, ao fazermos referência a um ou mais namespaces, estamos simplificando o nosso código.

Embora a utilização de referências não seja obrigatória, é uma prática recomendada, uma vez que simplifica o código-fonte. No nosso exemplo, se não tivéssemos feito referência ao namespace System, o nosso programa ficaria da seguinte maneira:

```
class ex1cap3
{
    // Exemplo1 - Capítulo 3.
    // Entrada e saída com C#
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {
        // Declaração das variáveis.

        System.Int32 Adicao;
        System.Int32 Produto;

        // Entrada dos valores de x e y
        System.Console.Write("Digite o primeiro valor inteiro ->");
    }
}
```

```

System.String Aux1=System.Console.ReadLine();

System.Console.Write("Digite o segundo valor inteiro ->");
System.String Aux2=System.Console.ReadLine();

// Cálculo da adição e do produto.

Adicao = System.Convert.ToInt32(Aux1) + System.Convert.ToInt32(Aux2);
Produto = System.Convert.ToInt32(Aux1) * System.Convert.ToInt32(Aux2);

// Exibição dos resultados.

System.Console.WriteLine("O valor da soma é -> {0}",Adicao);
System.Console.WriteLine("O valor da Produto é -> {0}",Produto);
}

}

```

Observe que neste caso teríamos que utilizar sempre a nomenclatura completa. Também cabe observar a maneira de acessar um método ou propriedade de uma classe em C#: “Utilizamos o nome do namespace (System), um ponto (System.), o nome da classe (System.Console), mais um ponto (System.Console.) e por último o nome do método e os parâmetros que o método deve receber (System.Console.WriteLine(“O valor da soma é -> {0}”,Adicao);). Com a utilização da nomenclatura completa o código fica mais longo e de difícil leitura.

Abaixo temos um exemplo no qual é feita referência a vários namespaces:

```

using System;
using System.Data;
using System.Net;
using System.Security;

```

Vamos analisar o seguinte trecho de código:

```

class ex1cap3
{
    // Exemplo1 - Capítulo 3.
    // Entrada e saída com C#
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

```

No C# toda a lógica de um programa deve estar contida dentro de uma classe. Na verdade, para ser mais precisos, teríamos que modificar a frase anterior para que a mesma ficasse assim: “No C# toda a lógica de um programa deve estar contida dentro de uma definição de tipo, sendo que uma definição de tipo pode ser uma classe, uma estrutura ou

outro tipo qualquer definido no Framework .NET. Para o nosso estudo de introdução ao C#, ficaremos com a primeira frase, ou seja, toda a lógica dos nossos exemplos estará contida dentro de uma classe.

No nosso caso estamos definindo uma classe chamada ex1cap3. Um detalhe importante é que o nome da classe não precisa ser igual ao nome do arquivo .cs. Por exemplo, poderíamos ter uma classe ex1cap3.cs e gravarmos o código-fonte em um arquivo chamado admult.cs.

Todo o conteúdo de uma classe (métodos, propriedades, etc.) deve estar entre chaves. Na linha após a definição do nome da classe, temos a abertura das chaves. Na última linha do programa temos o fechamento das chaves.

As linhas iniciadas com duas barras (//) são linhas de comentário. Qualquer semelhança com C/C++ não é mera coincidência. Como temos múltiplas linhas de comentários também poderíamos ter utilizado a sintaxe a seguir:

```
/* Exemplo1 - Capítulo 3.  
Entrada e saído com C#  
Por: Júlio Battisti  
MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA */
```

Vamos analisar o seguinte trecho de código:

```
public static void Main()
```

Após a chave de abertura da classe (colocada logo após a definição da classe), começamos a definir os métodos e propriedades da classe ex1cap3. Todo programa em C# deve conter um método chamado Main(). Este método funciona como o ponto de entrada da aplicação. Em outras palavras, a execução da aplicação se inicia pelo método Main(). Dentro do método Main() podemos fazer chamadas para outros métodos da classe ex1cap3 ou de outras classes. Novamente qualquer semelhança com o C/C++ não é mera coincidência.

Vamos analisar o seguinte trecho de código:

```
{  
  
    // Declaração das variáveis.  
  
    Int32 Adicao;  
    Int32 Produto;  
  
    // Entrada dos valores de x e y  
  
    Console.Write("Digite o primeiro valor inteiro ->");  
    String Aux1=Console.ReadLine();  
  
    Console.Write("Digite o segundo valor inteiro ->");
```

```

String Aux2=Console.ReadLine();

// Cálculo da adição e do produto.

Adicao = Convert.ToInt32(Aux1) + Convert.ToInt32(Aux2);
Produto = Convert.ToInt32(Aux1) * Convert.ToInt32(Aux2);

// Exibição dos resultados.

Console.WriteLine("O valor da soma     é -> {0}",Adicao);
Console.WriteLine("O valor da Produto é -> {0}",Produto);

}

```

Este trecho de código é o corpo do método Main().

Iniciamos com a declaração de duas variáveis do tipo Int32:

```

Int32 Adicao;
Int32 Produto;

```

Em seguida utilizamos o método Write da classe Console (Console.Write), para exibir uma mensagem na tela. Na próxima linha o método ReadLine() da classe Console (Console.ReadLine) exibe um cursor e fica aguardando que o usuário digite alguma coisa e pressione Enter. Ao pressionar Enter, o valor digitado pelo usuário é passado, no formato de uma string de texto, para a variável Aux1:

```

Console.Write("Digite o primeiro valor inteiro ->");
String Aux1=Console.ReadLine();

```

Ao final da execução destas duas linhas temos o valor digitado pelo usuário armazenado na variável Aux1, porém no formato de uma string de texto. As mesmas explicações são válidas para a variável Aux2.

O próximo passo é calcular o valor da adição, que será armazenado na variável Adicao e da multiplicação, que será armazenado na variável Produto. Porém as variáveis Aux1 e Aux2 são do tipo texto. Não podemos adicionar e multiplicar variáveis do tipo texto. Para converter o valor destas variáveis para o tipo Int32 (inteiro de 32 bits), utilizamos o métodoToInt32, da classe Convert, conforme indicado pelas duas linhas a seguir:

```

Adicao = Convert.ToInt32(Aux1) + Convert.ToInt32(Aux2);
Produto = Convert.ToInt32(Aux1) * Convert.ToInt32(Aux2);

```

Uma vez calculados os valores, o passo final é exibi-los na tela. Isso é feito como o método WriteLine da classe Console. Um detalhe interessante é a utilização do marcador {0} com esse método. O {0} é substituído pelo valor da primeira variável passada para o método, que no nosso exemplo é a variável Adicao:

```

Console.WriteLine("O valor da soma     é -> {0}",Adicao);

```

Neste caso o {0} é substituído pelo valor da variável adição. As mesmas considerações são válidas para a variável Produto, conforme indicado na linha a seguir:

```
Console.WriteLine("O valor da Produto é -> {0}", Produto);
```

Após a chave de fechamento do método Main(), temos a chave de fechamento da classe ex1cap3. Cabe observar que para a classe ex1cap3 temos um único método definido, que é justamente o método que é obrigatório para todas as classes: Main().

Outra observação importante é que todo comando é encerrado com um ponto-e-vírgula (;).

Com isso, podemos apresentar a estrutura geral de um programa em C#:

```
using namespace1;
using namespace2;
...
using namespaceN;

class nome_da_classe
{

    public static void Main()
    {

        Comandos do método Main()

    }

    Definição de outros métodos e propriedades.

}
```

Agora que já conhecemos a “cara” de um programa C#, vamos começar a estudar os diversos elementos da linguagem. Vamos iniciar falando sobre os tipos de dados disponíveis no C#.

Tipos da Linguagem C#

No C# temos os seguintes tipos:

- ◆ Value Types – Tipos de valor.
- ◆ Reference Types – Tipos de referência.

Vamos iniciar o nosso estudo pelos Value Types.

Value Types

Um tipo Value Type contém o valor da variável. Por conter o valor da variável e não uma referência (referência=endereço de memória) à variável, valores do tipo Value Type não podem conter o valor Null.

Outro detalhe importante a ser observado é que tudo no C#, mesmo os tipos mais simples como um inteiro, é considerado um objeto. Por exemplo, o tipo inteiro é definido a partir da Estrutura System.Int32 (ou System.Int16 dependendo do tamanho necessário). Ao criarmos uma variável do tipo System.Int32 estamos criando uma instância desta estrutura, o que na prática significa criar um objeto baseado na estrutura System.Int32. Esta estrutura possui métodos. Um dos métodos, a título de exemplo, é o método ToString, o qual converte o valor inteiro na string de texto correspondente.

Vamos a um exemplo. Vamos imaginar que as variáveis “a” e “b” são do tipo Value Type. Também vamos imaginar que a variável “a” contém o valor 10 e a variável “b” contém o valor 15. Agora considere o seguinte comando:

```
a = b
```

Neste caso o valor da variável “b” é atribuído à variável “a”. Ou seja, após a execução deste comando, a variável “a” contém o mesmo valor da variável “b”, que no nosso exemplo é 15. Agora, se modificarmos o valor da variável “b”, o que acontece com o valor da variável “a”?

Nada. Como as variáveis são do tipo Value Type, ao fazermos a=b, apenas atribuímos o valor de “b” para “a”. Porém “a” não fica com nenhuma referência para “b”, ou seja, se “b” for modificado, “a” não será afetado.

Para entendermos bem este conceito, observe o exemplo da Listagem 3.2.

Listagem 3.2 – Um exemplo de Value Types – ex2cap3.cs

```
using System;

class ex2cap3
{
    // Exemplo2 - Capítulo 3.
    // Value Types
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {
```

```
// Declaração das variáveis.  
  
Int32 a;  
Int32 b;  
  
// Atribuo valores às variáveis a e b.  
  
a=10;  
b=15;  
  
// Atribuo o valor da variável b à variável a.  
  
a=b;  
  
// Exibo os valores das variáveis a e b.  
  
Console.WriteLine("Valor da variável a: {0}",a);  
Console.WriteLine("Valor da variável b: {0}",b);  
  
// Agora modifiro o valor da variável b.  
// E observamos que o valor de a não se alterou.  
  
b=33;  
  
Console.WriteLine("*****");  
Console.WriteLine("Após a modificação de b!!!!");  
Console.WriteLine("*****");  
Console.WriteLine("Valor da variável a: {0}",a);  
Console.WriteLine("Valor da variável b: {0}",b);  
  
}  
}
```

Compile e execute o exemplo da listagem 3.2. Você obterá os resultados indicados na Figura 3.2.

```
C:\ExCSharp\cap3>csc ex2cap3.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap3>ex2cap3
Valor da variável a: 15
Valor da variável b: 15
*****
Após a modificação de b!!!!
*****
Valor da variável a: 15
Valor da variável b: 33

C:\ExCSharp\cap3>
```

Figura 3.2: Executando o programa ex2cap3.exe.

Observe que, após modificarmos o valor da variável “b”, o valor da variável “a” manteve-se inalterado. Este é o comportamento esperado, conforme descrito anteriormente.

Em C# também temos os tipos simples presentes na maioria das linguagens. Conforme descrevemos anteriormente, um tipo simples é uma instância de uma estrutura ou classe do namespace System. Para facilitar a programação, o C# oferece aliases (apelidos) para os tipos do Framework .NET. Por exemplo, ao invés de declararmos o tipo de uma variável inteira como System.Int32, podemos utilizar o alias do C#: int.

Na Tabela 3.1 temos uma descrição dos principais tipos básicos do C#. Também apresentamos o alias correspondente, bem como uma descrição da faixa de valores de cada tipo.

Tabela 3.1 Tipos básicos do C# – value types.

Tipo	Alias no C#	Descrição
System.Byte	byte	Byte de 8 bits, sem sinal. Valor variando entre 0 e 255.
System.SByte	sbyte	Byte de 8 bits, com sinal. Valor variando entre -128 e 127.
System.Int16	short	Inteiro de 16 bits, com sinal. Valor variando entre -32.768 e 32.767.
System.UInt16	ushort	Inteiro de 16 bits, sem sinal. Valor variando entre 0 e 65.535.
System.Int32	int	Inteiro de 32 bits, com sinal. Valor variando entre -2.147.438.648 e 2.147.438.647.
System.UInt32	uint	Inteiro de 32 bits, sem sinal. Valor variando entre 0 e 4.294.967.295.

Tipo	Alias no C#	Descrição
System.Int64	long	Inteiro de 64 bits, com sinal. Valor variando entre -9.223.372.036.854.775.808 e 9.223.372.036.854.775.807.
System.UInt64	ulong	Inteiro de 64 bits, sem sinal. Valor variando entre 0 e 184.467.440.737.095.551.615.
System.Char	char	Um único caractere unicode com 16 bits.
System.Single	float	Real de 32 bits, com sinal. Valor variando entre -3,402823E38 e 3,402823E38.
System.Double	double	Real de 64 bits, com sinal. Valor variando entre -1,79769313486232E308 e 1,79769313486232E308.
System.Boolean	bool	Pode conter os valores True ou False.
System.Decimal	decimal	Real de 128 bits, com sinal. Valor variando entre -79.228.162.514.264.337.593.543.950.335 e 79.228.162.514.264.337.593.543.950.335.

Algumas observações importantes sobre os tipos básicos no C#:

- ◆ No C/C++ o valor verdadeiro, para uma variável do tipo bool, é representado por qualquer valor diferente de zero. No C# o valor verdadeiro não é mais representado por qualquer valor diferente de zero.
- ◆ O tipo decimal é utilizado para valores que necessitam de uma alta precisão, como por exemplo valores para cálculos financeiros e monetários. Também devemos destacar que a precisão é dada em dígitos e não em casas decimais. As operações são precisas até um máximo de 28 casas decimais.

Agora vamos falar um pouco mais sobre um tipo que nos oferece mais opções: O tipo struct.

O Tipo Struct

Através da utilização da palavra struct podemos construir estruturas complexas de dados. Por exemplo, um ponto no plano cartesiano é representado pelo valor de x e y. Podemos criar uma estrutura chamada ponto, a qual contém dois campos: x e y. Além de campos, uma estrutura pode conter constantes, métodos, propriedades, indexadores, operadores e tipos aninhados.

Pode parecer que um struct é a mesma coisa que uma classe. Porém a diferença básica é que um struct é um value type, enquanto uma classe é um reference type (falaremos mais sobre reference type mais adiante). O namespace System contém uma série de estruturas, algumas delas descritas na tabela 3.1: System.Int16, System.Int32, System.Byte, System.SByte e assim por diante.

Vamos criar um exemplo simples, onde criamos uma estrutura para representar números complexos. Um número complexo possui uma parte real e uma parte imaginária (só para lembrarmos um pouco da época do segundo grau). A seguir temos alguns exemplos de números complexos:

```
x = 2 + 3i
y = -1 + 4,5i
z = 3 - 5,2i
```

Onde i, por definição, representa a raiz quadrada de -1.

Para representar um número complexo, precisamos de dois valores do tipo single: um para a parte real e outro para a parte imaginária. Para somar dois números complexos, basta somar as partes reais e as partes imaginárias, conforme exemplo a seguir:

```
x+y = (2-1)+(3+4,5)i
x=y = 1+7,5y
```

Vamos a um exemplo simples, onde criaremos uma estrutura para representar números complexos. Nossa estrutura conterá dois campos: um para representar a parte real e outro para representar a parte imaginária. O usuário irá digitar os valores para dois números complexos e o programa fará a soma dos mesmos e exibirá os números digitados e o resultado da soma.

Para definir uma estrutura utilizamos a palavra struct. Por exemplo, vamos definir a estrutura numcomplexo, a qual será utilizada no nosso programa:

```
struct numcomplexo
{
    public float preal,pimag;
}
```

Para utilizar esta estrutura no nosso exemplo, precisamos declarar uma variável do tipo numcomplexo. Como iremos utilizar três variáveis, sendo uma delas para conter a soma, precisamos declarar três variáveis do tipo numcomplexo:

```
numcomplexo num1,num2;
numcomplexo soma;
```

Para acessar os campos individuais da estrutura utilizamos a notação tradicional da orientação a objetos, ou seja, o nome da estrutura seguida de um ponto (.) mais o nome do campo. Por exemplo, para atribuirmos um valor ao campo preal, da estrutura num1, utilizamos a seguinte sintaxe:

```
num1.preal = 3,5;
```

Agora que já conhecemos o básico sobre estruturas, vamos apresentar o nosso exemplo. Considere o exemplo da Listagem 3.3.

Listagem 3.3 – Um exemplo utilizando estruturas – ex3cap3.cs

```
using System;
// Exemplo3 - Capítulo 3.
```

```
// Utilização de estruturas.  
// Números complexos.  
// Por: Júlio Battisti  
// MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA  
  
  
struct numcomplexo  
{  
    public float preal,pimag;  
}  
  
  
class ex3cap3  
{  
  
    public static void Main()  
    {  
  
        // Declaração das variáveis.  
  
        numcomplexo num1,num2;  
        numcomplexo soma;  
  
        // Entrada dos valores para a parte real  
        // e a parte imaginária dos números num1 e num2.  
  
        Console.Write("Digite a parte real do primeiro número ->");  
        String Auxreal1=Console.ReadLine();  
  
        num1.preal = Convert.ToSingle(Auxreal1);  
  
  
        Console.Write("Digite a parte imaginária do primeiro número ->");  
        String Auximag1=Console.ReadLine();  
  
        num1.pimag = Convert.ToSingle(Auximag1);  
        Console.Write("Digite a parte real do segundo número ->");
```

```

String Auxreal2=Console.ReadLine();

num2.preal = Convert.ToSingle(Auxreal2);

Console.WriteLine("Digite a parte imaginária do segundo número ->");
String Auximag2=Console.ReadLine();

num2.pimag = Convert.ToSingle(Auximag2);

// Cálculo da adição dos números num1 e num2.
// Para isto basta somar as partes reais e imaginárias
// dos respectivos números.

soma.preal = num1.preal + num2.preal;
soma.pimag = num1.pimag + num2.pimag;

// Exibição dos resultados.

Console.WriteLine("*****");
Console.WriteLine("O primeiro complexo é -> {0} + {1}i",num1.preal,num1.pimag);
Console.WriteLine("*****");
Console.WriteLine("O segundo complexo é -> {0} + {1}i",num2.preal,num2.pimag);
Console.WriteLine("*****");
Console.WriteLine("O resultado da soma é -> {0} + {1}i",soma.preal,soma.pimag);
Console.WriteLine("*****");

}

}

```

Digite o exemplo da Listagem 3.3 e salve o mesmo em um arquivo chamado ex3cap3.cs, na pasta C:\ExCsharp\cap3. Compile e execute o exemplo da Listagem 3.3. Você obterá os resultados indicados na Figura 3.3.

Este exemplo simples serve para ilustrar a criação e utilização de estruturas. Também observe que utilizamos o método ToSingle, da classe Convert – Convert.ToSingle. A utilização deste método foi necessária, porque o valor retornado pelo comando ReadLine é do tipo String. Precisamos converter este valor para o tipo Single, para que possamos fazer a operação de adição com os mesmos. O valor retornado por ReadLine é armazenado em uma variável auxiliar do tipo String, conforme o exemplo a seguir:

```
String Auxreal1=Console.ReadLine();
```

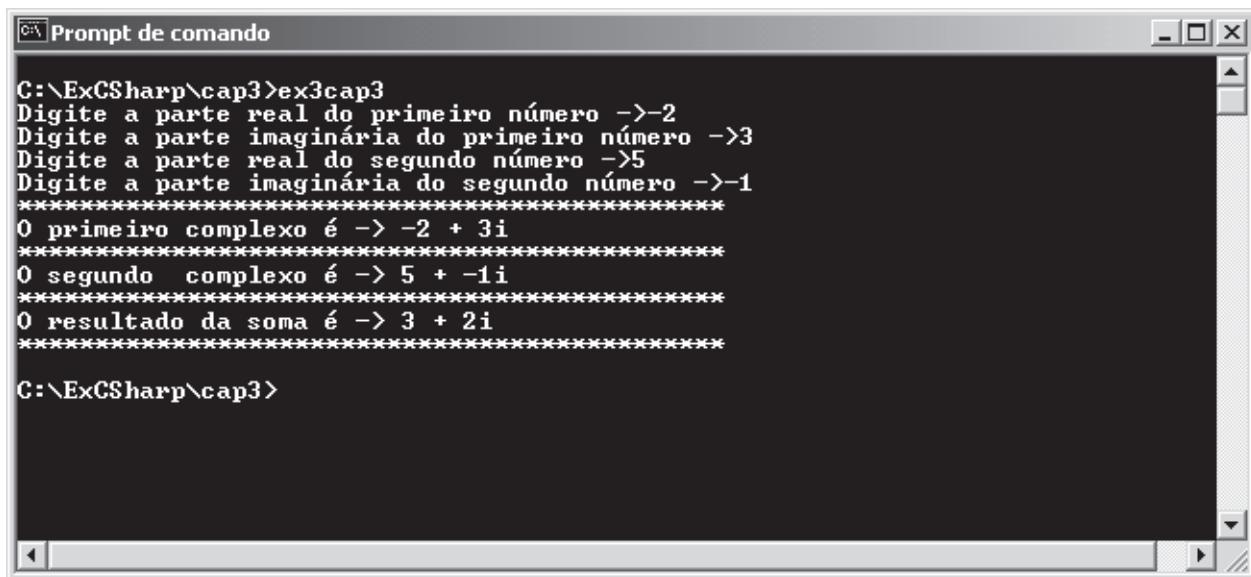


Figura 3.3: Executando o programa ex3cap3.exe.

Depois este valor é convertido para o tipo Single, antes de ser atribuído à parte real da variável num1, conforme exemplo a seguir:

```
num1.preal = Convert.ToSingle(Auxreal1);
```

Os Tipos de Enumeração

Utilizamos tipos de enumeração para criar um conjunto com valores definidos. Além do valor definido é associado um valor numérico para cada elemento do conjunto. Por padrão, o valor numérico associado é do tipo inteiro e inicia com zero.

No exemplo a seguir estamos definindo um tipo de enumeração chamado DiasDaSemana:

```
enum DiasDaSemana {Domingo, Segunda, Terça, Quarta, Quinta, Sexta, Sábado}
```

Neste caso temos associados os seguintes valores do tipo int, com cada elemento:

```
Domingo = 0
Segunda = 1
Terça = 2
Quarta = 3
Quinta = 4
Sexta = 5
Sábado = 6
```

Também podemos modificar o valor associado com o primeiro elemento do conjunto. Os valores dos demais elementos vão sendo incrementados de um em um. Vamos modificar um pouco o exemplo anterior:

```
enum DiasDaSemana {Domingo=1, Segunda, Terça, Quarta, Quinta, Sexta, Sábado}
```

Neste caso os valores inteiros associados aos elementos do conjunto seriam os seguintes:

```
Domingo = 1
Segunda = 2
Terça = 3
Quarta = 4
Quinta = 5
Sexta = 6
Sábado = 7
```

Também é possível definir valores específicos, associados com cada elemento do conjunto. Considere o exemplo a seguir:

```
enum DiasDaSemana {Domingo=1, Segunda=3, Terça=4, Quarta=8, Quinta=11, Sexta=15,
Sábado=0}
```

Também podemos fazer com que os valores associados com os elementos do conjunto sejam de um tipo diferente de int. Os tipos permitidos são os seguintes: long, int, short e byte. Para fazer com que os valores numéricos associados aos elementos do conjunto sejam de um tipo diferente, utilizamos a seguinte sintaxe:

```
enum nome_do_conjunto: tipo_desejado {Elemento1, Elemento2, ..., Elementon}
```

Considere o exemplo a seguir, onde definimos o conjunto DiasDaSemana com valores do tipo byte, associados aos elementos do conjunto:

```
enum DiasDaSemana : byte { Domingo=1, Segunda, Terça, Quarta, Quinta, Sexta, Sábado}
```

Neste caso os valores do tipo byte associados aos elementos do conjunto são os seguintes:

```
Domingo = 1
Segunda = 2
Terça = 3
Quarta = 4
Quinta = 5
Sexta = 6
Sábado = 7
```

Também podemos utilizar a seguinte sintaxe, onde fazemos a definição de cada elemento em uma linha separada, o que facilita a visualização do código:

```
enum DiasDaSemana
{
    Domingo = 1,
    Segunda = 2,
    Terça = 3,
```

```
Quarta      = 4,  
Quinta      = 5,  
Sexta       = 6,  
Sábado      = 7  
}
```

Reference Types

Os tipos de referência (reference types) não armazenam os dados reais de uma variável; ao invés disso, armazenam uma referência (endereço de memória) para o local onde estes dados estão armazenados. Quando uma variável do tipo reference type é passada como parâmetro de uma função, o endereço da variável é passado para a função e não o seu valor. Se a função modificar a variável passada como parâmetro, na verdade estará modificando a variável original.

Os tipos de referência são os seguintes:

- ◆ object
- ◆ class
- ◆ interfaces
- ◆ Delegações
- ◆ string
- ◆ Arrays

Neste capítulo vamos tratar dos tipos string e Arrays. Veremos alguns exemplos de utilização dos mesmos.

O Tipo String

Toda string (valor do tipo texto) no C# é uma instância da classe System.String do .NET Framework Class Library. Por ser uma classe, temos à disposição uma série de métodos, campos, propriedades, operadores e construtores para manipulação de strings.

Também cabe reforçar que uma string é um valor do tipo reference type.

Para exemplificar o uso de strings considere o exemplo da Listagem 3.4. Neste exemplo utilizaremos alguns métodos da classe System.String.

Listagem 3.4 – Um exemplo utilizando variáveis do tipo string – ex4cap3.cs

```
using System;  
  
class ex4cap3  
{
```

```
// Exemplo4 - Capítulo 3.  
// Entrada e saída com C#  
// Por: Júlio Battisti  
// MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA  
  
public static void Main()  
{  
    // Declaração de duas variáveis do tipo string.  
  
    string Texto1;  
    string Texto2;  
  
    //Defino valores para as variáveis Texto1 e Texto2;  
  
    Texto1 = "José da Silva";  
    Texto2 = "Maria do Socorro";  
  
    // Agora vamos utilizar o método  
    // String.ToLower()  
    // para converter as strings Texto1  
    // e Texto2 para minúsculas.  
  
    Console.WriteLine("*****");  
    Console.WriteLine("Variável Texto 1-> {0}", Texto1.ToLower());  
    Console.WriteLine("Variável Texto 2-> {0}", Texto2.ToLower());  
    Console.WriteLine("*****");  
  
    // Agora vamos utilizar o método  
    // String.ToUpper()  
    // para converter as strings Texto1  
    // e Texto2 para MAIÚSCULAS.  
  
    Console.WriteLine("Variável Texto 1-> {0}", Texto1.ToUpper());  
    Console.WriteLine("Variável Texto 2-> {0}", Texto2.ToUpper());  
    Console.WriteLine("*****");  
}  
}
```

Observe que o fato de uma string ser uma instância de uma classe (System.String) faz com que tenhamos acesso a uma série de métodos disponibilizados pela biblioteca de classes do .NET (.NET Framework Class Library). No nosso exemplo utilizamos os métodos ToLower() e ToUpper.

Digite o exemplo da Listagem 3.4 e salve o mesmo em um arquivo chamado ex4cap3.cs, na pasta C:\ExCsharp\cap3. Compile e execute o exemplo da Listagem 3.4. Você obterá os resultados indicados na Figura 3.4.

```
C:\ExCSharp\cap3>csc ex4cap3.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap3>ex4cap3
*****
Variável Texto 1-> josé da silva
Variável Texto 2-> maria do socorro
*****
Variável Texto 1-> JOSÉ DA SILVA
Variável Texto 2-> MARIA DO SOCORRO
*****
```

Figura 3.4: Executando o programa ex4cap3.exe.

Na Tabela 3.2 temos uma descrição dos principais métodos da classe System.String.

Tabela 3.2 Os principais métodos da classe System.String.

Método	Descrição
Compare()	Utilizado para comparar o valor de duas strings. Recebe como parâmetros duas string: Compare(str1,str2). Retorna um valor inteiro de 32 bits, com sinal. Se for menor do que zero, str1 é menor do que str2. Por exemplo: José é menor do que Pedro, conta a ordem alfabética. Se for igual a zero str1 é igual a str2 e, se for maior do que zero, str1 é maior do que str2.
CompareOrdinal()	Compara strings desconsiderando diferenças entre idiomas, ou seja, faz uma comparação posição a posição da string. Recebe como parâmetros duas strings.
CompareTo()	Compara uma string com a instância de um objeto recebido como parâmetro, sendo que este objeto deve ser capaz de ser avaliado como uma string.
EndsWith()	Utilizada para verificar se uma substring existe no final de uma string.
Concat()	Concatena duas ou mais strings passadas como parâmetros. Também podem ser passados objetos como parâmetros, desde que os mesmos possam ser avaliados como strings.

Método	Descrição
Replace()	Substitui todas as ocorrências de um determinado caractere por outro caractere.
ToLower()	Converte a string para letras minúsculas.
ToUpper()	Converte a string para letras maiúsculas.
Trim()	Remove os espaços em branco de uma string.

Uma variável do tipo string também tem duas propriedades interessantes. A primeira é a propriedade Length, a qual retorna o tamanho da strings, ou seja, o número de caracteres da string.

Para exemplificar o uso da propriedade Length considere o exemplo da Listagem 3.5. Neste exemplo o usuário deve digitar uma string de texto e o programa retorna o número de caracteres digitados. É importante salientar que espaços em branco também contam.

Listagem 3.5 – Um exemplo utilizando a propriedade Length – ex5cap3.cs

```
using System;

class ex5cap3
{
    // Exemplo5 - Capítulo 3.
    // Utilização da propriedade Length.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {

        // Entrada da string pelo usuário.

        Console.WriteLine("Digite algum texto e pressione <ENTER> ->");
        String Texto=Console.ReadLine();

        // Exibição dos resultados.

        Console.WriteLine("++++++");
        Console.WriteLine("Você digitou o seguinte texto -> {0}",Texto);
        Console.WriteLine("O mesmo tem -> {0} caracteres.",Texto.Length);
        Console.WriteLine("++++++");
    }
}
```

Digite o exemplo da Listagem 3.5 e salve o mesmo em um arquivo chamado ex5cap3.cs, na pasta C:\ExCsharp\cap3. Compile e execute o exemplo da Listagem 3.5. Quando o programa solicitar digite o texto: APRENENDO C# e pressione ENTER. Você obterá os resultados indicados na Figura 3.5.

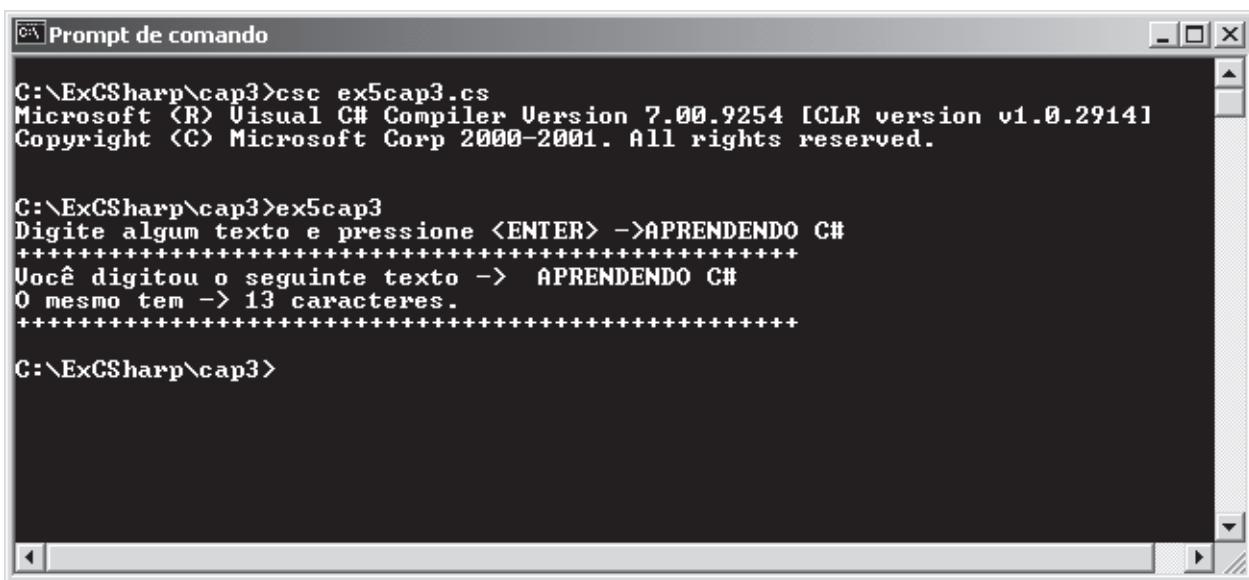


Figura 3.5: Executando o programa ex5cap3.exe.

Também podemos acessar caracteres individuais de uma string, com base na posição dos mesmos. O primeiro caractere é a posição zero; o segundo, a posição 1 e assim por diante. Por exemplo, considere a string Teste definida a seguir:

```
string Teste = "APRENENDO C#"
```

Agora considere a declaração e inicialização das seguintes variáveis do tipo char:

```
char primcar = teste[0];
char segcar = teste[1];
char deccar = teste[9];
```

A variável primcar conterá A.

A variável segcar conterá P.

A variável deccar conterá O.

O Tipo Arrays

Um array é uma variável que contém um conjunto de elementos sendo todos do mesmo tipo. Na época do Fortran ou do Pascal para DOS o array era chamado de vetor, quando fosse um array de uma única dimensão, e matriz, para um array de duas dimensões. Temos dois fatos fundamentais a respeito de arrays:

- ◆ Todos os elementos de um array devem ser do mesmo tipo.
- ◆ Acessamos os elementos de um array através de seu(s) índice(s).

Por exemplo, para criarmos um array de uma única dimensão (vetor), com 5 elementos do tipo Int32, utilizamos o seguinte comando:

```
int[] meuarray = new int[5];
```

Podemos definir os valores deste array, da seguinte maneira:

```
meuarray[0] = 25;
meuarray[1] = 15;
meuarray[2] = 20;
meuarray[3] = 12;
meuarray[4] = 44;
```

Observe que o primeiro elemento do array é o elemento de índice zero.

Também podemos criar arrays de mais dimensões. Observe o exemplo a seguir, onde estamos criando um array de duas dimensões (uma matriz), com 2 linhas e duas colunas:

```
int[,] minhamatriz = new int[2,2];
```

Podemos definir os valores deste array, da seguinte maneira:

```
meuarray[0,0] = 25;
meuarray[0,1] = 15;
meuarray[1,0] = 28;
meuarray[1,1] = 12;
```

Vamos a um exemplo simples. Onde criamos um array de duas dimensões, com 3 linhas e 3 colunas, atribuímos valores para os elementos do array, e depois exibimos os valores no vídeo.

Considere o exemplo da Listagem 3.6.

Listagem 3.6 – Um exemplo utilizando um array de duas dimensões – ex6cap3.cs

```
using System;

class ex6cap3
{
    // Exemplo6 - Capítulo 3.
    // Um array de duas dimensões.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
```

```
{  
  
    // Declaração do array de duas dimensões.  
  
    int[,] minhamatriz = new int[3,3];  
  
    // Definição dos valores do array.  
  
    minhamatriz[0,0] = 0;  
    minhamatriz[0,1] = 5;  
    minhamatriz[0,2] = 10;  
    minhamatriz[1,0] = 15;  
    minhamatriz[1,1] = 20;  
    minhamatriz[1,2] = 15;  
    minhamatriz[2,0] = 10;  
    minhamatriz[2,1] = 5;  
    minhamatriz[2,2] = 0;  
  
    // Exibição dos resultados.  
  
    Console.WriteLine("O valor da primeira linha primeira coluna é -> {0}",minhamatriz[0,0]);  
    Console.WriteLine("O valor da primeira linha segunda coluna é -> {0}",minhamatriz[0,1]);  
    Console.WriteLine("O valor da primeira linha terceira coluna é -> {0}",minhamatriz[0,2]);  
    Console.WriteLine("O valor da segunda linha primeira coluna é -> {0}",minhamatriz[1,0]);  
    Console.WriteLine("O valor da segunda linha segunda coluna é -> {0}",minhamatriz[1,1]);  
    Console.WriteLine("O valor da segunda linha terceira coluna é -> {0}",minhamatriz[1,2]);  
    Console.WriteLine("O valor da terceira linha primeira coluna é -> {0}",minhamatriz[2,0]);  
    Console.WriteLine("O valor da terceira linha segunda coluna é -> {0}",minhamatriz[2,1]);  
    Console.WriteLine("O valor da terceira linha terceira coluna é -> {0}",minhamatriz[2,2]);  
  
}  
}
```

Digite o exemplo da Listagem 3.6 e salve o mesmo em um arquivo chamado ex6cap3.cs, na pasta C:\ExCsharp\cap3. Compile e execute o exemplo da Listagem 3.6. Você obterá os resultados indicados na Figura 3.6.

```
C:\ExCSharp\cap3>csc ex6cap3.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap3>ex6cap3
0 valor da primeira linha primeira coluna é -> 0
0 valor da primeira linha segunda coluna é -> 5
0 valor da primeira linha terceira coluna é -> 10
0 valor da segunda linha primeira coluna é -> 15
0 valor da segunda linha segunda coluna é -> 20
0 valor da segunda linha terceira coluna é -> 15
0 valor da terceira linha primeira coluna é -> 10
0 valor da terceira linha segunda coluna é -> 5
0 valor da terceira linha terceira coluna é -> 0

C:\ExCSharp\cap3>
```

Figura 3.6: Executando o programa ex6cap3.exe.

NOTA: Apresentaremos mais alguns exemplos com arrays mais adiante neste capítulo, onde falaremos sobre as estruturas de controle do C#. Veremos alguns exemplos ao apresentarmos a estrutura de controle For, a qual é tipicamente utilizada para percorrer todos os elementos de um array.

Os valores para a(s) dimensão(ões) de um array podem ser fornecidos pelo usuário, para que o mesmo seja criado com tamanhos variáveis, cada vez que o programa é executado. Por exemplo, vamos supor que o usuário tenha fornecido o valor 5 para a dimensão de um array e que este valor esteja em uma variável chamada dimenarray. Podemos criar, por exemplo, um array de inteiros com a dimensão definida pela variável dimenarray, utilizando o seguinte comando:

```
int[] arraydinamico = new int[dimenarray];
```

Instruções de Fluxo de Controle no C#

Toda linguagem disponibiliza uma série de instruções para controlar o fluxo de execução do programa. São instruções para executar um ou outro conjunto de comando dependendo de uma condição ser verdadeira ou falsa; são instruções para executar um conjunto de comandos um número determinado de vezes e instruções para executar um conjunto de comandos até que uma condição se torne verdadeira ou falsa.

No C#, as instruções de fluxo de controle são divididas em três categorias:

- ◆ Instruções de seleção.
- ◆ Instruções de repetição.
- ◆ Instruções de salto (jump).

A maioria das instruções tem o seu funcionamento baseado em um teste lógico, o qual retorna True ou False.

Instruções de Seleção

As instruções de seleção executam um entre vários comandos disponíveis. O comando a ser executado é selecionado com base no valor de uma expressão.

A Instrução If

Este comando seleciona um comando ou grupos de comando para execução, com base no valor retornado por uma expressão booleana. Uma expressão booleana somente pode retornar dois valores: True ou False.

A forma mais simples deste comando é indicada a seguir:

```
if expressão
{
    comando1;
    comando2;
    ...
    comandon;
}
```

Comando1, Comando2 até Comandon somente serão executados se a expressão for verdadeira. Observe que podemos executar mais do que um comando, caso a expressão seja verdadeira. Para isto basta colocar os diversos comandos entre chaves. Considere o seguinte exemplo:

```
if (x>y)
{
    Console.WriteLine("x é maior do que y")
    x = x+1
    y = y-1
}
```

Os comandos entre as chaves somente serão executados quando x for maior do que y; caso contrário, a execução “pula” para o primeiro comando após o fechamento das chaves.

Uma outra forma da instrução if é a que inclui a cláusula else. Com a cláusula else podemos definir um conjunto de comandos que devem ser executados se a expressão de teste retornar False. A sintaxe para esta instrução é a seguinte:

```
if (x<y)
{
    comando1_verdadeiro;
    comando2_verdadeiro;
    ...
    comandon_verdadeiro;
```

```

}

else
{
    comando1_falso;
    comando2_falso;
    ...
    comandon_falso;
}

```

Vamos a um exemplo simples, onde o usuário entra com dois valores. O programa devolve diferentes mensagens para o caso de o primeiro ser maior do que o segundo ou para o caso do segundo ser maior ou igual ao primeiro.

Considere o exemplo da Listagem 3.7.

Listagem 3.7 – Um exemplo utilizando a instrução if – ex7cap3.cs

```

using System;

class ex7cap3
{
    // Exemplo7 - Capítulo 3.
    // Utilizando a instrução If.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {

        // Declaração das variáveis.

        int valor1;
        int valor2;

        // Entrada dos valores de x e y

        Console.Write("Digite o primeiro valor inteiro ->");
        String Aux1=Console.ReadLine();
        Console.Write("Digite o segundo valor inteiro ->");

    }
}

```

```
String Aux2=Console.ReadLine();

// Converto os valores para inteiro.

valor1 = Convert.ToInt32(Aux1);
valor2 = Convert.ToInt32(Aux2);

// Teste e exibição dos resultados.

if (valor1>valor2)
{
    Console.WriteLine("*****");
    Console.WriteLine("PRIMEIRO VALOR É MAIOR DO QUE O SEGUNDO !");
    Console.WriteLine("*****");
    Console.WriteLine("O PRIMEIRO VALOR DIGITADO FOI: {0}",valor1);
    Console.WriteLine("O SEGUNDO VALOR DIGITADO FOI: {0}",valor2);
    Console.WriteLine("*****");
}

else
{
    Console.WriteLine("*****");
    Console.WriteLine("PRIMEIRO VALOR É MENOR OU IGUAL AO SEGUNDO !");
    Console.WriteLine("*****");
    Console.WriteLine("O PRIMEIRO VALOR DIGITADO FOI: {0}",valor1);
    Console.WriteLine("O SEGUNDO VALOR DIGITADO FOI: {0}",valor2);
    Console.WriteLine("*****");
}

}
```

Digite o exemplo da Listagem 3.7 e salve o mesmo em um arquivo chamado ex7cap3.cs, na pasta C:\ExCsharp\cap3. Compile e execute o exemplo da Listagem 3.7. Para o primeiro valor digite 10 e para o segundo digite 5. Você obterá os resultados indicados na Figura 3.7.

Reita a execução do programa. Agora digite 10 para o primeiro valor e 15 para o segundo. Você obterá os resultados indicados na Figura 3.8.

```
C:\ExCSharp\cap3>ex7cap3
Digite o primeiro valor inteiro ->10
Digite o segundo valor inteiro ->15
*****
PRIMEIRO VALOR É MENOR OU IGUAL AO SEGUNDO !
*****
O PRIMEIRO VALOR DIGITADO FOI: 10
O SEGUNDO VALOR DIGITADO FOI: 15
*****
```

Figura 3.7: Executando o programa ex7cap3.exe.

```
C:\ExCSharp\cap3>ex7cap3
Digite o primeiro valor inteiro ->10
Digite o segundo valor inteiro ->15
*****
PRIMEIRO VALOR É MENOR OU IGUAL AO SEGUNDO !
*****
O PRIMEIRO VALOR DIGITADO FOI: 10
O SEGUNDO VALOR DIGITADO FOI: 15
*****
```

Figura 3.8: Executando o programa ex7cap3.exe com novos valores.

Também podemos utilizar mais do que um else na mesma instrução if. Neste caso, após a cláusula else, iniciamos um novo if. A melhor maneira de entendermos esta estrutura é através de um exemplo. Vamos supor que você tenha que executar diferentes comandos dependendo do valor de uma variável x ser igual a 0, 1, 2, ou 3. Para isso, poderíamos utilizar o seguinte bloco de código:

```

if (x == 0)
{
    comandos para x = 0
}

else if (x == 1)
{
    comandos para x = 1
}

else if (x==2)
{
    comandos para x = 2
}

else
{
    comandos para x = 3
}

```

Observe que a última hipótese ($x == 3$) não precisamos testar, pois, se a execução chegou até o último else, é porque x não é igual a 0, nem igual a 1 e nem igual a 2. Quando um dos testes for verdadeiro, os comandos associados são executados e a execução “pula” para o primeiro comando após o bloco if.

Na Tabela 3.3, temos uma descrição dos principais operadores de comparação.

Tabela 3.3 Os principais operadores de comparação.

Operador	Descrição
<code>==</code>	Igual. Retorna verdadeiro se os valores comparados forem iguais.
<code>!=</code>	Diferente. Retorna verdadeiro se os valores comparados forem diferentes.
<code>></code>	Maior do que. Retorna verdadeiro se o primeiro valor for maior do que o segundo – ($x > y$).
<code><</code>	Menor do que. Retorna verdadeiro se o primeiro valor for menor do que o segundo – ($x < y$).
<code>>=</code>	Maior ou igual a. Retorna verdadeiro se o primeiro valor for maior ou igual ao segundo – ($x \geq y$).
<code><=</code>	Menor ou igual a. Retorna verdadeiro se o primeiro valor for menor ou igual ao segundo – ($x \leq y$).

IMPORTANTE: Observe que o operador para comparação de igualdade não é um sinal simples de igual (`=`); ao invés disso é um sinal duplo de igualdade (`==`). O operador simples (`=`) é utilizado para atribuição de valores para variáveis. Para comparação utilize-se o operador `==`.

Porém, para situações em que temos que testar várias possibilidades, a utilização de sucessivas instruções if-else pode não ser a solução mais adequada. Para isso temos a instrução switch.

A Instrução Switch

A instrução switch tem uma expressão de controle bem no início do laço. Com base no valor da expressão de controle, diferentes comandos serão executados.

Sintaxe para a instrução switch:

```
switch (expressão_de_controle)
{
    case valor_1:
        comando1;
        comando2;
        ...
        comandon;
        comando para sair;

    case valor_2:
        comando1;
        comando2;
        ...
        comandon;
        comando para sair;

    ...
    case valor_n:
        comando1;
        comando2;
        ...
        comandon;
        comando para sair;

    default:
        comando1;
        comando2;
        ...
        comandon;
        comando para sair;
}
```

Para a expressão de controle podemos utilizar os seguintes tipos: sbyte, byte, short, ushort, long, ulong, char, string ou um tipo de enumeração.

A expressão_de_controle é avaliada. Se um dos valores (valor_1, valor_2, etc.) for coincidente com o valor da expressão_de_controle, os comandos respectivos serão executados. Se não tivermos nenhuma coincidência, os comandos do rótulo default: serão executados. Se não houver um rótulo default, e não houver coincidência com nenhum valor, a execução segue para o primeiro comando após o final do bloco switch.

“comando para sair” é um comando que faz com que a execução saia do bloco switch, de tal maneira que os demais valores não precisem ser avaliados, uma vez que uma coincidência já foi encontrada ou os comandos relacionados ao rótulo default: foram executados. O comando mais comumente utilizado é o break.

Vamos a um exemplo simples, no qual o usuário deve digitar um valor entre 1 e 7. O programa informa o dia da semana correspondente, de acordo com o seguinte critério:

- ◆ Domingo = 1
- ◆ Segunda-feira = 2
- ◆ Terça-feira = 3
- ◆ Quarta-feira = 4
- ◆ Quinta-feira = 5
- ◆ Sexta-feira = 6
- ◆ Sábado = 7

Considere o exemplo da Listagem 3.8.

Listagem 3.8 – Um exemplo utilizando a instrução switch – ex8cap3.cs

```
using System;

class ex8cap3
{
    // Exemplo 8 - Capítulo 3.
    // Utilizando a instrução switch.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {
```

```
// Declaração da variável do tipo inteiro.

int dia;

// Entrada do valor pelo usuário.

Console.WriteLine("Digite um número inteiro entre 1 e 7 ->");

String Aux1=Console.ReadLine();

// Converto o valor para inteiro.

dia = Convert.ToInt32(Aux1);

// Utilizo switch para testar o valor da variável dia.

switch (dia)

{

    case 1:

        Console.WriteLine("*****");
        Console.WriteLine("VOCÊ ESCOLHEU O DOMINGO !!!");
        Console.WriteLine("*****");
        break;

    case 2:

        Console.WriteLine("*****");
        Console.WriteLine("VOCÊ ESCOLHEU A SEGUNDA-FEIRA !!!");
        Console.WriteLine("*****");
        break;

    case 3:

        Console.WriteLine("*****");
        Console.WriteLine("VOCÊ ESCOLHEU A TERÇA-FEIRA !!!");
        Console.WriteLine("*****");
        break;

    case 4:
```

```
Console.WriteLine("*****");
Console.WriteLine("VOCÊ ESCOLHEU A QUARTA-FEIRA !!!");
Console.WriteLine("*****");
break;

case 5:
Console.WriteLine("*****");
Console.WriteLine("VOCÊ ESCOLHEU A QUINTA-FEIRA !!!");
Console.WriteLine("*****");
break;

case 6:
Console.WriteLine("*****");
Console.WriteLine("VOCÊ ESCOLHEU SEXTA-FEIRA !!!");
Console.WriteLine("*****");
break;

case 7:
Console.WriteLine("*****");
Console.WriteLine("VOCÊ ESCOLHEU O SÁBADO !!!");
Console.WriteLine("*****");
break;

default:
Console.WriteLine("*****");
Console.WriteLine("VOCÊ NÃO DIGITOU UM VALOR ENTRE 1 E 7 !!!");
Console.WriteLine("*****");
break;
} //Esta chave fecha o laço switch.

} //Esta chave fecha o método Main()

} Esta chave fecha a classe ex8cap3.
```

Digite o exemplo da Listagem 3.8 e salve o mesmo em um arquivo chamado ex8cap3.cs, na pasta C:\ExCsharp\cap3. Compile e execute o exemplo da Listagem 3.8. Digite o valor 5. Você obterá os resultados indicados na Figura 3.9.

```
C:\ExCSharp\cap3>csc ex8cap3.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap3>ex8cap3
Digite um número inteiro entre 1 e 7 ->5
*****
VOCÊ ESCOLHEU A QUINTA-FEIRA !!
*****
```

Figura 3.9: Executando o programa ex8cap3.exe.

Repita a execução do programa. Agora digite 10. Você obterá os resultados indicados na Figura 3.10.

```
C:\ExCSharp\cap3>csc ex8cap3.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap3>ex8cap3
Digite um número inteiro entre 1 e 7 ->5
*****
VOCÊ ESCOLHEU A QUINTA-FEIRA !!
*****
```



```
C:\ExCSharp\cap3>ex8cap3
Digite um número inteiro entre 1 e 7 ->10
*****
VOCÊ NÃO DIGITOU UM VALOR ENTRE 1 E 7 !!
*****
```

Figura 3.10: Executando o programa ex8cap3.exe com um valor igual a 10.

Instruções de Repetição

As instruções de repetição, como o próprio nome sugere, permitem que um conjunto de comandos seja executado um número definido de vezes ou até que uma determinada condição seja verdadeira. Vamos estudar as seguintes instruções:

- ◆ for
- ◆ while
- ◆ do/while
- ◆ foreach

A Instrução For

Esta instrução é utilizada para repetir um ou mais conjuntos, um número determinado de vezes. A instrução for do C# é praticamente igual ao comando for do C/C++. É composta de três partes, conforme indicado na sintaxe a seguir:

```
for (inicialização; expressão boleana; incremento)
{
    comando1
    comando2
    ...
    comandon
}
```

O número de vezes que os comandos da instrução for serão executados é determinado por uma variável de controle. Na primeira parte do laço for, definimos um valor para esta variável. Na segunda parte fazemos um teste para ver se a variável já atingiu um valor limite. A terceira e última parte informa o incremento da variável a cada passagem do laço.

Enquanto a expressão boleana for verdadeira, os comandos do laço continuam a ser executados. Quando a expressão torna-se falsa, a execução “pula” para o primeiro comando após a chave de fechamento do laço for.

Vamos a um exemplo simples, onde são exibidos os números entre 1 e 10 e o valor do número elevado ao quadrado e ao cubo.

Considere o exemplo da Listagem 3.9

Listagem 3.9 – Um exemplo utilizando a instrução For – ex9cap3.cs

```
using System;

class ex9cap3
{
    // Exemplo9 - Capítulo 3.
    // A instrução for.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {
        // Declaração da variável.
```

```

int i;

// Laço for que é executado dez vezes.

Console.WriteLine("Número"\t"Quadrado"\t"Cubo");

for (i=1;i<=10;i++)

{
    Console.Write(i"\t");
    Console.Write("{0}" "\t",i*i);
    Console.WriteLine("{0}",i*i*i);
}

}

```

Digite o exemplo da Listagem 3.9 e salve o mesmo em um arquivo chamado ex9cap3.cs, na pasta C:\ExCsharp\cap3. Compile e execute o exemplo da Listagem 3.9. Você obterá os resultados indicados na Figura 3.11.

Número	Quadrado	Cubo
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

Figura 3.11: Executando o programa ex9cap3.exe.

Observe que para o laço for utilizamos uma variável inteira i. Inicializamos a mesma com o valor 1. O teste $1 \leq 10$ faz com que o laço seja executado dez vezes. A última parte: i++ é novidade. O operador ++ incrementa a variável i de um em um. O resultado é o mesmo que se tivéssemos utilizado: i=i+1. No próximo capítulo faremos um estudo detalhado dos operadores do C#.

Para calcular o quadrado multiplicamos o número por ele mesmo: i*i.

Para calcular o cubo multiplicamos o número por ele mesmo, três vezes: i*i*i.

Na Tabela 3.4 temos uma descrição dos principais operadores aritméticos.

Tabela 3.4 Os principais operadores aritméticos.

Operador	Descrição
+	Adição.
-	Subtração.
*	Multiplicação.
/	Divisão.

Observe que também utilizamos o código de controle \t – tabulação. Utilizamos a tabulação para alinhar os resultados.

Também podemos utilizar instruções for “aninhadas”, isto é, instruções for uma dentro da outra. O uso típico para instruções for aninhadas é para percorrer os valores de uma matriz, onde a instrução “for” mais externa vai variando o número das linhas e a instrução “for” mais interna vai variando o número das colunas. Observe o exemplo a seguir:

```
for (i=1;i<=6;I++)
{
    for (j=1;j<=5;j++)
    {
        Comando1
        Comando2
        ...
        Comandon
    }
}
```

Neste caso primeiro i=1. O programa entra no laço interno e para i=1 o laço interno é executado 6 vezes, ou seja: i=1, j=1 e i=1, j=2 e i=1, j=3 e i=1, j=4 e i=1, j=5 . Agora i é incrementado e torna-se igual a 2. Para i=2 o j varia de 1 a 5 e a história se repete. Até que i tenha variado até 6 e j até 5.

Vamos comprovar estas variações através de um exemplo prático. Neste exemplo utilizaremos dois laços. No laço externo i varia de 1 até 6. No laço interno i varia de 1 até 5. Dentro do laço colocamos comandos que vão exibindo os valores de i e j respectivamente.

Considere o exemplo da Listagem 3.10

Listagem 3.10 – Exemplo utilizando instruções For “aninhadas” – ex10cap3.cs

```
using System;

class ex10cap3
{
    // Exemplo 10 - Capítulo 3.
    // Instruções for aninhadas.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {

        // Declaração das variáveis.

        int i;
        int j;

        // Laço que é executado com i de 1 até 6
        // e com j variando de 1 até 5.

        Console.WriteLine("i   j"+'\t'+ "i   j"+'\t'+ "i   j"+'\t'+ "i   j"+'\t'+ "i   j");

        for (i=1;i<=6;i++)
        {
            for (j=1;j<=5;j++)
            {
                Console.Write(i);
                Console.Write("  {0}",j);
                Console.Write("\t");
            }
            Console.WriteLine();
        }
    }
}
```

Digite o exemplo da Listagem 3.10 e salve o mesmo em um arquivo chamado ex10cap3.cs, na pasta C:\ExCsharp\cap3. Compile e execute o exemplo da Listagem 3.10. Você obterá os resultados indicados na Figura 3.12.

```
C:\ExCSharp\cap3>csc ex10cap3.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap3>ex10cap3
i   j   i   j   i   j   i   j
1   1   1   2   1   3   1   4   1   5
2   1   2   2   2   3   2   4   2   5
3   1   3   2   3   3   3   4   3   5
4   1   4   2   4   3   4   4   4   5
5   1   5   2   5   3   5   4   5   5
6   1   6   2   6   3   6   4   6   5

C:\ExCSharp\cap3>
```

Figura 3.12: Executando o programa ex10cap3.exe.

Após encerrada a execução do laço for, a execução segue normalmente para o comando seguinte ao encerramento do laço for. Também podemos utilizar a instrução break, dentro da laço for, para encerrá-lo e desviar a execução para o primeiro comando após o encerramento do laço. Normalmente a instrução break é colocada dentro de uma instrução if, de tal maneira que a mesma somente será executada se uma determinada condição for alcançada.

A Instrução While

A instrução while executa um ou mais comandos enquanto uma determinada condição for verdadeira. A condição é testada no início do laço. Se a condição for verdadeira o(s) comando(s) do laço são executados. Ao chegar no final do laço a execução volta para o teste no início do laço while. Se a condição continuar verdadeira o(s) comando(s) são executados novamente e o processo se repete, até que a condição no início do laço se torne falsa. Quando a condição se torna falsa, a execução “pula” para o primeiro comando após o final do laço while. A seguir temos a sintaxe para o comando while:

```
while (condição)
{
    comando1
    comando2
    ...
    comandón
}
```

IMPORTANTE: Os comandos de dentro do laço devem ser capazes de tornar a condição falsa em um determinado momento, pois caso contrário o laço continuará a ser executado infinitamente. Neste caso teremos criado um laço infinito, no qual o programa ficará executando, normalmente, até congelar.

Uma característica importante da instrução while é que, se no início do laço, a condição já for falsa, os comandos do interior do laço não serão executados nem uma única vez.

Vamos a um exemplo de utilização da instrução While. No exemplo da Listagem 3.11 utilizamos a função While para determinar a soma dos n primeiros números inteiros. O valor de n é informado pelo usuário.

Considere o exemplo da Listagem 3.11.

Listagem 3.11 – Um exemplo utilizando a instrução While – ex11cap3.cs

```
using System;

class ex11cap3
{
    // Exemplo 11 - Capítulo 3.
    // Instrução while.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {
        // Declaração das variáveis.

        int n;
        int i;
        int soma;

        // Entrada do valor de n pelo usuário.

        Console.Write("Digite um valor inteiro entre 1 e 100 ->");
        String Aux1 = Console.ReadLine();
        n = Convert.ToInt32(Aux1);

        // Utilizamos o laço While para determinar a soma dos n
        // primeiros números inteiros.
```

```

    soma = 0;
    i=1;

    while (i<=n)
    {
        soma=soma+i;
        i=i+1; //IMPORTANTÍSSIMO
    }

    //Exibição dos resultados.
    Console.WriteLine("*****");
    Console.WriteLine("Soma dos números inteiros de 1 a {0}",n);
    Console.WriteLine("*****");
    Console.WriteLine("SOMA -> {0}",soma);
    Console.WriteLine("*****");
}
}

```

Note a observação: //IMPORTANTÍSSIMO. Se não colocássemos essa linha – i=i+1, a variável i não seria incrementada a cada passo do laço e, portanto, o teste (i<=n) nunca se tornaria falso, o que faria com que o laço ficasse executando infinitamente.

Digite o exemplo da Listagem 3.11 e salve o mesmo em um arquivo chamado ex11cap3.cs, na pasta C:\ExCsharp\cap3. Compile e execute o exemplo da Listagem 3.11. Digite o valor 50. Você obterá os resultados indicados na Figura 3.13.

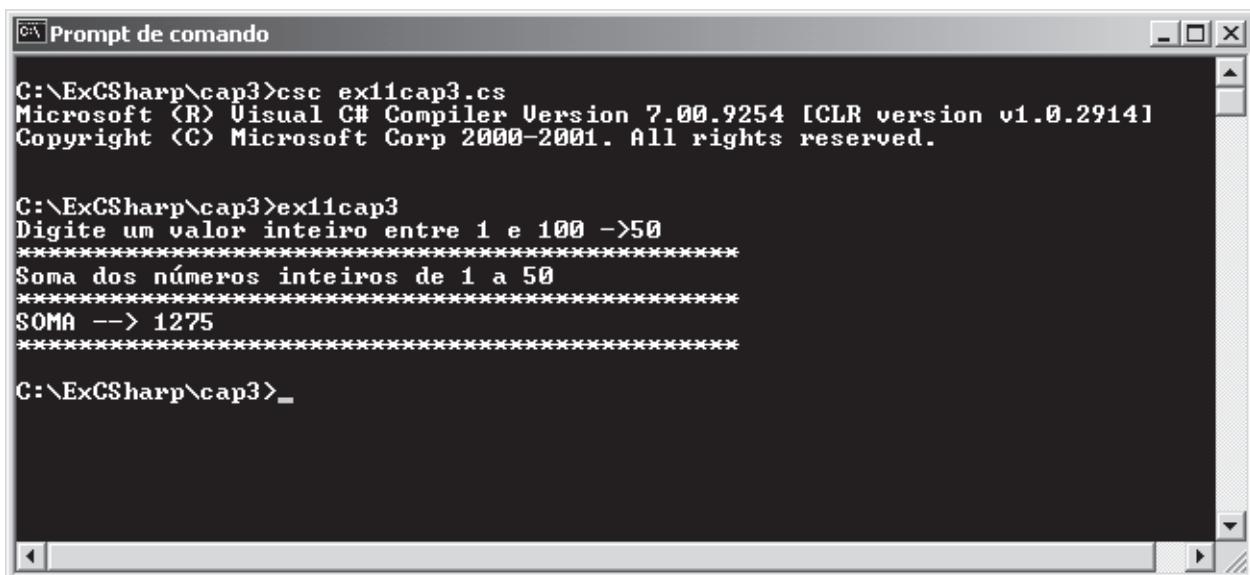


Figura 3.13: Executando o programa ex11cap3.exe com um valor 50.

Reita a execução do programa. Agora digite 85. Você obterá os resultados indicados na Figura 3.14.

```
C:\ExCSharp\cap3>csc ex11cap3.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap3>ex11cap3
Digite um valor inteiro entre 1 e 100 ->50
*****
Soma dos números inteiros de 1 a 50
*****
SOMA --> 1275
*****

C:\ExCSharp\cap3>ex11cap3
Digite um valor inteiro entre 1 e 100 ->80
*****
Soma dos números inteiros de 1 a 80
*****
SOMA --> 3240
*****
```

Figura 3.14: Executando o programa ex11cap3.exe com um valor igual a 80.

A Instrução do/while

Com a instrução do/while deslocamos o teste para o final do laço. Com isso estamos garantindo que os comandos no interior do laço serão executados pelo menos uma vez, já que a condição somente será avaliada no final do laço, após os comandos terem sido executados.

A sintaxe para a instrução do/while é a seguinte:

```
do
{
    comando1
    comando2
    ...
    comandoN
}
while (teste)
```

IMPORTANTE: Os comandos de dentro do laço devem ser capazes de tornar a condição falsa em um determinado momento, pois caso contrário o laço continuará a ser executado infinitamente. Neste caso teremos criado um laço infinito, no qual o programa ficará executando, normalmente, até congelar.

Uma característica importante da instrução while é que os comandos do laço são executados, pelo menos uma vez.

Vamos a um exemplo de utilização da instrução While. No exemplo da Listagem 3.12 utilizamos a função do/while para fazer com que o valor digitado pelo usuário esteja dentro de uma determinada faixa. No nosso caso, queremos que o usuário digite um valor menor do que 10. Se for digitado um valor maior ou igual a 10, o programa apresenta novamente uma mensagem para que seja digitado um valor menor do que 10.

Considere o exemplo da Listagem 3.12.

Listagem 3.12 – Exemplo utilizando a instrução Do/While – ex12cap3.cs

```
using System;

class ex12cap3
{
    // Exemplo 12 - Capítulo 3.
    // Instrução do/while.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {
        // Declaração da variável.

        int i;

        // Enquanto o usuário não digitar um valor menor
        // do que 10 o programa não vai adiante.

        do
    }
```

```

{
    Console.WriteLine("Digite um valor menor do que 10 ->");
    String Aux1 = Console.ReadLine();
    i = Convert.ToInt32(Aux1);
}

while (i>=10);

Console.WriteLine("*****");
Console.WriteLine("AGORA SIM VOCÊ DIGITOU UM VALOR MENOR DO QUE 10");
Console.WriteLine("*****");
Console.WriteLine("VALOR DIGITADO --> {0}",i);
Console.WriteLine("*****");

}
}

```

Digite o exemplo da Listagem 3.12 e salve o mesmo em um arquivo chamado ex12cap3.cs, na pasta C:\ExCsharp\cap3. Compile e execute o exemplo da Listagem 3.12. Digite o valor 25 e pressione ENTER. Observe que o programa solicita novamente que seja digitado um valor menor do que 10. Digite 50. Mesma coisa. Agora digite 8. Você obterá os resultados indicados na Figura 3.15.

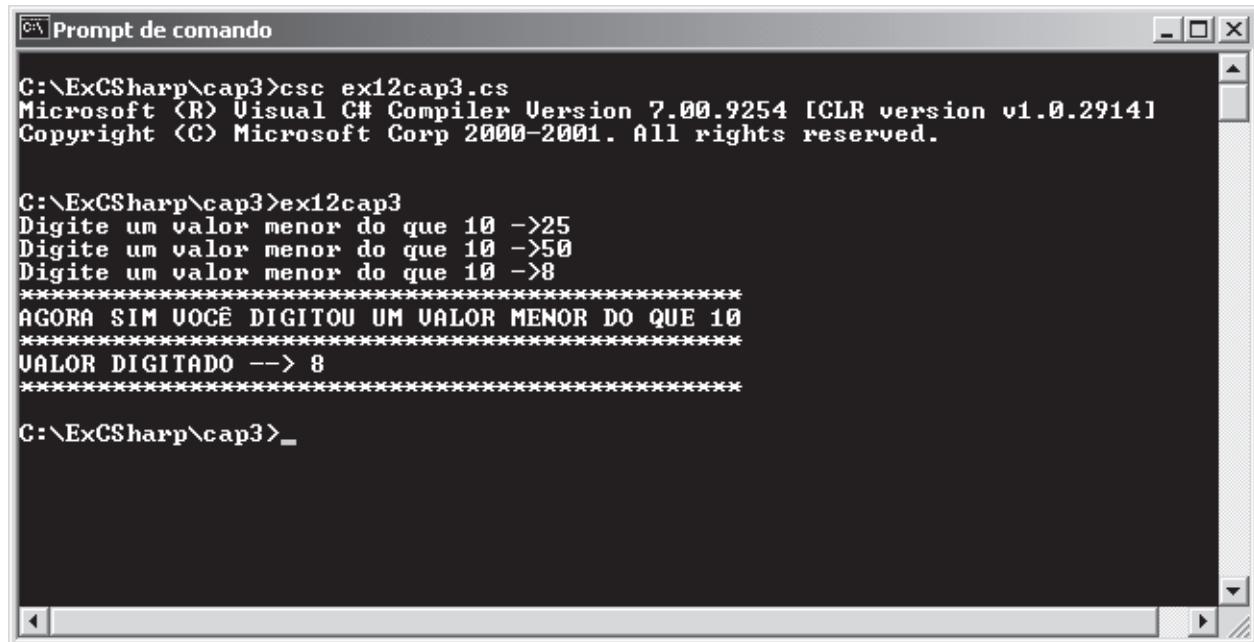


Figura 3.15: Executando o programa ex12cap3.exe.

A Instrução Foreach

Esta instrução já é uma velha conhecida da turma do Visual Basic. A instrução foreach é utilizada para percorrer todos os elementos de uma determinada coleção. Também podemos utilizar esta instrução para percorrer todos os elementos de um array de elementos, uma vez que um array não deixa de ser uma coleção de elementos do mesmo tipo.

A seguir um exemplo no qual utilizamos a instrução foreach para exibir os elementos de um array de strings.

Considere o exemplo da Listagem 3.13.

Listagem 3.13 – Exemplo utilizando a instrução ForEach – ex13cap3.cs

```
using System;

class ex13cap3
{
    // Exemplo 13 - Capítulo 3.
    // Instrução foreach.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {

        // Declaração e preenchimento do array.
        string[] nomes = {"jósé", "maria", "pedro", "antônio", "carlo"};
        // Utilizo foreach para percorrer todos os elementos do array.

        foreach (string nome in nomes)
            Console.WriteLine("Nome: {0}", nome);

    }
}
```

Digite o exemplo da Listagem 3.13 e salve o mesmo em um arquivo chamado ex13cap3.cs, na pasta C:\ExCsharp\cap3. Compile e execute o exemplo da Listagem 3.13. Você obterá os resultados indicados na Figura 3.16.

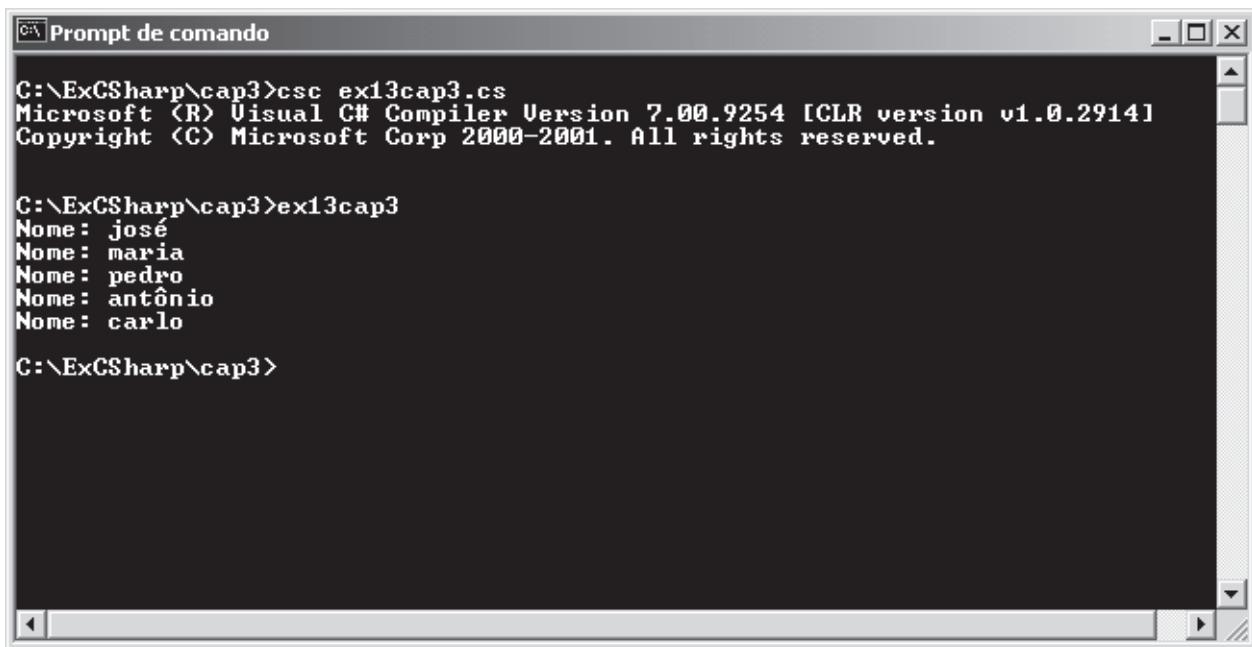


Figura 3.16: Executando o programa ex13cap3.exe.

Observe os seguintes comandos:

```
foreach (string nome in nomes)
    Console.WriteLine("Nome: {0}", nome);
```

O primeiro comando percorre os elementos do array nomes. O segundo comando exibe o valor de cada elemento.

Vamos a mais um exemplo de utilização de foreach para percorrer os elementos de uma coleção. Dentro do namespace System, temos a classe System.Environment. Dentro desta classe temos um método chamado GetLogicalDrives. Este método retorna um array de strings contendo o nome dos drives lógicos do computador. Depois utilizaremos foreach para percorrer os elementos do array de strings retornado por GetLogicalDrives.

Considere o exemplo da Listagem 3.14.

Listagem 3.14 – Outro exemplo utilizando a instrução ForEach – ex14cap3.cs

```
using System;

class ex14cap3
{
    // Exemplo 14 - Capítulo 3.
    // Instrução foreach.
    // Método Environment.GetLogicalDrives()
    // Por: Júlio Battisti
```

```
// MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

public static void Main()
{
    // Declaração e preenchimento do array.

    string[] drives = Environment.GetLogicalDrives();

    // Utilizo foreach para percorrer todos os elementos do array.

    foreach (string drive in drives)
        Console.WriteLine("Nome: {0}", drive);

}
}
```

Digite o exemplo da Listagem 3.14 e salve o mesmo em um arquivo chamado ex14cap3.cs, na pasta C:\ExCsharp\cap3. Compile e execute o exemplo da Listagem 3.14. Você obterá os resultados indicados na Figura 3.17.

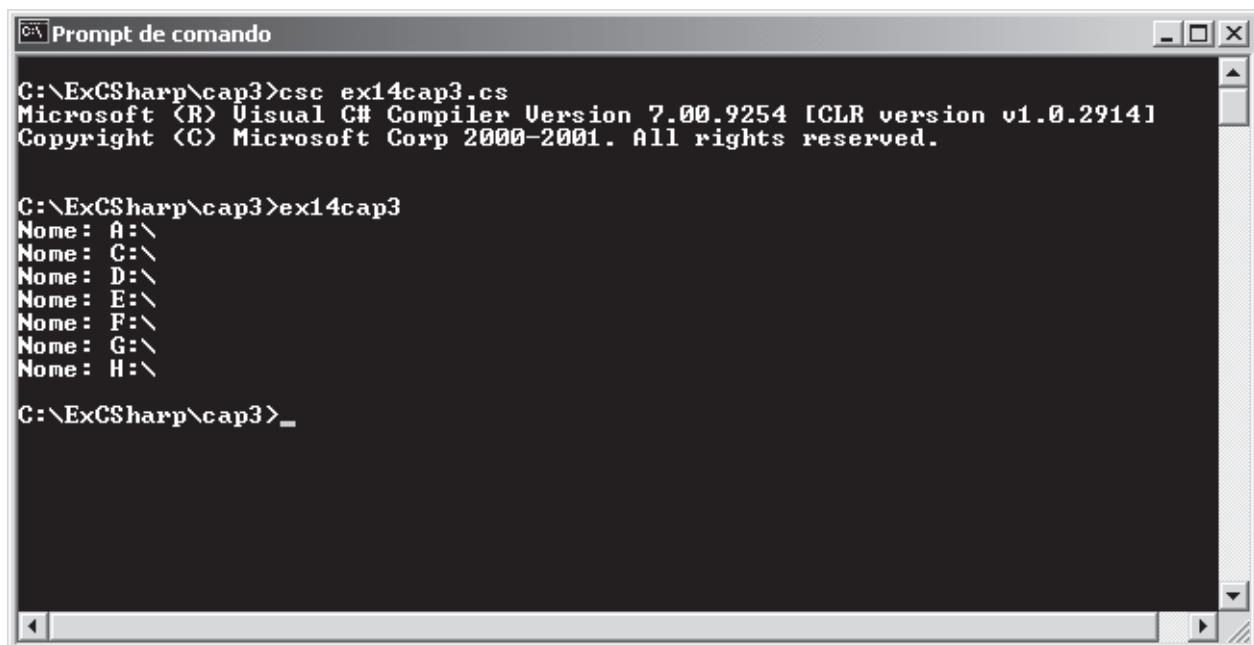


Figura 3.17: Executando o programa ex14cap3.exe.

NOTA: Os resultados obtidos podem ser diferentes dos exibidos na Figura 3.17, uma vez que o mesmo depende dos drives configurados no computador que você estiver utilizando.

De diferente neste exemplo, em relação ao anterior, apenas o fato de termos utilizado uma classe do .NET Framework Class Library – Environment.GetLogicalDrives(), a qual retorna o nome (letra da unidade) de todos os drives lógicos do computador.

Instruções de Salto (Jump)

As instruções de salto transferem, incondicionalmente, a execução do programa para fora de um laço ou para um ponto específico do programa. Por exemplo, podemos utilizar uma instrução break para sair de um laço while, mesmo que a condição do laço ainda não tenha se tornado falsa.

Temos as seguintes instruções de salto:

- ◆ break
- ◆ continue
- ◆ goto (Nããããão, goto não! Tem sim. Mas não se assuste.)
- ◆ return
- ◆ throw

A Instrução Break

Sintaxe:

break;

A instrução break é utilizada para sair de um laço switch, while, do/while, for ou foreach. A execução passa para o primeiro comando após o final do laço onde o break foi executado. Se colocarmos o break fora de um dos laços citados, obteremos um erro de compilação.

A Instrução Continue

Dentro de um laço, quando é encontrada uma instrução continue, os demais comandos do laço são ignorados e o laço é executado novamente, dependendo do valor do teste. A diferença entre a instrução break e a instrução continue é que o break sai definitivamente do laço, passando a execução para o comando imediatamente após o laço; enquanto o continue apenas suspende os comandos restantes e continua a execução do laço, até que o mesmo seja encerrado.

A Instrução Goto

A utilização da instrução goto é conhecida de longa data. Podemos utilizar a instrução goto para saltar diretamente para um rótulo definido no código do programa. Um rótulo é definido da seguinte maneira:

nome_do_rótulo:

Porém esta prática é altamente desaconselhada. Quem não lembra dos famosos programas “macarrão” da época do Basic. Um programa macarrão é o que utiliza muitas instruções goto, onde a execução fica pulando de uma parte para outra do

programa. O C# desencoraja o uso do goto, aplicando algumas restrições para a utilização do mesmo. Por exemplo, não podemos utilizar goto para deslocar a execução para dentro de um bloco de comandos, como por exemplo um for ou while.

A Instrução Return

Esta instrução, normalmente, é utilizada dentro de uma função. A instrução return retorna o controle do programa para o comando seguinte ao que fez a chamada à função e pode também retornar um valor calculado pela função.

A Instrução Throw

Esta instrução é utilizada no tratamento de exceções e será vista no próximo capítulo.

Conclusão

Neste capítulo tratamos dos aspectos básicos da linguagem C#. Como estaremos utilizando o C# para a criação da parte programada das páginas ASP.NET a partir do Capítulo 6, este capítulo forma uma base importante para o restante do livro.

Iniciamos o capítulo apresentando as características gerais do C# e o porquê de mais uma linguagem de programação. Vimos que a proposta do C# é ser simples como o Visual Basic porém tão poderoso quanto o C++.

Em seguida apresentamos a estrutura básica de um programa C#. Vimos que tudo (ou quase tudo) são classes e objetos. Toda a lógica de programação no C# precisa estar contida dentro de um tipo definido, normalmente uma classe.

Em seguida tratamos dos diversos tipos de dados e estruturas existentes no C#. Vimos que existem tipos de valor – value type – e tipos de referência – reference type. Aprendemos as diferenças entre estes dois tipos. Também aprendemos a utilizar alguns métodos da classe System.Convert para fazer conversão entre tipos. Estudamos tipos mais complexos como structs, strings e arrays.

Para finalizar o capítulo apresentamos as instruções de fluxo de controle. Vimos que existem três categorias de instruções:

- ◆ Instruções de seleção.
- ◆ Instruções de repetição.
- ◆ Instruções de salto (jump).

Analisamos as diversas instruções em cada categoria.

Junto com os conceitos teóricos apresentamos exemplos simples e completos, que o leitor pode compilar e testar para ver o C# em funcionamento. O objetivo ao criarmos programas simples, de poucas linhas, é salientar o tópico em destaque. Desta maneira cada exemplo dá ênfase a um comando, instrução ou técnica específicos.

Agora que já tratamos dos aspectos básicos do C# podemos tratar de assuntos mais complexos como por exemplo o tratamento de exceções. Mas isso já é assunto para o Capítulo 4.

Introdução

Neste capítulo continuaremos o nosso estudo da linguagem C#, sempre abordando os elementos que serão utilizados para a construção de páginas ASP.NET nos demais capítulos do livro.

Iniciaremos este capítulo apresentando os operadores utilizados no C#. Também falaremos sobre a ordem de precedência dos operadores. Trataremos dos seguintes tipos de operadores:

- ◆ Built-in
- ◆ Aritméticos
- ◆ Relacionais e Lógicos
- ◆ Atribuição (Assignment)

Em seguida trataremos da implementação dos conceitos de orientação a objetos no C#. Veremos de que maneira conceitos como classes, herança e polimorfismo são implementados na linguagem C#.

Aprenderemos a criar novas classes no C#. Serão apresentados os conceitos de “construtores” e “destrutores”. Para que a classe possa fornecer alguma funcionalidade, aprenderemos a criar métodos e veremos os diferentes tipos de métodos que podem ser criados. Também aprenderemos a criar métodos que recebem parâmetros.

Vamos aprender os diferentes tipos de parâmetros existentes:

- ◆ in parameters
- ◆ ref parameters
- ◆ out parameters

Também falaremos sobre os modificadores para classes e membros de uma classe. Um modificador, na prática, define a visibilidade e o escopo da classe e de cada um dos seus membros.

Para que o leitor possa acompanhar e entender os assuntos exemplares referentes a classes, métodos, propriedades, herança e polimorfismo é necessário que o mesmo conheça a teoria básica sobre orientação a objetos. Para uma revisão dos conceitos básicos de orientação a objetos consulte o Capítulo 2, no tópico: Conceitos básicos de orientação a objetos.

Novamente estaremos utilizando exemplos simples e curtos, os quais objetivam salientar o tópico que está sendo tratado.

CAPÍTULO 4

Classes, Métodos, Herança e Polimorfismo com o C#

Operadores e Mais Operadores

Uma expressão é formada por operadores e operandos. Os operadores de uma expressão indicam qual tipo de operação deve ser aplicada entre os operandos. Os exemplos mais conhecidos de operadores são os operadores aritméticos: adição (+), subtração (-), multiplicação (*) e divisão (/).

Neste tópico vamos estudar os seguintes tipos de operadores:

- ◆ Built-in
- ◆ Aritméticos
- ◆ Relacionais e Lógicos
- ◆ Atribuição (Assignment)

NOTA: Os arquivos com o código-fonte dos exemplos deste capítulo estão disponíveis para download no site da editora Axcel Books no seguinte endereço:
www.axcel.com.br.

Quanto ao número de operandos, os operadores também podem ser classificados em:

- ◆ Operadores unários: Este tipo de operador atua sobre um único operando, como por exemplo o sinal de menos (-) para tornar um número negativo, como por exemplo: -5.
- ◆ Operadores binários: Atua sobre dois operandos. O exemplo típico são os operadores aritméticos, como por exemplo: 2+3, 5*12.
- ◆ Operador ternário: Atua sobre três operandos. Temos um único operador ternário em C#, que é o ponto-de-interrogação: ? Na verdade é um ponto-de-interrogação metido a if, conforme veremos no exemplo a seguir.

A sintaxe para o operador? é a seguinte:

```
(teste) ? valor_se_verdadeiro : valor_se_falso
```

Considere o exemplo:

```
int y;
int x;

y=22;
x = (y<25) ? 12 : 18;
```

Neste exemplo x é igual a 12. O teste ($y < 25$) é avaliado. Como o teste é verdadeiro (pois $y = 22$), é retornado o valor antes dos dois-pontos. Se o teste fosse falso seria retornado o valor após os dois-pontos. É ou não é um if disfarçado?

Built-in Operators

Existem operadores chamados Built-in para os tipos de dados int, uint, long, ulong, float, double e decimal. Isso significa que em operações aritméticas básicas entre operadores destes tipos não precisamos fazer nenhuma conversão explícita. Considere o exemplo a seguir:

```
int num1 = 23;
int num2 = 32;
int soma = num1 + num2;
```

Neste caso não precisamos fazer nenhum tipo de conversão explícita (ou cast como é chamada no C#), uma vez que existe o operador de adição interno do próprio C# (Built-in) para o tipo int.

Agora, se tentássemos fazer a seguinte operação teríamos problemas:

```
short nums1 = 10;
short nums2 = 17;
short soma = nums1+nums2;
```

Se tentarmos compilar um programa que contém este trecho de código, obteremos a mensagem de erro indicada na Figura 4.1:

```
C:\ExCSharp\cap3>csc builtin.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

Builtin.cs(17,19): error CS0029: Cannot implicitly convert type 'int' to 'short'
C:\ExCSharp\cap3>_
```

Figura 4.1: Erro de compilação.

Para que o nosso exemplo possa funcionar precisamos fazer uma conversão explícita, conforme indicado a seguir:

```
short nums1 = 10;
short nums2 = 17;
short soma = (short) (nums1+nums2);
```

Operadores Aritméticos

Temos os operadores básicos que toda linguagem tem, ou seja, os operadores para as quatro operações matemáticas, conforme descrito na Tabela 4.1.

Tabela 4.1 Os principais operadores aritméticos.

Operador	Descrição
+	Adição.
-	Subtração.
*	Multiplicação.
/	Divisão.
%	Retorna o resto da divisão.

Também temos o operador unário +, o qual simplesmente indica que um número é positivo. Não altera o valor do número. Por exemplo, os dois comandos a seguir são exatamente iguais e válidos:

```
short nums1 = +10;
ou
short nums1 = +10;
```

Já o operador unário menos (-) inverte o sinal de um número. Considere o exemplo:

```
int x = -10;
int y = -x;
Console.WriteLine(y);
```

Neste caso o valor de y será igual a 10. Isso mesmo, pois o – inverte o sinal e, lembrando lá do primário, menos com menos dá mais.

Outra observação é que o operador binário +, além da adição, também é utilizado para concatenar strings. Observe o exemplo da Listagem 4.1.

Listagem 4.1 – Concatenando strings – ex1cap4.cs.

```
using System;
using System.Windows.Forms;

class ex1cap4
{
    // Exemplo1 - Capítulo 4.
    // Concatenando Strings.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA
```

```

public static void Main()
{
    // Declaração das variáveis.

    string nome;
    string sobrenome;
    string nomecompleto;

    // O usuário informa o nome e o sobrenome.

    Console.Write("Digite o seu nome ->");
    nome = Console.ReadLine();

    Console.Write("Digite o seu sobrenome ->");
    sobrenome = Console.ReadLine();

    // Utilizando o operador + para concatenar
    // o nome e o sobrenome.

    nomecompleto = nome + " " + sobrenome;

    // Exibição dos resultados.

    MessageBox.Show(nomecompleto);
}
}

```

Digite o exemplo da Listagem 4.1 e salve o mesmo em um arquivo chamado ex1cap4.cs, na pasta C:\ExCsharp\cap4. Compile e execute o exemplo da Listagem 4.1. Para o nome digite José e pressione ENTER. Para o sobrenome digite da Silva e pressione ENTER. Você obterá os resultados indicados na Figura 4.2. Dê um clique no botão OK para fechar a janela de mensagem.

Temos uma novidade neste exemplo: A utilização do método Show, da classe MessageBox, do namespace System.Windows.Forms. Observe que no início do programa adicionamos uma referência ao namespace System.Windows.Forms. Para exibir o nome completo utilizamos MessageBox.Show.

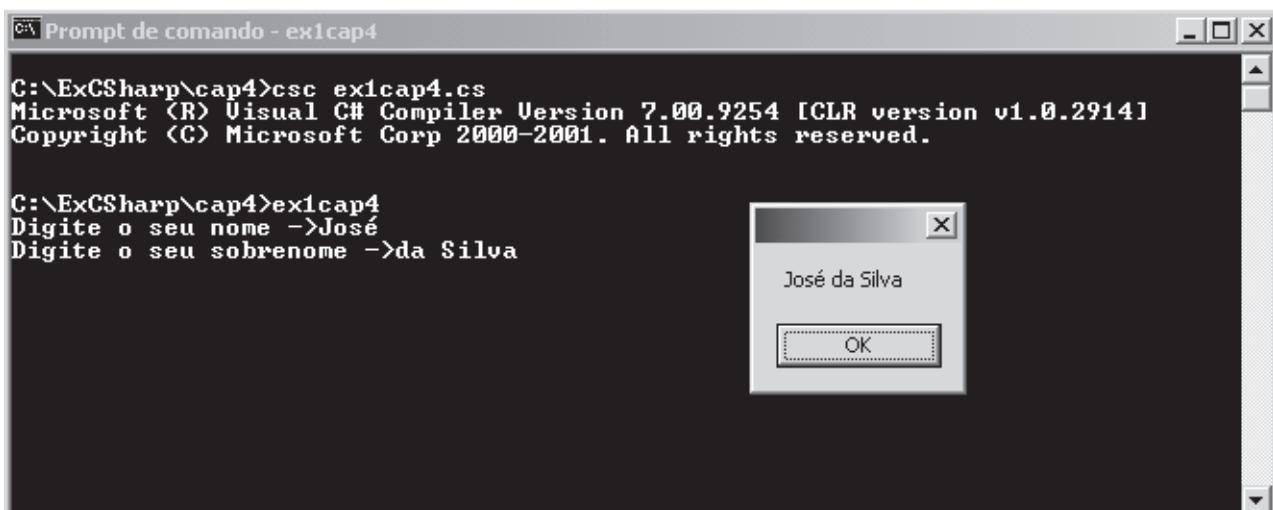


Figura 4.2: Executando o programa ex1cap4.exe.

Utilizamos o operador + para fazer a concatenação do nome, um espaço em branco (" ") e do sobrenome.

Operador de Incremento: ++

O operador de incremento aumenta o valor de uma variável em 1. O operador pode ser utilizado antes ou depois da variável. A melhor maneira de entendermos as diferenças entre o operador estar antes ou depois é através de um exemplo. Considere o seguinte trecho de código:

```
int x = 10;
int y = x++;
```

Quais os valores de x e y após a execução destes comandos?

```
x vale 11
y vale 10
```

Isso acontece porque, na segunda linha, primeiro y é feito igual ao valor atual de x (10) e depois x é incrementado passando o seu valor para 11.

Agora vamos alterar um pouco o nosso exemplo:

```
int x = 10;
int y = ++x;
```

Quais os valores de x e y após a execução destes comandos?

```
x vale 11
y vale 11
```

Isso acontece porque, na segunda linha, primeiro o x é incrementado, passando o seu valor para 11. Em seguida o novo valor de x (11) é atribuído para y.

Operador de Decremento —

O operador de decremento diminui o valor de uma variável em 1. O operador pode ser utilizado antes ou depois da variável. A melhor maneira de entendermos as diferenças entre o operador estar antes ou depois é através de um exemplo. Considere o seguinte trecho de código:

```
int x = 10;
int y = x-;
```

Quais os valores de x e y após a execução destes comandos?

```
x vale 9
y vale 10
```

Isso acontece porque, na segunda linha, primeiro y é feito igual ao valor atual de x (10) e depois x é decrementado passando o seu valor para 9.

Agora vamos alterar um pouco o nosso exemplo:

```
int x = 10;
int y = -x;
```

Quais os valores de x e y após a execução destes comandos?

```
x vale 9
y vale 9
```

Isso acontece porque, na segunda linha, primeiro o x é decrementado, passando o seu valor para 9. Em seguida o novo valor de x (9) é atribuído para y.

Operadores Relacionais e Lógicos

Os operadores relacionais são utilizados, principalmente, em testes nas estruturas de controle do C#. Neste tipo de teste são comparados dois valores e é retornado True ou False. Já os operadores lógicos são utilizados para operações do tipo AND, OR, etc.

Na Tabela 4.2 temos os principais operadores relacionais.

Tabela 4.2 Os principais operadores relacionais.

Operador	Exemplo	Descrição
<code>==</code>	<code>x == y</code>	Comparação. Retorna verdadeiro se x for igual a y.
<code>!=</code>	<code>x != y</code>	Diferente. Retorna verdadeiro se x for diferente de b.
<code><</code>	<code>x < y</code>	Menor do que. Retorna verdadeiro se x for menor do que y.
<code>></code>	<code>x > y</code>	Maior do que. Retorna verdadeiro se x for maior do que y.

Operador	Exemplo	Descrição
<code><=</code>	<code>x <= y</code>	Menor ou igual a. Retorna verdadeiro se x for menor ou igual a y.
<code>>=</code>	<code>x >= y</code>	Maior ou igual a. Retorna verdadeiro se x for maior ou igual a y.

Os operadores lógicos podem ser utilizados para a elaboração de testes mais complexos, inclusive testes compostos, onde temos dois ou mais testes ligados por operadores lógicos. Os dois principais operadores lógicos são `&&` (AND) e `o ||` (OR).

Quando temos uma expressão com vários testes ligados por operadores AND, o resultado somente será verdadeiro se todos os testes forem verdadeiros. Basta que um único teste seja falso, para que toda a expressão seja avaliada como falsa – False.

Considere o trecho de código a seguir:

```
If (10<11) && (5>3) && (3==3)
{
    Console.WriteLine("Este comando será executado !");
}
```

Neste exemplo, como todos os testes são verdadeiros, a expressão é verdadeira e o comando será executado. Agora considere o trecho de código a seguir:

```
If (10<11) && (5<3) && (3==3) & (123<=114)
{
    Console.WriteLine("Este comando NUNCA será executado !");
}
```

Neste exemplo, como um dos testes é falso (`5<3`), toda a expressão é avaliada como False e o comando no interior do laço NUNCA será executado.

Quando temos uma expressão com vários testes ligados por operadores OR, o resultado somente será falso se todos os testes forem falsos. Basta que um único teste seja verdadeiro, para que toda a expressão seja avaliada como verdadeira – True.

Considere o trecho de código a seguir:

```
If (10>11) && (5<3) && (3==3)
{
    Console.WriteLine("Este comando será executado !");
}
```

Neste exemplo, como um dos testes é verdadeiro ($3==3$), a expressão é verdadeira e o comando será executado. Agora considere o trecho de código a seguir:

```
If (10>11) && (5<3) && (3!=3) & (123>=114)
{
    Console.WriteLine("Este comando NUNCA será executado !");
}
```

Neste exemplo, como todos os testes são falsos, toda a expressão é avaliada como False e o comando no interior do laço NUNCA será executado.

Na Tabela 4.3 temos os principais operadores lógicos.

Tabela 4.3 Os principais operadores lógicos.

Operador	Descrição
&	Faz um AND bit a bit entre dois operandos.
	Faz um OR bit a bit entre dois operandos.
^	Faz um OR exclusivo (XOR) entre dois operandos.
&&	Faz um AND lógico entre dois operandos.
	Faz um OR lógico entre dois operandos.

Operadores de Atribuição (Assignment)

Temos operadores de atribuição simples e compostos. Temos um único operador de atribuição simples que é a igualdade (=), utilizado para atribuir um valor a uma variável ou o valor de uma variável à outra variável, conforme exemplo a seguir:

```
int x;
int y;
x = 123;
y = x;
```

Os operadores de atribuição compostos, além de fazerem a atribuição, realizam, ao mesmo tempo, uma determinada operação. Para entendermos os operadores compostos vamos a um exemplo simples com o operador +=. Considere o trecho de código a seguir:

```
int x;
int y;
y = 10;
x = 5;
x += y;
```

Neste caso, o valor de x será 15, pois o último comando `x += y` é equivalente à: `x = x + y`.

Na Tabela 4.4 temos os principais operadores de atribuição compostos.

Tabela 4.4 Os principais operadores de atribuição compostos.

Operador	Exemplo	É equivalente à:
<code>+=</code>	<code>x += y</code>	<code>x = x + y</code>
<code>-=</code>	<code>x -= y</code>	<code>x = x - y</code>
<code>*=</code>	<code>x *= y</code>	<code>x = x * y</code>
<code>/=</code>	<code>x /= y</code>	<code>x = x / y</code>
<code>%=</code>	<code>x %= y</code>	<code>x = x % y</code>

Precedência de Operadores

Em expressões complexas, onde podemos ter diversos operandos e operadores, deve existir uma maneira de decidir quais operações serão realizadas em primeiro lugar. A ordem em que as operações são realizadas é determinada pelas regras de precedência de operadores.

As regrinhas básicas são as mesmas das demais linguagens. Uma expressão é avaliada da esquerda para a direita. Os operadores de incremento e decremento após a variável têm maior prioridade (`x++` ou `x--`). Em seguida são avaliados os operadores unários (`+`, `-`, `!`) e os operadores de incremento e decremento antes da variável (`++x` e `-x`). Na seqüência vêm os operadores de multiplicação (`*`), divisão (`/`) e resto da divisão (`%`). Por último, os operadores de adição (`+`) e subtração (`-`).

Podemos alterar a ordem de precedência dos operadores utilizando parênteses. Considere a expressão a seguir:

`x = (y+z)*k`

Neste caso, a adição (`y+z`) será executada em primeiro lugar, pois a mesma está dentro do parênteses. O resultado da adição é então multiplicado por k.

Para ilustrar a ordem de precedência vamos considerar o exemplo da Listagem 4.2

Listagem 4.2 – Precedência de operadores. – ex2cap4.cs

```
using System;

class ex2cap4
{
    // Exemplo 2 - Capítulo 4.
```

```
// Precedência de operadores.  
// Por: Júlio Battisti  
// MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA  
  
public static void Main()  
{  
  
    // Declaração das variáveis.  
  
    int i=5;  
    int j=21;  
    int k=10;  
    int l=20;  
    int m=4;  
    int expr1;  
    int expr2;  
    int expr3;  
  
    // Alguns cálculos  
  
    expr1 = i++*j+k-l*m++;  
    expr2 = (i++*j)+(k-1)*m++;  
    expr3 = (i++*j+k)-l*m++;  
  
    // Exibição dos resultados.  
  
    Console.WriteLine("*****");  
    Console.WriteLine("Valor de expr1: {0}",expr1);  
    Console.WriteLine("Valor de expr2: {0}",expr2);  
    Console.WriteLine("Valor de expr3: {0}",expr3);  
    Console.WriteLine("*****");  
  
}  
}
```

Digite o exemplo da Listagem 4.2 e salve o mesmo em um arquivo chamado ex2cap4.cs, na pasta C:\ExCsharp\cap4. Compile e execute o exemplo da Listagem 4.2. Você obterá os resultados indicados na Figura 4.3.

```
C:\ExCSharp\cap4>csc ex2cap4.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap4>ex2cap4
*****
Valor de expr1: 35
Valor de expr2: 76
Valor de expr3: 37
*****
```

Figura 4.3: Executando o programa ex2cap4.exe.

Observe que temos a mesma expressão, e apenas utilizamos parênteses para mudar a ordem de execução dos elementos da expressão. Vamos analisar a primeira expressão, na qual resultou em $\text{expr1} = 35$.

expr1 = i+++j+k-1*m++;

Aqui não temos nenhum parênteses. Neste caso vale a ordem de precedência dos operadores. Em primeiro, na ordem de precedência estão os operadores de incremento. Porém cabe lembrar que, para a expressão, como o operador de incremento está após a variável, é utilizado o valor original de cada variável e somente depois disso é que as variáveis são incrementadas. Então, para a primeira expressão serão utilizados os seguintes valores:

```
i=5
j=21
k=10
l=20
m=4
```

Com isso nossa expressão fica:

expr1 = 5*21+10-20*4

Primeiro resolvo as multiplicações:

expr1 = 105+10-80

no que obtemos:

expr1 = 35

e, após o cálculo da expressão, os valores de i e m são incrementados, passando a valer:

i=6

j=21

k=10

l=20

m=5

Se o operador de incremento estivesse antes das variáveis, primeiro as mesmas seriam incrementadas e depois, já com o novo valor, a expressão seria calculada.

Classes

O conceito de Classes é fundamental para a orientação a objetos. Como o Framework .NET é fortemente baseado nos conceitos de orientação a objetos, é óbvio que o conceito de Classes é igualmente importante para o Framework .NET.

Para sair do discurso teórico e partir para a prática basta analisar os exemplos que apresentamos até o momento. Não houve um único exemplo em que não utilizamos pelo menos uma classe do .NET Framework Class Library (biblioteca de classes do .NET). Conforme já havíamos salientado, a biblioteca de classes do .NET nos oferece uma infinidade de classes que podem ser utilizadas em nossos programas e páginas.

Através da utilização de classes estamos viabilizando, na prática, os princípios da herança e da reutilização de código. Ao criarmos uma classe estamos, na verdade, definindo as características de um novo objeto. A definição da classe contém os métodos, propriedades, indexadores (novidade no .NET) e Eventos da classe. Também contém informação sobre a visibilidade dos elementos da classe. A visibilidade define, por exemplo, se um método da classe somente poderá ser utilizado por outros métodos da própria classe ou poderá ser utilizado externamente à classe.

Uma vez definida a classe, podemos utilizá-la em nossos programas e páginas. Para utilizar uma classe criamos e instanciamos um objeto como sendo do tipo definido pela classe. A partir deste momento o objeto passa a ter acesso a todos os elementos – métodos, propriedades, etc., da classe. Em outras palavras: o objeto herda todos os elementos da classe.

Neste capítulo aprenderemos a criar e utilizar classes no C#. Também aprenderemos a criar os diversos elementos que compõem uma classe.

Para Começar um Exemplo Simples

Vamos aprender sobre a estrutura de uma classe criando uma. Criaremos uma classe extremamente simples. Daremos o nome de ClasseEx1. O único componente que a nossa classe irá conter será a definição de duas variáveis: um inteiro e uma string. No programa principal iremos criar uma instância da classe e utilizar as variáveis da mesma.

Considere o exemplo da Listagem 4.3.

Listagem 4.3 – Uma classe bastante simples. – Ex4Cap4.cs

```
using System;
class ClasseEx1
{
    //Definição dos membros da classe.

    public int contador = 0;
    public string teste = " ";
}

class Ex3cap4

{
    // Exemplo 3 - Capítulo 4.
    // Criação de uma classe com dois membros.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {

        // Agora crio uma variável do tipo ClasseEx1.
        // Esta variável é uma instância da classe ClasseEx1.
        ClasseEx1 minhaclasse = new ClasseEx1();

        //Definição de novos valores para os membros da classe.

        minhaclasse.contador = 125;
        minhaclasse.teste = "PRIMEIRA CLASSE COM O C#";

        //Agora exibo os resultados.

        Console.WriteLine("*****");
        Console.WriteLine("VALOR DO MEMBRO INTEIRO: {0}",minhaclasse.contador);
        Console.WriteLine("VALOR DO MEMBRO STRING: {0}",minhaclasse.teste);
        Console.WriteLine("*****");
    }
}
```

Digite o exemplo da Listagem 4.3 e salve o mesmo em um arquivo chamado ex3cap4.cs, na pasta C:\ExCsharp\cap4. Compile e execute o exemplo da Listagem 4.3. Você obterá os resultados indicados na Figura 4.4.

```
C:\ExCSharp\cap4>csc ex3cap4.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap4>ex3cap4
*****
VALOR DO MEMBRO INTEIRO: 125
VALOR DO MEMBRO STRING: PRIMEIRA CLASSE COM O C#
*****
```

Figura 4.4: Executando o programa ex3cap4.exe.

Temos vários detalhes importantes a considerar neste exemplo.

Em primeiro lugar a definição da classe ClasseEx1, a qual é feita no seguinte trecho de código. Para definir uma nova classe utilizamos a palavra class. A definição dos elementos da classe acontece dentro do abre e fecha chaves { }.

```
class ClasseEx1
{
    //Definição dos membros da classe.

    public int contador = 0;
    public string teste = " ";
}
```

Neste exemplo definimos dois campos:

- ◆ contador
- ◆ teste

Estes campos são também chamados de membros da classe (na verdade membro é qualquer elemento da classe, quer seja uma propriedade, um campo, um método, etc.). A palavra public faz com que estes campos possam ser acessados fora da classe, isto é, dentro do código do programa que estiver utilizando a classe.

Após a definição da classe temos uma segunda classe, chamada Ex3cap4.

```
class Ex3cap4
```

Esta é a classe principal do nosso exemplo, a qual conterá uma função Main(). Dentro da função Main() criamos um objeto chamado minhaclasse. Observe que este objeto é definido como sendo do tipo da classe criada anteriormente, ou seja, o objeto minhaclasse é baseado na classe ClasseEx1. A declaração e instanciação da variável objeto minhaclasse é feita na seguinte linha de código:

```
ClasseEx1 minhaclasse = new ClasseEx1();
```

Neste caso a novidade é o uso do operador new para a instanciação do objeto.

Após termos criado uma objeto baseado na classe ClasseEx1 podemos acessar os seus membros public. Para acessar um membro utilizamos a seguinte sintaxe:

```
nome_da_classe.nome_do_membro
```

No trecho de código a seguir alteramos o valor do campo contador e do campo teste, do objeto minhaclasse:

```
minhaclasse.contador = 125;  
minhaclasse.teste = "PRIMEIRA CLASSE COM O C#";
```

O restante do exemplo é apenas para exibição dos resultados.

Membros de uma Classe

A seguir apresentamos uma descrição dos elementos que podem ser criados em uma classe, isto é, dos possíveis membros de uma classe.

Campos (Fields)

Um campo é um membro do tipo variável utilizado para armazenar um determinado valor. No exemplo da Listagem 4.3 criamos dois campos: contador e teste. Os campos são os dados do objeto e podem determinar, juntamente com as propriedades, o estado do objeto em um determinado instante. Para cada membro de uma classe podemos aplicar modificadores, os quais descrevem a maneira como um determinado membro pode ser utilizado. Por exemplo, se queremos que um campo possa ser acessado fora da classe, precisamos utilizar o modificador public antes da definição do campo, como fizemos no exemplo da Listagem 4.3. Se quisermos que o campo somente seja acessado de dentro da classe podemos utilizar o modificador private. Outros modificadores para campos são: static, readonly e const. Falaremos sobre estes modificadores mais adiante.

Métodos (Methods)

Um método contém código que atua sobre os dados do objeto. Os métodos, falando em termos de orientação a objetos, descrevem as ações que podem ser realizadas pela classe. Na prática, a funcionalidade de uma classe é implementada através dos seus métodos.

IMPORTANTE: Observe que não “destruímos” o objeto minhaclasse no final do programa. Este procedimento não é mais necessário, como era em versões anteriores do VB, Visual C++ e VBA. Agora, quando a função Main() encerra a sua execução, a referência ao objeto minhaclasse deixa de existir. Se a referência não foi gravada, a instância da classe estará à disposição do coleto de lixo (garbage collector) do .NET, o qual irá liberar os recursos ocupados pela classe, quando necessário.

Propriedades (Properties)

As propriedades são um pouco diferentes dos campos, pois para definir e ler o valor de uma propriedade são definidos métodos específicos. Get para obter o valor da propriedade e Set para definir o valor da propriedade. Veremos como criar e utilizar propriedades ainda neste capítulo.

Constantes (Constants)

Uma constante é um campo cujo valor não pode ser alterado. Aprenderemos a criar constantes ainda neste capítulo.

Indexadores (Indexers)

Esta é uma novidade do .NET. Um indexador é um array. O indexador permite que, facilmente, possamos indexar um objeto para facilitar a definição e obtenção dos valores da classe.

Eventos (Events)

Um evento, normalmente, corresponde a uma ação do usuário, ação esta que é detectada pela classe e dispara a execução de algum código em resposta ao evento. É o conceito de evento que todos nós já conhecemos. Por exemplo, ao clicar em um botão de comando, é disparado o evento Ao Clicar.

Agora aprenderemos a criar os diversos membros de uma classe. Mas antes vamos falar um pouco sobre Construtores (Constructs) e Destrutores (Destructors).

Construtores e Destrutores de uma Classe

Ao criarmos uma classe podemos definir um método que será executado toda vez que uma instância da classe for criada. Este método especial é chamado de Construtor (Constructor) da classe. Colocamos neste método todo o código que deve ser executado na criação da classe. Por exemplo, se estamos criando uma classe que possui métodos para fazer pesquisas em um banco de dados do SQL Server 2000. No método Construtor podemos incluir o código que estabelece a conexão com o banco de dados. Para que a classe seja de uso genérico, podemos definir parâmetros a serem passados para o método Construtor, de tal forma que a conexão seja estabelecida de acordo com o valor dos parâmetros passados para o método construtor. Se não definirmos um método construtor explicitamente, o Framework .NET define um método construtor sem parâmetros. No nosso exemplo anterior utilizamos o seguinte comando para criar uma instância da classe ClasseEx1 para o objeto minhaclasse:

```
ClasseEx1 minhaclasse = new ClasseEx1();
```

O método construtor criado por padrão possui o mesmo nome da classe e não possui parâmetros – ClasseEx1(). Se tivéssemos definido um método construtor com parâmetros, os mesmos teriam que ser passados no momento da criação do objeto. Por exemplo, se o método construtor deve receber um parâmetro inteiro, a criação do objeto minhaclasse ficaria assim:

```
ClasseEx1 minhaclasse = new ClasseEx1(10);
```

Observe que estamos passando um parâmetro (10) para o método construtor.

A principal vantagem de utilizarmos um método construtor personalizado é que podemos garantir que o objeto seja instanciado corretamente, através da execução do código necessário à instanciação do objeto, no método construtor do mesmo. Quando o usuário instancia um determinado objeto, o método construtor do mesmo é chamado e deve finalizar a sua execução sem erros; caso contrário o usuário não poderá utilizar o objeto. Este comportamento garante que o código de inicialização, contido no método construtor do objeto, seja executado, antes que os demais membros do objeto possam ser utilizados.

O método construtor deve ter o mesmo nome da classe. Para provar que o método construtor é sempre executado, vamos considerar o exemplo da Listagem 4.4.

Considere o exemplo da Listagem 4.4.

Listagem 4.4 – Utilizando o método construtor. – Ex4cap4.cs

```
using System;
class ClasseEx2
{
    // Definição do construtor da classe.
    // Observe que o nome do método construtor
    // deve ter o mesmo nome da classe.

    public ClasseEx2()
    {
        Console.WriteLine("*****");
        Console.WriteLine("EXECUTANDO OS COMANDOS DO CONSTRUTOR !");
        Console.WriteLine("ESTES COMANDOS SERÃO EXECUTADOS SEMPRE");
        Console.WriteLine("QUE UMA INSTÂNCIA DA CLASSE FOR CRIADA");
        Console.WriteLine("*****");
    }

    public int contador = 0;
    public string teste = " ";
}

class Ex4cap4
```

```

// Exemplo 3 - Capítulo 4.

// Criação de uma classe com dois membros.

// Por: Júlio Battisti

// MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

public static void Main()
{
    // Agora crio uma variável do tipo ClasseEx2.

    // Esta variável é uma instância da classe ClasseEx2.

    // A simples criação de um objeto da classe fará
    // com que o seu construtor seja disparado e os
    // comandos do construtor serão executados.

    ClasseEx2 minhaclasse = new ClasseEx2();

}

}

```

Digite o exemplo da Listagem 4.4 e salve o mesmo em um arquivo chamado ex4cap4.cs, na pasta C:\ExCsharp\cap4. Compile e execute o exemplo da Listagem 4.4. Você obterá os resultados indicados na Figura 4.5.

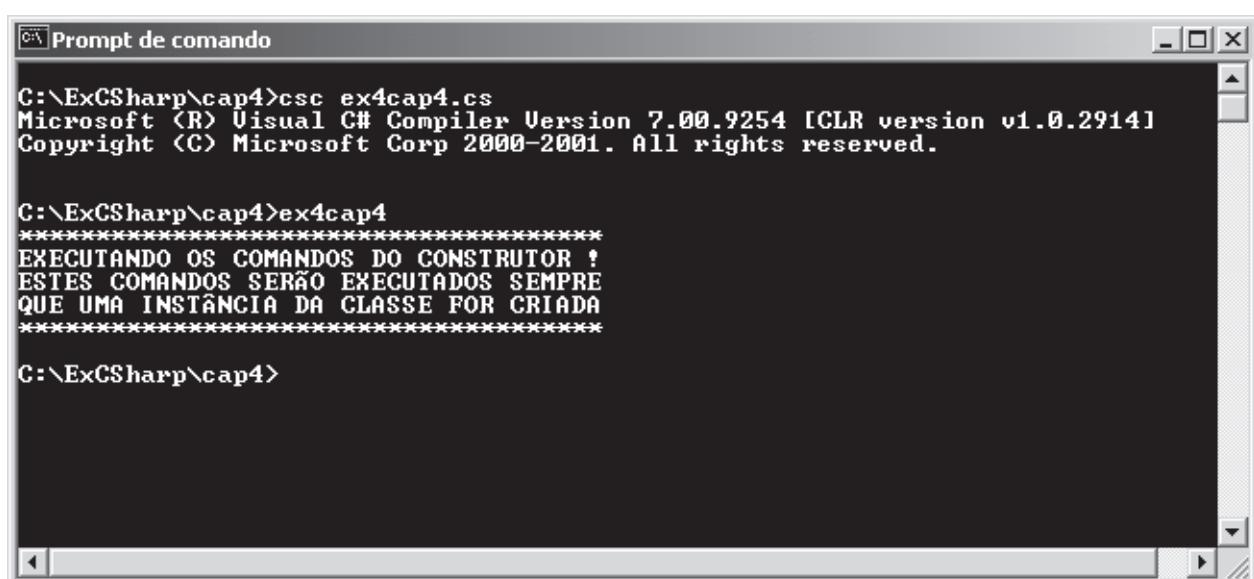


Figura 4.5: Executando o programa ex4cap4.exe.

No nosso exemplo, no programa principal simplesmente criamos um objeto do tipo ClasseEx2. Ao instanciarmos o objeto, foi disparado o seu método construtor, o qual exibe algumas informações na tela.

Além de um método construtor podemos criar um método destrutor. O método destrutor pode ser chamado para executar alguns comandos quando a referência ao objeto deixar de existir, ou seja, os comandos do método destrutor são chamados quando do encerramento do objeto.

Para criar um método destrutor basta criar um método com o mesmo nome da classe, porém precedido de um til (~). Por exemplo, para criar um método destrutor para a classe ClasseEx2(), criada no exemplo anterior, utilizamos a seguinte sintaxe:

```
~ClasseEx2 ()
{
    comando1;
    comando2;
    ...
    comandon;
}
```

Modificadores Para os Membros de uma Classe

Conforme descrito anteriormente existem modificadores que definem a visibilidade dos membros de uma classe. Por exemplo, o modificador public torna o membro acessível de fora da classe; já o modificador private torna o membro disponível somente dentro da própria classe. A seguir uma descrição dos principais modificadores.

- ◆ **public:** Torna o membro acessível de fora da definição da classe.
- ◆ **protected:** O membro não pode ser acessado fora da classe, porém o membro está disponível para outras classes derivadas da classe base.
- ◆ **private:** O membro não pode ser acessado fora da classe, nem mesmo por outras classes derivadas da classe base.
- ◆ **internal:** O membro somente é visível na unidade de código onde o mesmo está definido. É um meio-termo entre public e protected, uma vez que o membro pode ser acessado por todas as classes definidas na mesma unidade (público para as classes da mesma unidade), porém não pode ser acessado por classes definidas em outras unidades (protected para unidades definidas em outras unidades).

Adicionando Funcionalidade a uma Classe – Métodos

A funcionalidade de uma classe é definida através da criação de métodos. Um método pode receber parâmetros e retornar um determinado resultado. Vamos fazer um pequeno estudo dos tipos de parâmetros que podem ser definidos para um determinado método.

Podemos definir os seguintes tipos de parâmetros:

- ◆ **Parâmetros In:** Este tipo de parâmetro é o mais conhecido. É utilizado para passarmos o valor de uma variável para o método. Um inconveniente deste tipo de parâmetro é que o método não pode alterar o valor da variável, uma vez que somente foi passado o valor da variável e não uma referência para a mesma. O parâmetro definido no método é inicializado com uma cópia do valor da variável passada como parâmetro na chamada do método.

Mais uma vez vamos fazer uso de um exemplo para entendermos a criação de métodos e a utilização de parâmetros in.

Considere o exemplo da Listagem 4.5.

Listagem 4.5 – Utilizando parâmetro do tipo in. – Ex5cap4.cs

```
using System;

public class ClasseEx3
{
    // Definimos um método para calcular o cubo de
    // uma variável passada como parâmetro para o método.

    public int calcula_cubo(int valor)
    {
        // Crio uma variável local do tipo int.
        // Atribuo o valor do parâmetro valor para a
        // variável local.

        int auxint;
        auxint = valor;

        // Altero o valor do parâmetro recebido.

        valor = 6;

        return auxint*auxint*auxint;
    }

}

class Ex5cap4
```

```
// Exemplo 5 - Capítulo 4.  
// Criação de uma classe com um método  
// que calcula o cubo de um número inteiro.  
// Parâmetros in.  
// Por: Júlio Battisti  
// MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA  
  
public static void Main()  
{  
  
    // Agora crio uma variável do tipo ClasseEx3.  
    // Esta variável é uma instância da classe ClasseEx3.  
  
    ClasseEx3 minhaclasse = new ClasseEx3();  
  
    // Defino uma variável do tipo int e chamo o método  
    // calcula_cubo, da classe minhaclasse, passando o  
    // valor inteiro como parâmetro.  
  
    int valor;  
  
    // Solicito que o usuário digite um valor inteiro.  
  
    Console.WriteLine("Digite um número inteiro ->");  
    string aux1 = Console.ReadLine();  
    valor = Convert.ToInt32(aux1);  
  
    // Exibição do resultado.  
  
    Console.WriteLine("*****");  
    Console.WriteLine("O NÚMERO DIGITADO FOI      -> {0}",valor);  
    Console.WriteLine("*****");  
    Console.WriteLine("ELEVADO AO CUBO -> {0}",minhaclasse.calcula_cubo(valor));  
    Console.WriteLine("*****");  
    Console.WriteLine("VALOR ORIGINAL DA VARIÁVEL VALOR -> {0}",valor);  
    Console.WriteLine("*****");  
  
}  
}
```

Digite o exemplo da Listagem 4.5 e salve o mesmo em um arquivo chamado ex5cap4.cs, na pasta C:\ExCsharp\cap4. Compile e execute o exemplo da Listagem 4.5. Você obterá os resultados indicados na Figura 4.6.

```
C:\ExCSharp\cap4>csc ex5cap4.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap4>csc ex5cap4.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap4>ex5cap4
Digite um número inteiro ->10
*****
O NÚMERO DIGITADO FOI      -> 10
*****
ELEVADO AO CUBO -> 1000
*****
VALOR ORIGINAL DA VARIÁVEL VALOR -> 10
*****
```

Figura 4.6: Executando o programa ex5cap4.exe.

Temos muitos detalhes interessantes a serem comentados no código deste exemplo.

Vamos iniciar pela definição de um método para calcular o cubo de um número passado como parâmetro. Isso é feito pelo trecho de código a seguir:

```
// Definimos um método para calcular o cubo de
// uma variável passada como parâmetro para o método.

public int calcula_cubo(int valor)
{
    // Crio uma variável local do tipo int.
    // Atribuo o valor do parâmetro valor para a
    // variável local.

    int auxint;
    auxint = valor;

    // Altero o valor do parâmetro recebido.

    valor = 6;

    return auxint*auxint*auxint;
}
```

O método recebe um parâmetro do tipo inteiro chamado valor. Como o parâmetro é passado por valor, o método recebe uma cópia do valor da variável e não uma referência para a mesma. Neste caso temos um parâmetro in. Dentro do método declaramos uma variável do tipo int chamada auxint. Atribuímos o valor recebido como parâmetro para a variável auxint, através do comando:

```
auxint = valor;
```

Em seguida modificamos o valor da variável recebida como parâmetro. Fizemos isso para provar que esta modificação não altera o valor da variável original, definida no programa principal. Este fato pode ser comprovado na Figura 4.6, onde a variável original continua com o valor digitado pelo usuário, ou seja, a alteração feita no método calcula_cubo não afetou a variável original. Este é o comportamento esperado, uma vez que o parâmetro foi passado por valor e não por referência.

A última linha do método faz o cálculo, retornando o valor passado (agora armazenado na variável auxint), elevado ao cubo.

No programa principal declaramos e instanciamos uma variável chamada minhaclasse, a qual é do tipo ClasseEx3. Isto é feito com a seguinte linha de código:

```
ClasseEx3 minhaclasse = new ClasseEx3();
```

Uma vez criado um objeto do tipo ClasseEx3 temos acesso ao método calcula_cubo(valor), ou seja: minhaclasse.calcula_cubo(valor), o qual é utilizado na seguinte linha de código:

```
Console.WriteLine("ELEVADO AO CUBO -> {0}",minhaclasse.calcula_cubo(valor));
```

Este é um exemplo simples mas que salienta a criação e utilização de um método público. Também comprovamos que as alterações feitas em um parâmetro passado por valor não afetam o valor da variável no programa principal.

- ◆ **Parâmetros ref:** Quando utilizamos parâmetros do tipo ref, como o próprio nome sugere, é passada uma referência para o parâmetro e não somente o valor do mesmo. Na prática isto significa que, se o método alterar o valor do parâmetro, esta alteração terá efeito na variável original, no programa principal. Vamos utilizar o mesmo exemplo anterior, porém apenas modificando o tipo de parâmetro. Aí vamos conferir se o valor da variável original, no programa principal, foi modificado.

Considere o exemplo da Listagem 4.6.

Listagem 4.6 – Utilizando parâmetro do tipo ref. – Ex6cap4.cs

```
using System;  
  
public class ClasseEx4  
{  
    // Definimos um método para calcular o cubo de  
    // uma variável passada como parâmetro para o método.  
  
    public int calcula_cubo(ref int valor)  
    {
```

```
// Crio uma variável local do tipo int.  
// Atribuo o valor do parâmetro valor para a  
// variável local.  
  
int auxint;  
auxint = valor;  
  
// Altero o valor do parâmetro recebido.  
  
valor = 6;  
  
return auxint*auxint*auxint;  
}  
  
}  
  
class Ex6cap4  
  
{  
    // Exemplo 6 - Capítulo 4.  
    // Criação de uma classe com um método  
    // que calcula o cubo de um número inteiro.  
    // Parâmetros ref.  
    // Por: Júlio Battisti  
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA  
  
    public static void Main()  
    {  
  
        // Agora crio uma variável do tipo ClasseEx4.  
        // Esta variável é uma instância da classe ClasseEx4.  
  
        ClasseEx4 minhaclasse = new ClasseEx4();  
  
        // Defino uma variável do tipo int e chamo o método  
        // calcula_cubo, da classe minhaclasse, passando o
```

```

// valor inteiro como parâmetro.

int valor;

// Solicito que o usuário digite um valor inteiro.

Console.WriteLine("Digite um número inteiro ->");
string aux1 = Console.ReadLine();
valor = Convert.ToInt32(aux1);

// Exibição do resultado.

Console.WriteLine("*****");
Console.WriteLine("O NÚMERO DIGITADO FOI      -> {0}", valor);
Console.WriteLine("*****");
Console.WriteLine("ELEVADO AO CUBO -> {0}", minhaclasse.calcula_cubo(ref valor));
Console.WriteLine("*****");
Console.WriteLine("VALOR ORIGINAL DA VARIÁVEL VALOR -> {0}", valor);
Console.WriteLine("*****");

}

}

```

Digite o exemplo da listagem 4.6 e salve o mesmo em um arquivo chamado ex6cap4.cs, na pasta C:\ExCsharp\cap4. Compile e execute o exemplo da listagem 4.6. Você obterá os resultados indicados na Figura 4.7.

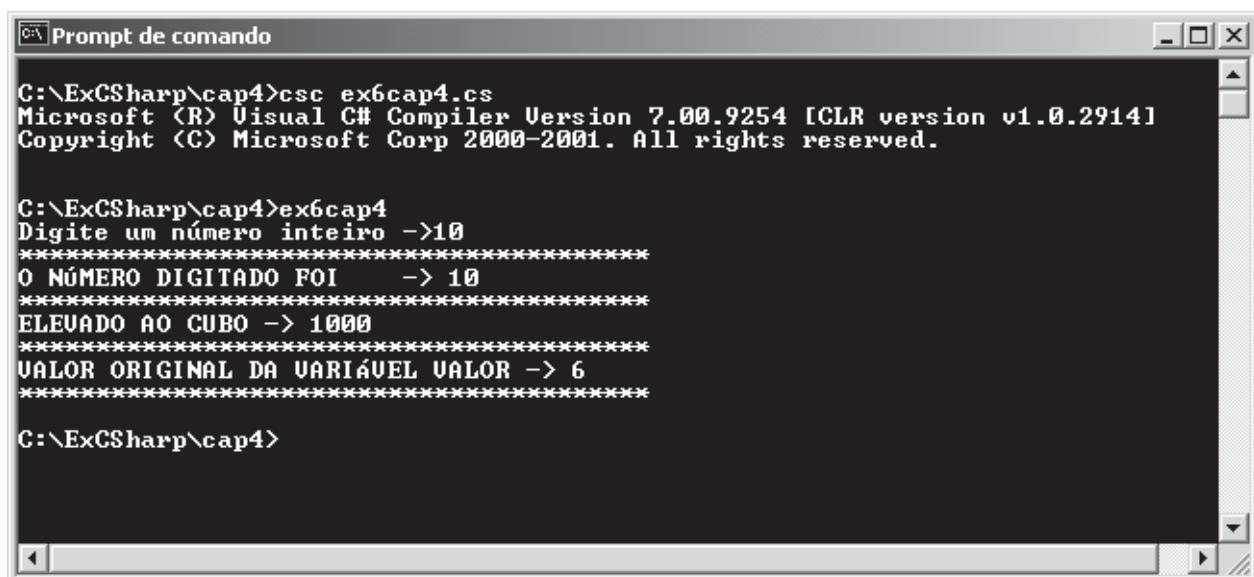


Figura 4.7: Executando o programa ex6cap4.exe.

Vamos comentar as diferenças em relação ao exemplo anterior.

Em primeiro lugar, na definição do método calcula_cubo, utilizamos a palavra ref para definir que o parâmetro será passado por referência, conforme indicado no trecho a seguir:

```
public int calcula_cubo(ref int valor)
{
    // Crio uma variável local do tipo int.
    // Atribuo o valor do parâmetro valor para a
    // variável local.

    int auxint;
    auxint = valor;

    // Altero o valor do parâmetro recebido.

    valor = 6;

    return auxint*auxint*auxint;
}
```

Neste caso, como o parâmetro está sendo passado por referência, a alteração feita no parâmetro valor terá reflexos na variável valor no programa principal, o que pode ser comprovado pelos resultados exibidos na Figura 4.7.

Quando passamos um parâmetro por referência, na verdade estamos passando o endereço da variável. Por isso quando o método altera o valor da variável passada como parâmetro está, na prática, alterando o valor da variável original.

Também é importante salientar que no momento de utilizarmos o método, a palavra ref deve ser colocada antes do parâmetro que está sendo passado, conforme indicado na linha de código a seguir:

```
Console.WriteLine("ELEVADO AO CUBO -> {0}", minhaclasse.calcula_cubo(ref valor));
```

- ◆ **Parâmetros out:** Um parâmetro do tipo out é utilizado para que o método possa passar um resultado de volta para o programa que fez a chamada ao método. O programa que está chamando o método não precisa inicializar a variável que será retornada.

Vamos modificar o exemplo anterior para introduzirmos um parâmetro do tipo out.

Considere o exemplo da Listagem 4.7.

Listagem 4.7 – Utilizando parâmetro do tipo out. – Ex7cap4.cs

```
using System;
public class ClasseEx4
{
    // Definimos um método para calcular o cubo de
    // uma variável passada como parâmetro para o método.

    public void calcula_cubo(int valor, out int valoracubo)
    {
        valoracubo = valor*valor*valor;
    }

}

class Ex7cap4
{

    // Exemplo 7 - Capítulo 4.
    // Criação de uma classe com um método
    // que calcula o cubo de um número inteiro.
    // Parâmetros out.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {

        // Agora crio uma variável do tipo ClasseEx4.
        // Esta variável é uma instância da classe ClasseEx4.

        ClasseEx4 minhaclasse = new ClasseEx4();

        // Defino uma variável do tipo int e chamo o método
        // calcula_cubo, da classe minhaclasse, passando o
        // valor inteiro como parâmetro.
    }
}
```

```

int valor;
int valoraocubo;

// Solicito que o usuário digite um valor inteiro.

Console.WriteLine("Digite um número inteiro ->");
string aux1 = Console.ReadLine();
valor = Convert.ToInt32(aux1);

// Chamada do método calcula_cubo.

// Após a execução do método o valor elevado
// ao cubo está disponível na variável valoraocubo
// a qual foi retornada pelo método.

minhaclasse.calcula_cubo(valor,out valoraocubo);

// Exibição do resultado.

Console.WriteLine("*****");
Console.WriteLine("O NÚMERO DIGITADO FOI      -> {0}",valor);
Console.WriteLine("*****");
Console.WriteLine("ELEVADO AO CUBO -> {0}",valoraocubo);
Console.WriteLine("*****");
Console.WriteLine("VALOR ORIGINAL DA VARIÁVEL VALOR -> {0}",valor);
Console.WriteLine("*****");

}

}

```

Digite o exemplo da Listagem 4.7 e salve o mesmo em um arquivo chamado ex7cap4.cs, na pasta C:\ExCsharp\cap4. Compile e execute o exemplo da Listagem 4.7. Você obterá os resultados indicados na Figura 4.8.

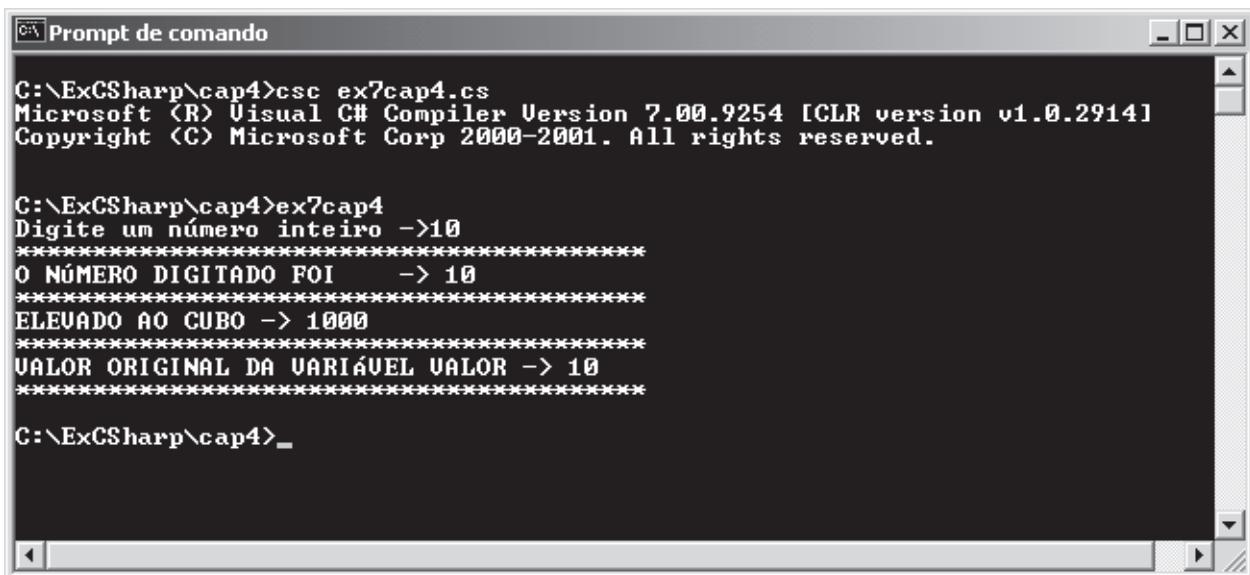


Figura 4.8 Executando o programa ex7cap4.exe.

Novamente mudamos o método calcula_cubo. Agora o método recebe um parâmetro inteiro por valor e retorna o cubo deste parâmetro através do parâmetro valoraocubo. O parâmetro valoraocubo é definido como um parâmetro de retorno ou saída (out), pelo uso da palavra out. O valor que deve ser retornado é atribuído ao parâmetro valoraocubo pelo código do método, conforme indicado no fragmento de código a seguir:

```
public void calcula_cubo(int valor, out int valoraocubo)
{
    valoraocubo = valor*valor*valor;
}
```

Também é importante salientar a utilização da palavra void antes do nome do método. O void informa que não será retornado nenhum valor pelo próprio método. O valor será retornado através da utilização de um parâmetro do tipo out.

A utilização do método calcula_cubo, no programa principal, também é diferente dos exemplos anteriores. Neste exemplo primeiro declaramos uma variável do tipo int:

```
int valoraocubo;
```

Esta variável receberá o valor retornado pelo método calcula_cubo. Em seguida fazemos uma chamada ao método calcula_cubo, passando como primeiro parâmetro o valor digitado pelo usuário e o segundo parâmetro é o nome da variável que receberá o valor de retorno, conforme indicado no código a seguir:

```
minhaclasse.calcula_cubo(valor,out valoraocubo);
```

O restante do programa é simplesmente a exibição dos resultados obtidos.

O Polimorfismo Posto em Prática – Override

Pelo princípio do Polimorfismo em uma classe derivada podemos sobrescrever (override), isto é, redefinir um ou mais métodos da classe base, desde que ao criar a classe básica o método tenha sido definido com permissão para ser redefinido. Para criar um método e permitir que o mesmo seja redefinido em uma classe derivada, utilizamos a palavra virtual. Considere o exemplo de definição a seguir:

```
virtual void metodo_que_pode_ser_redefinido(parâmetros)
```

Na classe derivada, para redefinirmos o método utilizamos a palavra override:

```
override void metodo_que_pode_ser_redefinido(parâmetros)
```

Para vermos o princípio do polimorfismo em ação vamos a um exemplo prático. Criaremos uma classe chamada Funcionários, na qual definimos um método chamado calcula_salario. Depois criaremos uma classe baseada na classe Funcionários, a qual faz um override do método calcula_salário.

Considere o exemplo da Listagem 4.8.

Listagem 4.8 – Fazendo o override de métodos no C#. – Ex8cap4.cs

```
using System;

public class ClasseFuncionarios
{
    // Definimos um método para calcular o salário
    // com base no cargo que foi passado como parâmetro.

    public virtual int calcula_salario(string cargo)
    {

        switch (cargo)
        {
            case "gerente":
                return 2500;
            case "diretor":
                return 4500;
            case "funcionario":
                return 1500;
            default:
                return 1500;
        }
    }
}
```

```
}

// Agora crio uma classe derivada de ClasseFuncionarios
// chamada HerdaDeFuncionarios.

// Nesta nova classe redefino o método calcula_salario

public class HerdaDeFuncionarios:ClasseFuncionarios
{
    // Vamos redefinir o método para calcular o salário

    public override int calcula_salario(string cargo)
    {
        switch (cargo)
        {
            case "gerente":
                return 3000;
            case "diretor":
                return 4000;
            case "funcionario":
                return 2000;
            default:
                return 1000;
        }
    }
}

// Agora vamos definir o programa principal, onde
// utilizaremos o método que foi redefinido.

class Ex8cap4
{
    // Exemplo 8 - Capítulo 4.
    // Exemplo de override de método.
    // Por: Júlio Battisti
```

```
// MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

public static void Main()
{
    // Crio um objeto FuncionarioOriginal baseado na classe
    // ClasseFuncionarios.

    ClasseFuncionarios FuncionarioOriginal = new ClasseFuncionarios();

    // Chamo o método originalmente definido na classe
    // Passando como parâmetro "gerente"

    String aux1 = "gerente";

    Console.WriteLine("*****");
    Console.WriteLine("RESULTADOS DO MÉTODO ORIGINAL");
    Console.WriteLine("VALOR DO SALÁRIO PARA {0}, É -> {1}", aux1, FuncionarioOriginal.calcula_salario(aux1));

    // Crio um objeto FuncionarioModificado baseado na classe
    // HerdaFuncionarios.

    HerdaDeFuncionarios FuncionarioModificado = new HerdaDeFuncionarios();

    // Chamo o método que foi sobreescrito na classe HerdaDeFuncionarios
    // Passando como parâmetro "gerente"

    String aux2 = "gerente";

    Console.WriteLine("*****");
    Console.WriteLine("RESULTADOS DO MÉTODO REDEFINIDO");
    Console.WriteLine("VALOR DO SALÁRIO PARA {0}, É -> {1}", aux1, FuncionarioModificado.calcula_salario(aux2));
}

}
```

IMPORTANTE: Alguns comandos aparecem divididos em duas linhas, por questão de espaço para exibição. Estes comandos devem ser digitados em uma única linha. A seguir temos um exemplo de um comando que está dividido em duas linhas por questões de espaço:

```
Console.WriteLine("VALOR DO SALÁRIO PARA {0}, É ->
{1}",aux1,FuncionarioModificado.calcula_salario(aux2));
```

Digite o exemplo da Listagem 4.8 e salve o mesmo em um arquivo chamado ex8cap4.cs, na pasta C:\ExCsharp\cap4. Compile e execute o exemplo da Listagem 4.8. Você obterá os resultados indicados na Figura 4.9.

```
C:\ExCSharp\cap4>csc ex8cap4.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap4>ex8cap4
*****
RESULTADOS DO MÉTODO ORIGINAL
VALOR DO SALÁRIO PARA gerente, É -> 2500
*****
RESULTADOS DO MÉTODO REDEFINIDO
VALOR DO SALÁRIO PARA gerente, É -> 3000
C:\ExCSharp\cap4>
```

Figura 4.9: Executando o programa ex8cap4.exe.

Vamos analisar o código da Listagem 4.8.

Inicialmente criamos uma classe ClasseFuncionarios, na qual criamos um método chamado calcula_salario. Este método recebe um parâmetro do tipo string e retorna o valor do salário correspondente ao cargo passado como parâmetro para o método. Utilizamos a palavra virtual para informar que este método pode ser redefinido (override) em classes derivadas.

Em seguida criamos uma classe chamada HerdaDeFuncionarios. Esta classe é baseada na classe Classefuncionarios. Observe que para definir uma classe como sendo baseada em outra, utilizamos a sintaxe indicada a seguir:

```
public class HerdaDeFuncionarios:ClasseFuncionarios
```

simplesmente colocamos dois-pontos (:) após o nome da classe e o nome da classe base.

Dentro da classe HerdaDeFuncionarios utilizamos a palavra override para redefinir o método calcula_salario, conforme indicado no trecho de código a seguir:

```

public class HerdaDeFuncionarios:ClasseFuncionarios
{
    // Vamos redefinir o método para calcular o salário

    public override int calcula_salario(string cargo)
    {

        switch (cargo)
        {
            case "gerente":
                return 3000;
            case "diretor":
                return 4000;
            case "funcionario":
                return 2000;
            default:
                return 1000;
        }
    }
}

```

No programa principal criamos dois objetos. Um do tipo ClasseFuncionarios e um do tipo HerdaDeFuncionarios. Dentro da função Main chamamos o método calcula_salario das duas classes e comprovamos que realmente eles retornam valores diferentes para o cargo de gerente. Isto comprova que o método calcula_salario foi redefinido na classe herdada HerdaDeFuncionarios.

Conclusão

Iniciamos o capítulo falando de operadores e precedência de operadores. Tratamos das diversas categorias de operadores e apresentamos alguns exemplos de utilização dos mesmos.

Em seguida começamos a aprender como implementar, na prática, os conceitos de orientação a objetos com o C#. Aprendemos a criar classes. Também falamos sobre métodos construtores e destrutores.

Na seqüência tratamos da criação de métodos, dos diferentes tipos de parâmetros que podem ser passados para um método, e para finalizar tratamos do override de métodos.

No próximo capítulo finalizaremos o nosso estudo básico da linguagem C#. Veremos tópicos como o tratamento de exceções e classes e métodos para operações matemáticas e outras operações básicas no C#. A partir do Capítulo 6 iniciaremos o nosso estudo de ASP.NET.

Introdução

Neste capítulo veremos mais alguns tópicos do C#, tópicos estes que serão utilizados na construção de páginas ASP.NET nos demais capítulos do livro.

Vamos iniciar o capítulo apresentando a classe System.Math. Como o nome sugere, esta classe pertence ao namespace System. Através dos métodos desta classe temos à disposição uma série de funções para cálculos matemáticos. Temos funções para cálculos trigonométricos, logarítmicos e numéricos.

Em seguida aprenderemos a utilizar os métodos e propriedades da estrutura System.DateTime, a qual pertence ao namespace System. Com o uso da estrutura System.DateTime temos acesso a propriedades que permitem retornar apenas parte de uma hora (hora, minuto ou segundo) e parte de uma data (ano, mês, dia).

Operações como ler ou gravar em um arquivo de texto são bastante comuns. Por isso aprenderemos a realizar estas operações com o C#. Trabalharemos com as classes File e FileInfo do namespace System.IO. Vamos apresentar exemplos práticos de utilização de métodos e propriedades destas classes.

Quando acontecem erros, o programa deve ser capaz de tratar os mesmos de uma maneira “elegante”. Por exemplo, se o usuário der um comando para gravar no disquete e não existir um disquete no drive, o programa deve ser capaz de interceptar e interpretar este erro e emitir uma mensagem para que o usuário insira um disquete e repita o comando. Não seria nada elegante se o programa simplesmente fechasse devido a um erro deste tipo. Para fazer com que nossos programas sejam capazes de lidar bem com situações como a descrita anteriormente devemos fazer o tratamento de exceções. Neste capítulo vamos aprender a fazer o tratamento de exceções com o C#.

Com este capítulo encerramos o nosso estudo dos aspectos básicos da linguagem C#. Estudamos algumas das classes disponíveis no .NET Framework Class Library. Existem, sem exagero, literalmente, milhares de classes e métodos no .NET Framework Class Library. Abordamos os elementos básicos da linguagem, elementos estes que iremos utilizar na construção de páginas ASP.NET nos demais capítulos deste livro.

Procure entender os conceitos apresentados nos Capítulos de 1 a 5. Estes conceitos são fundamentais para o acompanhamento dos demais capítulos. Saber quais os componentes do Framework .NET, bem como conhecer as funções de cada um, é de fundamental importância.

CAPÍTULO 5

Tópicos Diversos em C#

A Classe System.Math – Operações Matemáticas

A classe System.Math fornece métodos para operações trigonométricas, logarítmicas e outras funções comumente utilizadas na matemática. Além dos métodos, temos duas propriedades, as quais serão descritas na seqüência.

Campos da Classe System.Math

Temos dois campos, conforme descrito a seguir.

- ◆ **E:** Este campo retorna o valor do número e, o qual é a base para os logarítmos neperianos. O valor do número e 2,7182818284590452354. O valor retornado é do tipo double.
- ◆ **PI:** Este campo retorna o valor da constante PI, cujo valor é 3.14159265358979323846. O valor retornado é do tipo double.

Vamos apresentar um exemplo que ilustra a utilização da classe System.Math e dos campos E e PI. Observe o exemplo da Listagem 5.1.

Listagem 5.1 – Campos da classe System.Math – ex1cap5.cs

```
using System;

class ex1cap5
{
    // Exemplo1 - Capítulo 5.
    // Campos da classe System.Math.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {

        // Exibição dos valores dos campos E e PI.
        Console.WriteLine("*****");
        Console.WriteLine("VALOR DO CAMPO E -> {0}", Math.E);
        Console.WriteLine("VALOR DO CAMPO PI -> {0}", Math.PI);
        Console.WriteLine("*****");

    }
}
```

Digite o exemplo da Listagem 5.1 e salve o mesmo em um arquivo chamado ex1cap5.cs, na pasta C:\ExCsharp\ cap5. Compile e execute o exemplo da Listagem 5.1. Você obterá os resultados indicados na Figura 5.1.

```
C:\ExCSharp\cap5>csc ex1cap5.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap5>ex1cap5
*****
VALOR DO CAMPO E -> 2,71828182845905
VALOR DO CAMPO PI -> 3,14159265358979
*****
C:\ExCSharp\cap5>
```

Figura 5.1: Executando o programa ex1cap5.exe.

Como a classe Math faz parte do namespace System, simplesmente utilizamos Math. E para fazer referência ao campo E da classe Math. O mesmo é válido para o campo PI.

Se utilizarmos a nomenclatura completa, conforme indicado no trecho de código a seguir, o resultado será o mesmo:

```
Console.WriteLine("VALOR DO CAMPO E -> {0}", System.Math.E);
Console.WriteLine("VALOR DO CAMPO PI -> {0}", System.Math.PI);
```

Métodos da Classe System.Math

Neste tópico trataremos dos diversos métodos da classe System.Math.

- ◆ **Método Abs** – System.Math.Abs(número): Este método retorna o valor absoluto do número, ou seja, o módulo do número passado como parâmetro. O número passado como parâmetro pode ser de um dos seguintes tipos: Decimal, Double, Int16, Int32, Int64, SByte ou Single.

Ex:

```
System.Math.Abs (-32.15) retorna 32,15.
```

- ◆ **Método Acos** – System.Math.Acos(número): Este método retorna o valor do ângulo em que o coseno é o número passado como parâmetro. O valor passado como parâmetro deve estar no intervalo de -1 até +1, que são os valores válidos para o coseno. O valor do ângulo retornado está em radianos. Em termos de radianos, 3,14 radianos equivalem a 180 graus. Para converter de radiano para graus é só utilizar a seguinte fórmula: valor_em_graus = (valor_em_radianos*180)/PI

Ex:

```
System.Math.Acos(-1)
System.Math.Acos(-0.5)
System.Math.Acos(0)
System.Math.Acos(0.5)
System.Math.Acos(1)
```

Vamos a um exemplo completo no qual utilizamos os métodos Abs e Acos. Considere o exemplo da Listagem 5.2

Listagem 5.2 – Métodos Abs e Acos da classe Math – ex2cap5.cs

```
using System;

class ex2cap5
{
    // Exemplo2 - Capítulo 5.
    // Métodos da classe System.Math.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {
        // Exibição dos valores dos campos E e PI.

        Console.WriteLine("*****");
        Console.WriteLine("VALOR ABSOLUTO DE -32.10 -> {0}", Math.Abs(-32.10));

        Console.WriteLine("*****");
        Console.WriteLine("UTILIZAÇÃO DO MÉTODO ACOS");

        Console.WriteLine("*****");
        Console.WriteLine("ÂNGULO PARA COS = -1      -> {0}", Math.Acos(-1));
        Console.WriteLine("ÂNGULO PARA COS = -0,5     -> {0}", Math.Acos(-0.5));
        Console.WriteLine("ÂNGULO PARA COS = 0       -> {0}", Math.Acos(0));
        Console.WriteLine("ÂNGULO PARA COS = 0,5     -> {0}", Math.Acos(0.5));
        Console.WriteLine("ÂNGULO PARA COS = 1       -> {0}", Math.Acos(1));
    }
}
```

```
Console.WriteLine("*****");
Console.WriteLine("*****");
Console.WriteLine("CONVERSÃO DOS ÂNGULOS PARA GRAUS");
Console.WriteLine("*****");
Console.WriteLine("ÂNGULO PARA COS = -1      -> {0}", ((Math.Acos(-1)*180)/Math.PI));
Console.WriteLine("ÂNGULO PARA COS = -0,5   -> {0}", ((Math.Acos(-0.5)*180)/Math.PI));
Console.WriteLine("ÂNGULO PARA COS = 0       -> {0}", ((Math.Acos(0)*180)/Math.PI));
Console.WriteLine("ÂNGULO PARA COS = 0,5    -> {0}", ((Math.Acos(0.5)*180)/Math.PI));
Console.WriteLine("ÂNGULO PARA COS = 1      -> {0}", ((Math.Acos(1)*180)/Math.PI));
Console.WriteLine("*****");
}

}
```

Neste exemplo, primeiro exibimos os valores dos ângulos em radianos, depois utilizamos a fórmula para conversão de radianos para graus, que é simplesmente multiplicar por 180 e dividir por PI. Observe que, para obter o valor do PI, utilizamos o campo PI da classe System.Math.

Digite o exemplo da Listagem 5.2 e salve o mesmo em um arquivo chamado ex2cap5.cs, na pasta C:\ExCsharp\cap5. Compile e execute o exemplo da Listagem 5.2. Você obterá os resultados indicados na Figura 5.2.

◆ **Método Asin – System.Math.Asin(número):** Este método retorna o valor do ângulo em que o seno é o número passado como parâmetro. O valor passado como parâmetro deve estar no intervalo de -1 até +1, que são os valores válidos para o seno. O valor do ângulo retornado está em radianos. Em termos de radianos, 3,14 radianos equivalem a 180 graus. Para converter de radiano para graus é só utilizar a seguinte fórmula: valor_em_graus = (valor_em_radianos*180)/PI

Ex:

```
System.Math.Asin(-1)
System.Math.Asin(-0.5)
System.Math.Asin(0)
System.Math.Asin(0.5)
System.Math.Asin(1)
```

```
C:\ExCSharp\cap5>csc ex2cap5.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap5>ex2cap5
*****
VALOR ABSOLUTO DE -32.10 -> 32,1
*****
UTILIZAÇÃO DO MÉTODO ACOS
*****
ÂNGULO PARA COS = -1      -> 3,14159265358979
ÂNGULO PARA COS = -0,5    -> 2,0943951023932
ÂNGULO PARA COS = 0       -> 1,5707963267949
ÂNGULO PARA COS = 0,5     -> 1,0471975511966
ÂNGULO PARA COS = 1       -> 0
*****
CONVERSÃO DOS ÂNGULOS PARA GRAUS
*****
ÂNGULO PARA COS = -1      -> 180
ÂNGULO PARA COS = -0,5    -> 120
ÂNGULO PARA COS = 0       -> 90
ÂNGULO PARA COS = 0,5     -> 60
ÂNGULO PARA COS = 1       -> 0
*****

C:\ExCSharp\cap5>_
```

Figura 5.2: Executando o programa ex2cap5.exe.

Se no exemplo da Listagem 5.2, simplesmente trocássemos Acos por Asin, obteríamos os resultados indicados na Figura 5.3.

```
C:\ExCSharp\cap5>csc ex2cap5.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap5>ex2cap5
*****
VALOR ABSOLUTO DE -32.10 -> 32,1
*****
UTILIZAÇÃO DO MÉTODO Asin
*****
ÂNGULO PARA SIN = -1      -> -1,5707963267949
ÂNGULO PARA SIN = -0,5    -> -0,523598775598299
ÂNGULO PARA SIN = 0       -> 0
ÂNGULO PARA SIN = 0,5     -> 0,523598775598299
ÂNGULO PARA SIN = 1       -> 1,5707963267949
*****
CONVERSÃO DOS ÂNGULOS PARA GRAUS
*****
ÂNGULO PARA SIN = -1      -> -90
ÂNGULO PARA SIN = -0,5    -> -30
ÂNGULO PARA SIN = 0       -> 0
ÂNGULO PARA SIN = 0,5     -> 30
ÂNGULO PARA SIN = 1       -> 90
*****

C:\ExCSharp\cap5>_
```

Figura 5.3: Executando o programa ex2cap5.exe, após trocar Acos por Asin.

- ◆ Método Atan – System.Math.Atan(número): Este método retorna o valor do ângulo em que a tangente é o número passado como parâmetro. O valor do ângulo retornado está em radianos.

Ex:

```
System.Math.Atan(0)
```

```
System.Math.Atan(2)
```

- ◆ Método Atan2 – System.Math.Atan2(número,número): Este método retorna o valor do ângulo em que a tangente é igual à divisão dos dois números passados como parâmetros. O valor do ângulo retornado está em radianos.

Ex:

```
System.Math.Atan2(6,2) retorna 1,24904577239825 radiano  
System.Math.Atan2(4,2) retorna 1,10714871779409 radiano
```

- ◆ Método Ceiling – System.Math.Ceiling(número): Este método retorna o menor número inteiro que é igual ou maior do que o número passado como parâmetro. O valor do parâmetro é do tipo double.

Ex:

```
System.Math.Ceiling(2.3498) retorna 3  
System.Math.Ceiling(-2.3498) retorna -2 (lembre que -2 é maior do que -3)  
System.Math.Ceiling(5.0001) retorna 6  
System.Math.Ceiling(-1.9999) retorna -1 (lembre que -1 é maior do que -2)
```

- ◆ Método Cos – System.Math.Cos(valor em radianos): Este método retorna o cosseno do valor que foi passado como parâmetro. O valor do parâmetro deve estar em radianos. Por exemplo, ao invés de 180 graus, informamos 3,14 radianos (PI radianos).

Nos exemplos a seguir utilizamos o campo PI para passar os valores em radianos para o método Cos.

Ex:

```
System.Math.Cos(Math.PI)      retorna -1  
System.Math.Cos(0)           retorna 1
```

- ◆ Método Exp – System.Math.Exp(número): Este método retorna o número e (Math.E) elevado no expoente passado como parâmetro.

Ex:

```
System.Math.Exp(2)      retorna 7,38905609893065  
System.Math.Exp(0)      retorna 1
```

- ◆ Método Floor – System.Math.Floor(número): Este método retorna o maior número inteiro que é igual ou menor do que o número passado como parâmetro. O valor do parâmetro é do tipo double.

Ex:

```
System.Math.Floor(2.3498)  retorna 2  
System.Math.Floor(-2.3498) retorna -3 (lembre que -3 é menor do que -2)  
System.Math.Floor(5.0001)  retorna 5  
System.Math.Floor(-1.9999) retorna -2 (lembre que -2 é menor do que -1)
```

- ◆ Método IEEERemainder – System.Math.IEEERemainder(x,y): Este método retorna o resto da divisão de x por y.

Ex:

```
System.Math.IEEERemainder(10,4) retorna 2  
System.Math.IEEERemainder(250,6)  retorna 4  
System.Math.IEEERemainder(10,5)  retorna 0
```

- ◆ Método Log – System.Math.Log(número) ou System.Math.Log(número,base): Se o método Log receber um único parâmetro, irá retornar o logaritmo natural (na base e) do número passado como parâmetro. Também podemos passar dois parâmetros, neste caso o primeiro parâmetro é o número e o segundo, a base.

Ex:

```
System.Math. Log(10)      retorna 2,30258509299405
System.Math. Log(100,10)   retorna 2
System.Math. Log(16,2)     retorna 4
```

- ◆ Método Log10 – System.Math.Log10(número). O método Log retorna o logaritmo na base 10, do número passado como parâmetro.

Ex:

```
System.Math. Log10(10)    retorna 1
System.Math. Log10(100)   retorna 2
System.Math. Log10(1000)  retorna 3
```

- ◆ Método Max – System.Math.Max(número,número): Este método retorna o maior valor dentre dois valores passados como parâmetros. Os dois valores devem ser do mesmo tipo. Os tipos permitidos são os seguintes: Byte, Decimal, Double, Int16, Int32, Int64, SByte, Single, UInt16, UInt32 ou UInt64.

Ex:

```
System.Math.Max(-5,-6)  retorna -5
System.Math.Max(5,6)    retorna 6
```

- ◆ Método Min – System.Math.Min(número,número): Este método retorna o menor valor dentre dois valores passados como parâmetros. Os dois valores devem ser do mesmo tipo. Os tipos permitidos são os seguintes: Byte, Decimal, Double, Int16, Int32, Int64, SByte, Single, UInt16, UInt32 ou UInt64.

Ex:

```
System.Math.Min(-5,-6)  retorna -6
System.Math.Min(5,6)    retorna 5
```

- ◆ Método Pow – System.Math.Pow(x,y): Este método retorna o número x elevado no expoente y.

Ex:

```
System.Math.Pow(2,2)     retorna 4
System.Math.Pow(2,3)     retorna 8
System.Math.Pow(2,4)     retorna 16
System.Math.Pow(5,2)     retorna 25
```

- ◆ Método Round – System.Math.Round(número) Ou System.Math.Round (número,precisão): Este método faz o arredondamento para o inteiro mais próximo ou para um número de casas decimais especificado. O parâmetro número pode ser do tipo Double ou do tipo Decimal. O parâmetro precisão, se especificado, deve ser do tipo Int32 e define o número de casas decimais.

Ex:

```
System.Math.Round(2.49)    retorna 2
System.Math.Round(2,4599)   retorna 2
System.Math.Round(10.4987564,3) retorna 10,499
```

- ◆ Método Sin – System.Math.Sin(valor em radianos): Este método retorna o seno do valor que foi passado como parâmetro. O valor do parâmetro deve estar em radianos. Por exemplo, ao invés de 180 graus, informamos 3,14 radianos (PI radianos).

Nos exemplos a seguir utilizamos o campo PI para passar os valores em radianos para o método Sin.

Ex:

```
System.Math.Sin(Math.PI/2)    retorna 1  
System.Math.Sin(0)           retorna 0
```

- ◆ Método Sign – System.Math.Sign(número): Este método retorna um valor indicando o sinal do número passado como parâmetro. O número passado como parâmetro pode ser de um dos seguintes tipos: Decimal, Double, Int16, Int32, Int64, SByte ou Single.

Ex:

```
System.Math.Sign(-32.15)     retorna -1  
System.Math.Sign(32.15)      retorna 1
```

- ◆ Método Sqrt – System.Math.Sqrt(número): Este método retorna a raiz quadrada do número passado como parâmetro.

Ex:

```
System.Math.Sqrt(25)         retorna 5  
System.Math.Sqrt(144)        retorna 12
```

- ◆ Método Tan – System.Math.Tan(valor em radianos): Este método retorna a tangente do valor que foi passado como parâmetro. O valor do parâmetro deve estar em radianos. Por exemplo, ao invés de 180 graus informamos 3,14 radianos (PI radianos).

IMPORTANTE: Não existe a tangente para valores onde o cosseno vale 0. Para 90 (PI/2) e 270 graus (3PI/2) não existe o valor da tangente.

Nos exemplos a seguir utilizamos o campo PI para passar os valores em radianos para o método Tan.

Ex:

```
System.Math.Tan(Math.PI/3)    retorna 1,73205080756888  
System.Math.Tan(0)           retorna 0
```

Vamos a um exemplo completo no qual utilizamos todos os métodos matemáticos apresentados neste item. Considere o exemplo da Listagem 5.3

Listagem 5.3 – Métodos da classe Math – ex3cap5.cs

```
using System;  
  
class ex3cap5  
{  
    // Exemplo 3 - Capítulo 5.  
    // Campos e Métodos da classe System.Math.  
    // Por: Júlio Battisti  
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA
```

```

public static void Main()
{
    // Exibição dos valores dos campos E e PI.

    Console.WriteLine("*****");
    Console.WriteLine("VALOR ABSOLUTO DE -32.10 -> {0}", Math.Abs(-32.10));

    Console.WriteLine("*****");
    Console.WriteLine("UTILIZAÇÃO DO MÉTODO Asin");

    Console.WriteLine("*****");
    Console.WriteLine(<<ÂNGULO PARA SIN = -1      -> {0}>, Math.Asin(-1));
    Console.WriteLine(<<ÂNGULO PARA SIN = -0,5   -> {0}>, Math.Asin(-0.5));
    Console.WriteLine(<<ÂNGULO PARA SIN = 0       -> {0}>, Math.Asin(0));
    Console.WriteLine(<<ÂNGULO PARA SIN = 0,5     -> {0}>, Math.Asin(0.5));
    Console.WriteLine(<<ÂNGULO PARA SIN = 1       -> {0}>, Math.Asin(1));
    Console.WriteLine(<<*****>>);

    Console.WriteLine("*****");
    Console.WriteLine("CONVERSÃO DOS ÂNGULOS PARA GRAUS");

    Console.WriteLine("*****");
    Console.WriteLine(<<ÂNGULO PARA SIN = -1      -> {0}>, ((Math.Asin(-1)*180)/Math.PI));
    Console.WriteLine(<<ÂNGULO PARA SIN = -0,5   -> {0}>, ((Math.Asin(-0.5)*180)/
    Math.PI));
    Console.WriteLine(<<ÂNGULO PARA SIN = 0       -> {0}>, ((Math.Asin(0)*180)/Math.PI));
    Console.WriteLine(<<ÂNGULO PARA SIN = 0,5     -> {0}>, ((Math.Asin(0.5)*180)/Math.PI));
    Console.WriteLine(<<ÂNGULO PARA SIN = 1       -> {0}>, ((Math.Asin(1)*180)/Math.PI));

    Console.WriteLine(<<*****>>);
    Console.WriteLine(<<ÂNGULO PARA TAN = 3      -> {0}>, Math.Atan2(6,2));
    Console.WriteLine(<<ÂNGULO PARA TAN = 2      -> {0}>, Math.Atan2(4,2));
    Console.WriteLine(<<*****>>);

    Console.WriteLine("ARREDONDA UTILIZ. Ceiling -> {0}", Math.Ceiling(2.3498));
    Console.WriteLine("ARREDONDA UTILIZ. Ceiling -> {0}", Math.Ceiling(-2.3498));
    Console.WriteLine("ARREDONDA UTILIZ. Ceiling -> {0}", Math.Ceiling(5.0001));
}

```

```
Console.WriteLine("ARREDONDA UTILIZ. Ceiling -> {0}",Math.Ceiling(-1.9999));  
  
Console.WriteLine("*****");  
Console.WriteLine("COSENO PARA 180 graus PI-> {0}",Math.Cos(Math.PI));  
Console.WriteLine("COSENO PARA 0 graus -> {0}",Math.Cos(0));  
  
Console.WriteLine("*****");  
Console.WriteLine("e elevado ao quadrado-> {0}",Math.Exp(2));  
Console.WriteLine("e elevado na zero -> {0}",Math.Exp(0));  
  
Console.WriteLine("*****");  
Console.WriteLine("ARREDONDA UTILIZ. Floor -> {0}",Math.Floor(2.3498));  
Console.WriteLine("ARREDONDA UTILIZ. Floor -> {0}",Math.Floor(-2.3498));  
Console.WriteLine("ARREDONDA UTILIZ. Floor -> {0}",Math.Floor(5.0001));  
Console.WriteLine("ARREDONDA UTILIZ. Floor -> {0}",Math.Floor(-1.9999));  
  
Console.WriteLine("*****");  
Console.WriteLine("RESTO DA DIVISÃO 10/4 -> {0}",Math.IEEEremainder(10,4));  
Console.WriteLine("RESTO DA DIVISÃO 250/6 -> {0}",Math.IEEEremainder(250,6));  
Console.WriteLine("RESTO DA DIVISÃO 10/5 -> {0}",Math.IEEEremainder(10,5));  
  
Console.WriteLine("*****");  
Console.WriteLine("LOGARITMO NATURAL DE 10 -> {0}",Math.Log(10));  
Console.WriteLine("LOG DE 100 NA BASE 10 -> {0}",Math.Log(100,10));  
Console.WriteLine("LOG DE 16 NA BASE 2 -> {0}",Math.Log(16,2));  
  
Console.WriteLine("*****");  
Console.WriteLine("LOGARITMO DE 10 -> {0}",Math.Log10(10));  
Console.WriteLine("LOGARITMO DE 100 -> {0}",Math.Log10(100));  
Console.WriteLine("LOGARITMO DE 1000 -> {0}",Math.Log10(1000));  
  
Console.WriteLine("*****");  
Console.WriteLine("MÁXIMO ENTRE -5 E -6 -> {0}",Math.Max(-5,-6));  
Console.WriteLine("MÁXIMO ENTRE 5 E 6 -> {0}",Math.Max(5,6));  
Console.WriteLine("*****");
```

```

Console.WriteLine("MÍNIMO ENTRE -5 E -6      -> {0}",Math.Min(-5,-6));
Console.WriteLine("MÍNIMO ENTRE  5 E  6      -> {0}",Math.Min(5,6));

Console.WriteLine("*****");
Console.WriteLine("2 ELEVADO NA 2      -> {0}",Math.Pow(2,2));
Console.WriteLine("2 ELEVADO NA 3      -> {0}",Math.Pow(2,3));
Console.WriteLine("2 ELEVADO NA 4      -> {0}",Math.Pow(2,4));
Console.WriteLine("5 ELEVADO NA 2      -> {0}",Math.Pow(5,2));

Console.WriteLine("*****");
Console.WriteLine("ROUND EM 2,49        -> {0}",Math.Round(2.49));
Console.WriteLine("ROUND EM 2,4599       -> {0}",Math.Round(2.4599));
Console.WriteLine("10,4987564 COM 3 CASAS -> {0}",Math.Round(10.4987564,3));

Console.WriteLine("*****");
Console.WriteLine("SENO    PARA 90 graus PI/2-> {0}",Math.Sin(Math.PI/2));
Console.WriteLine("SENO    PARA 0  graus      -> {0}",Math.Sin(0));

Console.WriteLine("*****");
Console.WriteLine("SINAL PARA -32,15      -> {0}",Math.Sign(-32.15));
Console.WriteLine("SINAL PARA  32,15      -> {0}",Math.Sign(32.15));

Console.WriteLine("*****");
Console.WriteLine("RAIZ QUADRADA DE 25      -> {0}",Math.Sqrt(25));
Console.WriteLine("RAIZ QUADRADA DE 144      -> {0}",Math.Sqrt(144));

Console.WriteLine("*****");
Console.WriteLine("TANENTE PARA 60  graus PI-> {0}",Math.Tan(Math.PI/3));
Console.WriteLine("TANENTE PARA    0 graus   -> {0}",Math.Tan(0));

}
}

```

Digite o exemplo da Listagem 5.3 e salve o mesmo em um arquivo chamado ex3cap5.cs, na pasta C:\ExCsharp\ cap5. Compile e execute o exemplo da Listagem 5.3. Obteremos os resultados indicados a seguir:

SAÍDA DO PROGRAMA ex3cap5.exe:

```
*****
```

VALOR ABSOLUTO DE -32.10 -> 32,1

```
*****
```

UTILIZAÇÃO DO MÉTODO Asin

```
*****
```

ÂNGULO PARA SIN = -1 -> -1,5707963267949

ÂNGULO PARA SIN = -0,5 -> -0,523598775598299

ÂNGULO PARA SIN = 0 -> 0

ÂNGULO PARA SIN = 0.5 -> 0,523598775598299

ÂNGULO PARA SIN = 1 -> 1,5707963267949

```
*****
```

CONVERSÃO DOS ÂNGULOS PARA GRAUS

```
*****
```

ÂNGULO PARA SIN = -1 -> -90

ÂNGULO PARA SIN = -0,5 -> -30

ÂNGULO PARA SIN = 0 -> 0

ÂNGULO PARA SIN = 0.5 -> 30

ÂNGULO PARA SIN = 1 -> 90

```
*****
```

ÂNGULO PARA TAN = 3 -> 1,24904577239825

ÂNGULO PARA TAN = 2 -> 1,10714871779409

```
*****
```

ARREDONDA UTILIZ. Ceiling -> 3

ARREDONDA UTILIZ. Ceiling -> -2

ARREDONDA UTILIZ. Ceiling -> 6

ARREDONDA UTILIZ. Ceiling -> -1

```
*****
```

COSENO PARA 180 graus PI-> -1

COSENO PARA 0 graus -> 1

```
*****
```

```
*****
```

e elevado ao quadrado -> 7,38905609893065

e elevado na zero -> 1

```
*****
```

```

ARREDONDA UTILIZ. Floor -> 2
ARREDONDA UTILIZ. Floor -> -3
ARREDONDA UTILIZ. Floor -> 5
ARREDONDA UTILIZ. Floor -> -2
*****
RESTO DA DIVISÃO 10/4 -> 2
RESTO DA DIVISÃO 250/6 -> 4
RESTO DA DIVISÃO 10/5 -> 0
*****
LOGARITMO NATURAL DE 10 -> 2,30258509299405
LOG DE 100 NA BASE 10 -> 2
LOG DE 16 NA BASE 2 -> 4
*****
LOGARITMO DE 10 -> 1
LOGARITMO DE 100 -> 2
LOGARITMO DE 1000 -> 3
*****
MÁXIMO ENTRE -5 E -6 -> -5
MÁXIMO ENTRE 5 E 6 -> 6
*****
MÍNIMO ENTRE -5 E -6 -> -6
MÍNIMO ENTRE 5 E 6 -> 5
*****
2 ELEVADO NA 2 -> 4
2 ELEVADO NA 3 -> 8
2 ELEVADO NA 4 -> 16
5 ELEVADO NA 2 -> 25
*****
ROUND EM 2,49 -> 2
ROUND EM 2,4599 -> 2
10,4987564 COM 3 CASAS -> 10,499
*****
SENO PARA 90 graus PI/2-> 1
SENO PARA 0 graus -> 0
*****
SINAL PARA -32,15 -> -1

```

```
SINAL PARA 32,15      -> 1
*****
RAIZ QUADRADA DE 25      -> 5
RAIZ QUADRADA DE 144      -> 12
*****
TANENTE PARA 60 graus PI/3 -> 1,73205080756888
TANENTE PARA 0 graus       -> 0
TANENTE PARA 180 graus PI -> 1,63317787283838E+16
```

A Estrutura System.DateTime – Datas e Horas

No namespace System temos uma estrutura chamada System.DateTime. Para realizar operações com valores de data e hora utilizamos os campos, propriedades e métodos desta estrutura. A estrutura System.DateTime é bastante flexível. Só para se ter uma idéia, existem sete construtores diferentes para esta estrutura.

Vamos apresentar um exemplo onde utilizamos os principais construtores da estrutura System.DateTime. Considere o exemplo da Listagem 5.4

Listagem 5.4 – Construtores de System.DateTime – ex4cap5.cs

```
using System;

class ex4cap5
{
    // Exemplo 4 - Capítulo 5.
    // Construtores de System.DateTime.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {

        // Utilização de diversos construtores de System.DateTime.

        DateTime constrex1 = new DateTime(2001,7,21);
        DateTime constrex2 = new DateTime(0);
        DateTime constrex3 = new DateTime(2001,7,21,16,30,45);
```

```

//Exibição dos resultados

Console.WriteLine("*****");
Console.WriteLine("DATA E HORA ATUAIS      -> {0}",DateTime.Now);
Console.WriteLine("CONSTRUTOR -dia,mês,ano   -> {0}",constrex1);
Console.WriteLine("CONSTRUTOR -milisegundos  -> {0}",constrex2);
Console.WriteLine("CONSTRUTOR -d,m,a,h,min,seg-> {0}",constrex3);
Console.WriteLine("*****");

}

}

```

Digite o exemplo da Listagem 5.4 e salve o mesmo em um arquivo chamado ex4cap5.cs, na pasta C:\ExCsharp\cap5. Compile e execute o exemplo da Listagem 5.4. Você obterá os resultados indicados na Figura 5.4.

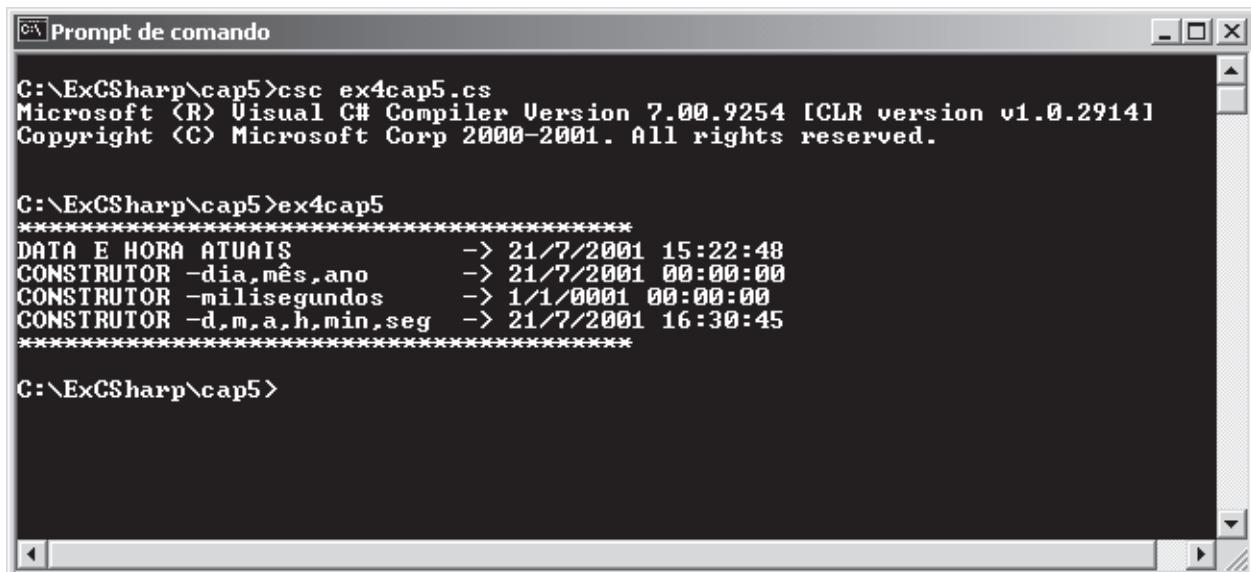


Figura 5.4: Executando o programa ex4cap5.exe.

Neste exemplo utilizamos os três construtores mais utilizados. Observe o código a seguir:

```
DateTime constrex1 = new DateTime(2001,7,21);
```

Neste exemplo estamos utilizando um construtor no qual são passados três valores: ano, mês e dia. Os valores para hora, minuto e segundo são inicializados com zero, conforme pode ser confirmado na Figura 5.4.

```
DateTime constrex2 = new DateTime(0);
```

Neste exemplo utilizamos um construtor no qual passamos, com único parâmetro, o número de milisegundos a partir da menor data suportada pelo Framework .NET. Como passamos um valor zero, o mesmo irá exibir a menor data possível, que, conforme podemos confirmar pela saída do programa, é: 1/1/0001 00:00:00.

```
DateTime constrex3 = new DateTime(2001,7,21,16,30,45);
```

Neste exemplo utilizamos um construtor no qual são passados seis valores: ano, mês, dia, hora, minuto e segundo.

Campos da Estrutura System.DateTime

A estrutura DateTime tem dois campos:

- ◆ **DateTime.MaxValue:** Este campo retorna o valor da maior data/hora suportada pela estrutura DateTime. O valor retornado é: 31/12/9999 23:59:59, para o sistema de Data/Hora do Brasil.
- ◆ **DateTime.MinValue:** Este campo retorna o valor da menor data/hora suportada pela estrutura DateTime. O valor retornado é: 1/1/0001 00:00:00, para o sistema de Data/Hora do Brasil.

Propriedades da Estrutura System.DateTime

Na Tabela 5.1 temos a descrição das propriedades da estrutura System.DateTime.

Tabela 5.1 Propriedades da estrutura System.DateTime

Propriedade	Descrição
Date	Retorna apenas a parte referente à data.
Day	Retorna apenas o dia.
DayOfWeek	Retorna um número que indica o dia da semana. O número varia de zero para o Domingo, 1 para a segunda-feira, 2 para a terça-feira e assim por diante, até 6 para o sábado.
DayOfYear	Retorna um número que indica o dia do ano. Retorna 1 para 1º de Janeiro, 2 para 02 de Janeiro e assim por diante, até 365 (ou 366 se for ano bissexto) para 31 de Dezembro.
Hour	Retorna apenas a hora.
Minute	Retorna apenas o minuto.
Month	Retorna apenas o mês.
Now	Retorna a Data e a Hora do sistema.
Second	Retorna os segundos.
Year	Retorna apenas o Ano.

Vamos apresentar um exemplo no qual utilizamos as propriedades da estrutura System.DateTime.

Considere o exemplo da Listagem 5.5

Listagem 5.5 – Propriedades de System.DateTime – ex5cap5.cs

```
using System;

class ex5cap5
{
    // Exemplo 5 - Capítulo 5.
    // Propriedades de System.DateTime.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {

        // Utilização de diversos construtores de System.DateTime.

        DateTime constrex1 = DateTime.Now;

        //Exibição dos resultados

        Console.WriteLine("*****");
        Console.WriteLine("HORA DO SISTEMA      -> {0}", constrex1.Hour);
        Console.WriteLine("MINUTO DO SISTEMA           -> {0}", constrex1.Minute);
        Console.WriteLine("SEGUNDOS DO SISTEMA       -> {0}", constrex1.Second);
        Console.WriteLine("ANO DO SISTEMA-> {0}", constrex1.Year);
        Console.WriteLine("MÊS DO SISTEMA-> {0}", constrex1.Month);
        Console.WriteLine("DIA DO SISTEMA-> {0}", constrex1.Day);
        Console.WriteLine("DATA COMPLETA          -> {0}", constrex1.Date);
        Console.WriteLine("DIA DA SEMANA            -> {0}", constrex1.DayOfWeek);
        Console.WriteLine("DIA DO ANO               ->
{0}", constrex1.DayOfYear);

    }
}
```

Digite o exemplo da Listagem 5.5 e salve o mesmo em um arquivo chamado ex5cap5.cs, na pasta C:\ExCsharp\cap5. Compile e execute o exemplo da Listagem 5.5. Você obterá os resultados indicados na Figura 5.5.

```
C:\ExCSharp\cap5>csc ex5cap5.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap5>ex5cap5
*****
HORA DO SISTEMA      -> 16
MINUTO DO SISTEMA    -> 5
SEGUNDOS DO SISTEMA  -> 40
ANO DO SISTEMA        -> 2001
MÊS DO SISTEMA        -> ?
DIA DO SISTEMA        -> 21
DATA COMPLETA DO SISTEMA -> 21/7/2001 00:00:00
DIA DA SEMANA         -> Saturday
DIA DO ANO            -> 202

C:\ExCSharp\cap5>
```

Figura 5.5: Executando o programa ex5cap5.exe.

O Namespace System.IO – Operações com Arquivos e Pastas

Na versão do ASP – Active Server Pages, que acompanha o Windows 2000 – ASP 3.0, temos um objeto chamado FileSystemObject. Com o uso deste objeto temos acesso ao sistema de arquivos do servidor. Existem diversos métodos que possibilitam obter informações sobre drives, pastas e arquivos. O Framework .NET fornece o namespace System.IO, no qual encontramos diversas classes para operações com drives, pastas e arquivos.

Nesta seção apresentaremos alguns métodos e propriedades das classes System.IO.File e System.IO.FileInfo. Nos demais capítulos deste livro vamos utilizar novamente estas classes, além da classe System.IO.Directory.

A Classe System.IO.File – Operações com Arquivos

A classe System.IO.File fornece métodos para criar, copiar, mover e abrir arquivos. Quando criamos um novo arquivo, utilizando o método para criação de arquivos da classe File, são definidas permissões de acesso total para todos os usuários, por padrão. É o famoso Everyone -> Full Control.

Alguns métodos recebem caminhos como parâmetros. Existem várias maneiras de especificar um caminho. Pode ser o caminho completo: C:\\Documentos\\relatorio.doc. Pode ser o caminho para uma pasta: C:\\Documentos. Pode ser um caminho relativo, em relação ao diretório atual: Documentos\\Fiscais\\2001. Pode ser um caminho de rede no formato UNC (Universal Naming Convention). Por exemplo, para usar o formato UNC para acessar um arquivo chamado vendas.doc, que está em uma pasta compartilhada chamada Vendas em um servidor chamado vendassrv,

devemos utilizar o seguinte caminho: \\vendassrv\ vendas\ vendas.doc. Todo caminho UNC inicia com duas barras, porém a barra invertida é considerada um caractere especial e todo caractere especial deve ser precedido de uma barra invertida. O mesmo comentário é válido para os casos anteriores, onde ao invés de uma estamos utilizando duas barras. Por isso cada barra invertida é representada duas vezes.

Métodos da Classe System.IO.File

Na Tabela 5.2 temos a descrição dos principais métodos da classe System.IO.File.

Tabela 5.2 Principais métodos da classe System.IO.File.

Método	Descrição
AppendText	Anexa texto a um arquivo já existente ou cria um novo arquivo se ainda não existir o arquivo passado como parâmetro.
Copy	Efetua a cópia de um arquivo para uma pasta diferente ou para a mesma pasta com um nome diferente. Podemos especificar se o arquivo deve ser substituído caso o mesmo já exista.
Create	Cria um arquivo em uma pasta especificada como parâmetro.
Delete	Exclui um arquivo passado como parâmetro. Caso o arquivo não exista, será gerada uma exceção.
Exists	Determina se o arquivo passado como parâmetro existe.
GetAttributes	Obtém os atributos do arquivo passado como parâmetro.
GetCreationTime	Retorna a data e a hora em que o arquivo foi criado.
GetLastAccessTime	Retorna a data e a hora em que o arquivo foi acessado pela última vez.
GetLastWriteTime	Retorna a data e a hora em que foram gravadas informações no arquivo, pela última vez.
Move	Move o arquivo para uma nova localização.
SetAttributes	Define os atributos de um arquivo.
SetCreationTime	Define a informação sobre a hora e a data em que o arquivo foi criado.
SetLastAccessTime	Define a informação sobre a última data e hora em que o arquivo foi acessado.
SetLastWriteTime	Define a informação sobre a última data e hora em que foram feitas gravações no arquivo.

Vamos apresentar um exemplo no qual utilizamos alguns métodos da classe System.IO.File. Neste exemplo vamos criar um arquivo chamado teste.txt. Este arquivo será criado na pasta C:\. Depois utilizaremos alguns métodos da classe System.IO.File.

Considere o exemplo da Listagem 5.6.

Listagem 5.6 – Métodos da classe System.IO.File – ex6cap5.cs

```
using System;
using System.IO;

class ex6cap5
{
    // Exemplo 6 - Capítulo 5.
    // Métodos de System.IO.File
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {

        // Vamos copiar o arquivo C:\autoexec.bat
        // para C:\teste.bat

        File.Copy("C:\\\\autoexec.bat","C:\\\\teste.bat");

        // Agora passaremos a utilizar alguns métodos.

        // Vamos testar se o arquivo teste.bat foi criado.

        Console.WriteLine("Existe teste.bat ? {0}",File.Exists("C:\\\\teste.bat"));

        // Exibe os atributos do arquivo teste.bat
        Console.WriteLine("Atributos de teste.bat -> {0}",File.GetAttributes("C:\\\\teste.bat"));

        // Exibe a data e hora de criação do arquivo teste.bat
        Console.WriteLine("teste.bat foi criado em -> {0}",File.GetCreationTime("C:\\\\teste.bat"));

        // Exibe a data e hora do último acesso ao arquivo teste.bat
        Console.WriteLine("teste.bat foi acessado em -> {0}",File.GetLastAccessTime("C:\\\\teste.bat"));

        // Exibe a data e hora da última gravação no arquivo teste.bat
```

```

Console.WriteLine("teste.bat foi gravado em ->
{0}", File.GetLastWriteTime("C:\\teste.bat"));

// Deleto o arquivo teste.bat

File.Delete("C:\\\\teste.bat");

}

}

```

Digite o exemplo da Listagem 5.6 e salve o mesmo em um arquivo chamado ex6cap5.cs, na pasta C:\ExCsharp\cap5. Compile e execute o exemplo da Listagem 5.6. Você obterá os resultados indicados na Figura 5.6.

```

C:\ExCSharp\cap5>csc ex6cap5.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap5>ex6cap5
Existe teste.bat ? True
Atributos de teste.bat -> Hidden, Archive
teste.bat foi criado em -> 21/7/2001 17:52:39
teste.bat foi acessado em -> 21/7/2001 00:00:00
teste.bat foi gravado em -> 12/7/2001 00:01:54

C:\ExCSharp\cap5>_

```

Figura 5.6: Executando o programa ex6cap5.exe.

IMPORTANTE: Se o arquivo C:\teste.bat já existir, será gerado o erro de execução indicado nas Figuras 5.7 e 5.8.

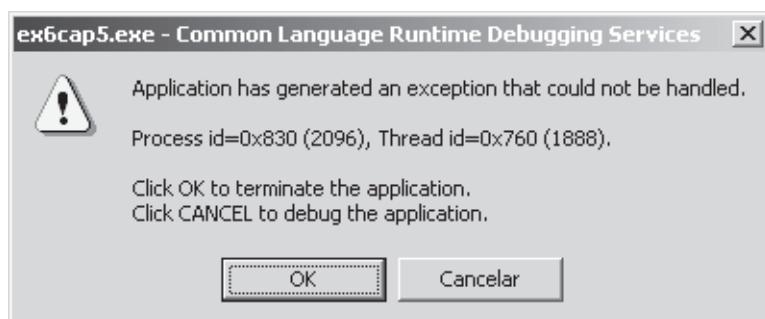


Figura 5.7: Erro gerado se o arquivo C:\teste.bat já existir.

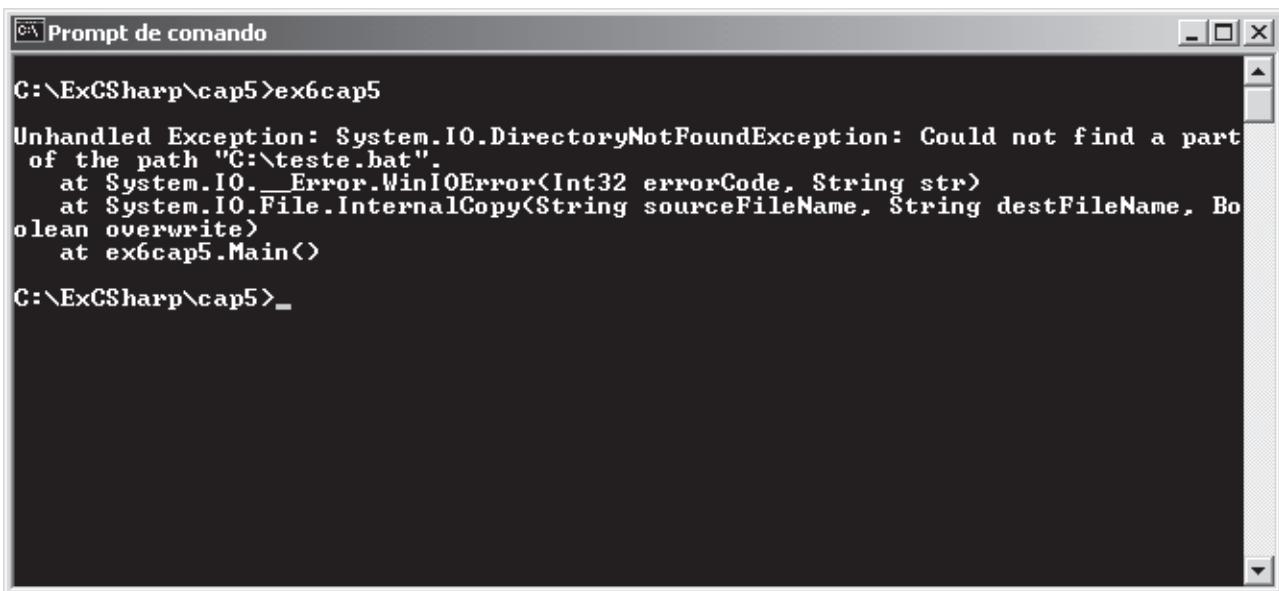


Figura 5.8: Erro gerado se o arquivo C:\teste.bat já existir.

Também é importante salientar que, para termos acesso à classe File, fizemos referência ao namespace System.IO no início do programa:

```
using System.IO;
```

Se não tivéssemos feito esta referência, ao invés de utilizar simplesmente File, teríamos que utilizar o caminho completo:

```
System.IO.File
```

A Classe System.IO.FileInfo – Informações Sobre Arquivos

A classe System.IO.FileInfo fornece algumas propriedades que retornam informações sobre um arquivo, tais como: nome do arquivo, nome do diretório, caminho completo e tamanho.

As considerações sobre o caminho a ser fornecido como parâmetro são as mesmas que foram feitas para a classe System.IO.File.

Propriedades da Classe System.IO.FileInfo

Na Tabela 5.3 temos a descrição das principais propriedades da classe System.IO.FileInfo.

Tabela 5.3 Principais propriedades da classe System.IO.FileInfo.

Propriedade	Descrição
Name	Retorna o nome do arquivo.
Length	Retorna o tamanho do arquivo ou pasta passado como parâmetro.

Propriedade	Descrição
Directory	Retorna uma instância de um objeto do tipo Directory, a qual aponta para a pasta onde está o arquivo.
DirectoryName	Retorna o caminho completo para o arquivo passado como parâmetro.

Vamos apresentar um exemplo no qual utilizamos as propriedades da classe System.IO.FileInfo.

Considere o exemplo da Listagem 5.7.

Listagem 5.7 – Propriedades da classe System.IO.FileInfo – ex7cap5.cs

```
using System;

using System.IO;

class ex7cap5
{
    // Exemplo 7 - Capítulo 5.

    // Propriedades de System.IO.FileInfo

    // Por: Júlio Battisti

    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {

        // Vamos criar uma variável meuarquivo, do tipo FileInfo,
        // a qual aponta para o arquivo C:\autoexec.bat

        FileInfo meuarquivo = new FileInfo("C:\\autoexec.bat");

        // Agora passaremos a utilizar algumas propriedades.

        // Vamos verificar o nome do arquivo c:\\autoexec.bat.

        Console.WriteLine("Nome -> {0}", meuarquivo.Name);

        // Vamos verificar o nome do arquivo c:\\autoexec.bat.
```

```

Console.WriteLine("Tamanho -> {0} bytes", meuarquivo.Length);

// VAMOS VERIFICAR O NOME DO ARQUIVO C:\AUTOEXEC.BAT.

Console.WriteLine("Caminho completo -> {0}", meuarquivo.DirectoryName);

}

}

```

Digite o exemplo da Listagem 5.7 e salve o mesmo em um arquivo chamado ex7cap5.cs, na pasta C:\ExCsharp\cap5. Compile e execute o exemplo da Listagem 5.7. Você obterá os resultados indicados na Figura 5.9.

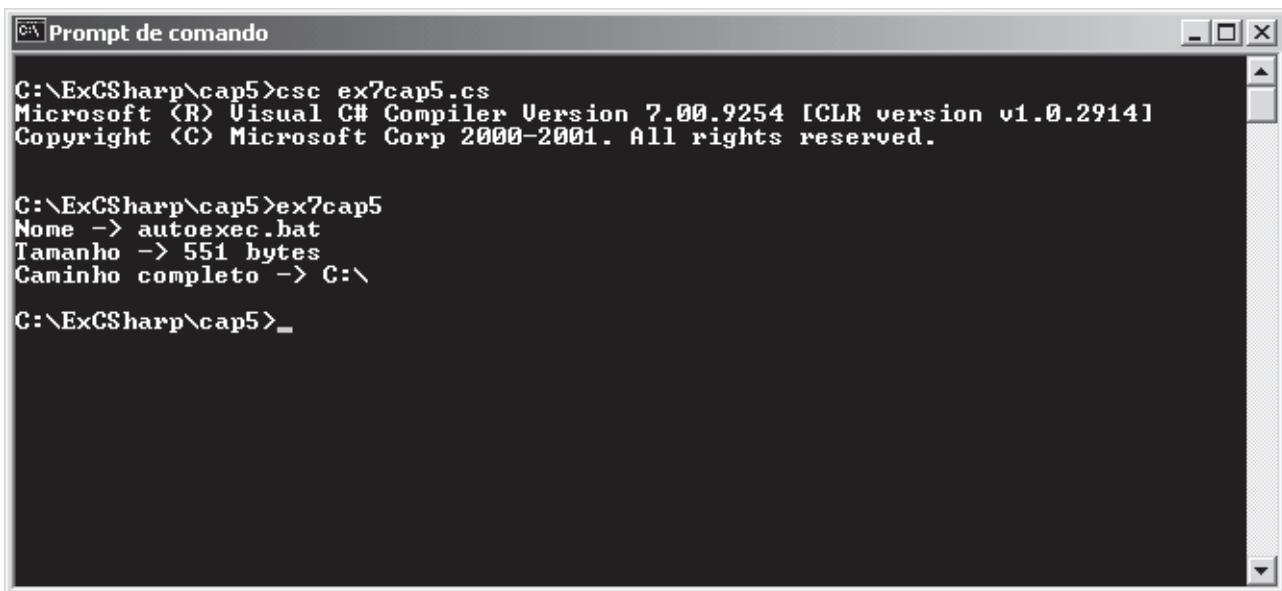


Figura 5.9: Executando o programa ex7cap5.exe.

O Tratamento de Exceções

Tratar exceções é de fundamental importância para que um programa não seja encerrado inesperadamente. Uma exceção pode acontecer durante o processamento do programa, quando algo inesperado acontece e deve ser tratado pelo programa, para que o mesmo não seja encerrado sem que o usuário saiba o que está acontecendo. Dois exemplos típicos de exceções:

- ◆ O programa tenta fazer uma leitura no disquete e não existe disquete no drive.
- ◆ Uma divisão por zero.

As exceções devem ser detectadas e opções devem ser oferecidas para o usuário do programa. Por exemplo, no caso do disquete que não está no drive, a exceção deve ser detectada e o programa deve exibir uma mensagem solicitando que o usuário insira um disquete no drive. Este procedimento é muito mais amigável do que simplesmente encerrar o programa.

Outra grande vantagem do Framework .NET é que o tratamento de exceções é padronizado, independentemente da linguagem que está sendo utilizada. Uma exceção gerada em um componente escrito em C++ pode ser tratada em um cliente escrito em C# e vice-versa.

Neste tópico veremos como tratar exceções em nossos programas C#.

Utilizando “try” e “catch”

Para definir o tratamento de exceções em nossos programas precisamos organizar os códigos em um bloco try e um bloco catch. Dentro do bloco try colocamos o código que pode gerar uma exceção – por exemplo os comandos que farão a leitura de um arquivo no disquete e que, se o disquete não estiver no drive, será gerada uma seção. O código para o tratamento da exceção é colocado dentro do bloco catch.

Vamos apresentar um exemplo no qual utilizamos try e catch para fazer o tratamento de exceções. O nosso programa solicita que o usuário digite dois números. Depois, o programa faz a divisão dos números e exibe o resultado. Para forçar uma exceção vamos fornecer um valor zero para o segundo número, de tal forma que o programa, ao tentar fazer uma divisão por zero, irá gerar uma exceção.

Considere o exemplo da Listagem 5.8.

Listagem 5.8 – Tratamento de exceções com try e catch – ex8cap5.cs

```
using System;

class ex8cap5
{
    // Exemplo 8 - Capítulo 5.
    // Tratamento de exceções com try e catch.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {

        // Início do bloco try.
        // Contém o código que pode gerar a exceção.

        try
        {
            // Códigos que podem gerar exceção
        }
        catch (Exception ex)
        {
            // Códigos que tratam a exceção
        }
    }
}
```

```
// Declaração das variáveis.

int divisao;

// Entrada dos valores de x e y

Console.WriteLine("Digite o NUMERADOR ->");

String Aux1=Console.ReadLine();

Console.WriteLine("Digite o DENOMINADOR ->");

String Aux2=Console.ReadLine();

// Cálculo da divisão.

divisao = Convert.ToInt32(Aux1) / Convert.ToInt32(Aux2);

// Exibição dos resultados.

Console.WriteLine("O valor da DIVISÃO é -> {0}",divisao);

}

// Final do bloco try.

// Início do bloco catch.

// Código que será executado se uma exceção
// for gerada no bloco try.

catch (Exception e)

{

    Console.WriteLine("FOI GERADA A SEGUINTE EXCEÇÃO: " + e.Message);

}

// Final do bloco catch.

}

}
```

Digite o exemplo da Listagem 5.8 e salve o mesmo em um arquivo chamado ex8cap5.cs, na pasta C:\ExCsharp\cap5. Compile e execute o exemplo da Listagem 5.8. Digite 10 para o numerador e 2 para o denominador. Você obterá os resultados indicados na Figura 5.10.

```
C:\ExCSharp\cap5>csc ex8cap5.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap5>ex8cap5
Digite o NÚMERADOR ->10
Digite o DENOMINADOR ->2
O valor da DIVISÃO é -> 5

C:\ExCSharp\cap5>
```

Figura 5.10: Executando, sem exceções, o programa ex8cap5.exe.

Observe que o programa executa normalmente. Agora vamos forçar uma exceção, e para isso digitaremos 0 para o segundo valor, forçando uma divisão por zero. Vamos executar novamente o programa. Digite 10 para o primeiro valor e 0 para o segundo. Você obterá os resultados indicados na Figura 5.11.

Neste segundo caso, ao tentar fazer uma divisão por zero, uma exceção será gerada. Ao ser gerada a execução, o código do bloco catch será executado. O bloco catch recebe um parâmetro do tipo Exception. Exception é uma classe do namespace System – System.Exception. Uma das propriedades desta classe é Message, a qual contém a mensagem associada com a exceção. No nosso exemplo a mensagem é: “Attempted to divide by zero”, o que confirma a nossa tentativa de fazer uma divisão por zero.

```
C:\ExCSharp\cap5>ex8cap5
Digite o NÚMERADOR ->10
Digite o DENOMINADOR ->0
FOI GERADA A SEGUINTE EXCEÇÃO: Attempted to divide by zero.

C:\ExCSharp\cap5>
```

Figura 5.11: Executando, forçando uma exceção, o programa ex8cap5.exe.

Existem classes que tratam exceções mais específicas, como por exemplo:

- ◆ System.OutOfMemoryException
- ◆ System.OverflowException
- ◆ System.NullReferenceException
- ◆ System.NotSupportedException
- ◆ System.NotImplementedException
- ◆ System.NotFiniteNumberException
- ◆ System.MissingMethodException
- ◆ System.MissingMemberException
- ◆ System.MissingFieldException
- ◆ System.MethodAccessException
- ◆ System.MemberAccessException
- ◆ System.InvalidProgramException
- ◆ System.InvalidOperationException
- ◆ System.InvalidCastException
- ◆ System.IndexOutOfRangeException
- ◆ System.FormatException
- ◆ System.FieldAccessException
- ◆ System.ExecutionEngineException
- ◆ System.EntryPointNotFoundException
- ◆ System.DuplicateWaitObjectException
- ◆ System.DllNotFoundException
- ◆ System.DivideByZeroException

Vamos utilizar algumas destas classes nos demais capítulos deste livro.

Utilizando “try” e “finally”

Em determinadas situações queremos que um determinado bloco de código seja executado, mesmo que não tenha sido gerada nenhuma exceção. Para que isso seja possível podemos utilizar finally ao invés de catch. O código dentro do bloco finally é sempre executado, mesmo que não tenha sido gerada nenhuma exceção.

Vamos modificar um pouco o exemplo anterior.

Considere o exemplo da Listagem 5.9.

Listagem 5.9 – Tratamento de exceções com try e finally – ex9cap5.cs

```
using System;

class ex9cap5
{
    // Exemplo 9 - Capítulo 5.
    // Tratamento de exceções com try e finally.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {
        // Bloco try.
        // Contém o código que pode gerar a exceção.

        try
        {

            // Declaração das variáveis.

            int divisao;

            // Entrada dos valores de x e y

            Console.Write("Digite o NUMERADOR ->");
            String Aux1=Console.ReadLine();

            Console.Write("Digite o DENOMINADOR ->");
            String Aux2=Console.ReadLine();

            // Cálculo da divisão.

            divisao = Convert.ToInt32(Aux1) / Convert.ToInt32(Aux2);

            // Exibição dos resultados.

        }
        catch (Exception ex)
        {
            // Tratamento da exceção.
            // Pode ser feito aqui.
        }
    }
}
```

```

        Console.WriteLine("O valor da DIVISÃO é -> {0}",divisao);
    }

    // Final do bloco try.

    // Início do bloco finally.

    // Código que será executado mesmo que nenhuma exceção
    // seja gerada no bloco try.

finally

{
    Console.WriteLine("CÓDIGO EXECUTADO TENHA OU NÃO SIDO GERADA UMA EXCEÇÃO");
}

// Final do bloco finally.

}
}

```

Digite o exemplo da Listagem 5.9 e salve o mesmo em um arquivo chamado ex9cap5.cs, na pasta C:\ExCsharp\cap5. Compile e execute o exemplo da Listagem 5.9. Digite 10 para o numerador e 2 para o denominador. Você obterá os resultados indicados na Figura 5.12. Observe que o código do bloco finally foi executado, mesmo sem ter sido gerada nenhuma exceção.

```

C:\Promt de comando
C:\ExCSharp\cap5>csc ex9cap5.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap5>ex9cap5
Digite o NÚMERADOR ->10
Digite o DENOMINADOR ->2
O valor da DIVISÃO é -> 5
CÓDIGO EXECUTADO TENHA OU NÃO SIDO GERADA UMA EXCEÇÃO

C:\ExCSharp\cap5>

```

Figura 5.12: Executando, sem exceções, o programa ex9cap5.exe.

Agora vamos forçar uma exceção, e para isso digitaremos 0 para o segundo valor, forçando uma divisão por zero. Vamos executar novamente o programa. Digite 10 para o primeiro valor e 0 para o segundo. Na Figura 5.13 é aberta uma janela indicando que ocorreu uma exceção no programa.

Neste caso, como não havia um bloco catch, a exceção não foi tratada. Por isso surgiu a janela indicada na Figura 5.13. Dê um clique em OK para fechar a janela de aviso e observe que o código do bloco finally foi executado, mesmo tendo sido gerada uma exceção, conforme indicado pela Figura 5.14:

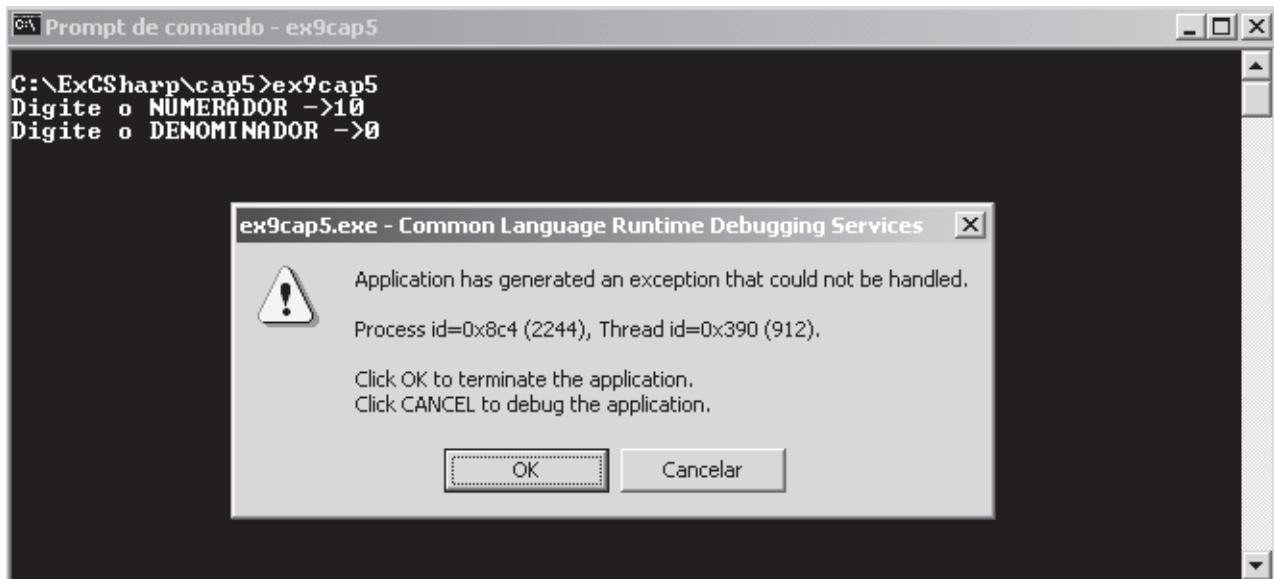


Figura 5.13: Aviso de que uma exceção foi gerada.

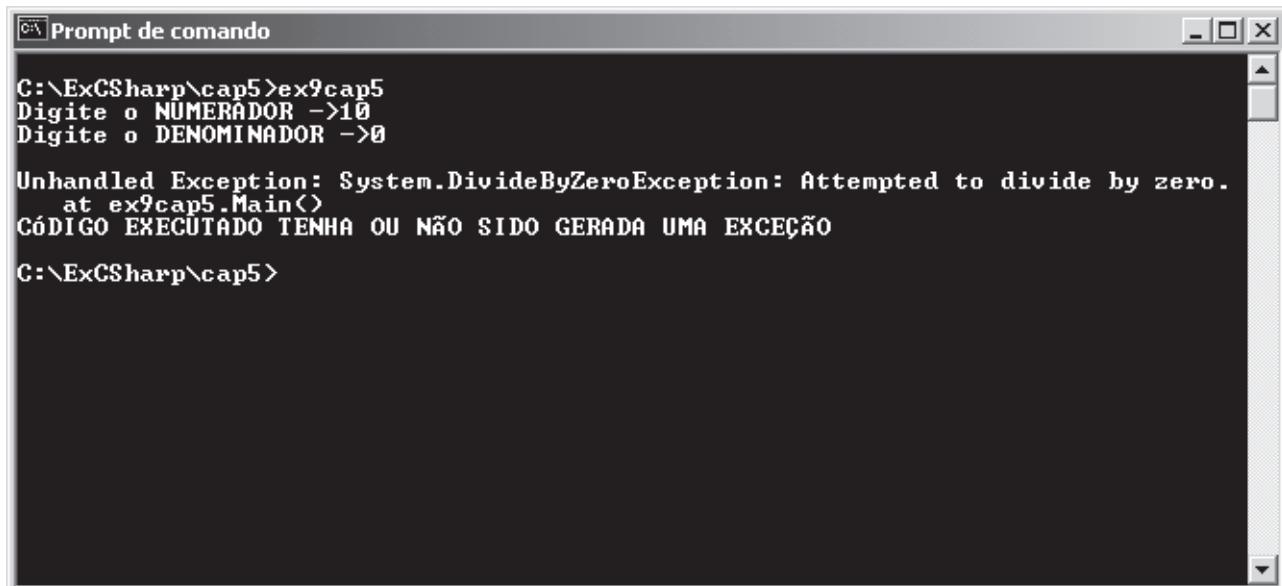


Figura 5.14: O código do bloco finally é sempre executado.

Mas, se quisermos ter as duas coisas, ou seja:

- ◆ Código para tratamento de exceções.
- ◆ Código que sempre executa.

Neste caso temos que juntar os blocos catch e finally, em um arranjo conhecido como try – catch – finally. Vamos alterar o exemplo anterior para termos uma estrutura try – catch – finally, de tal forma que as exceções sejam tratadas e tenhamos um código que sempre executa no encerramento do programa.

Considere o exemplo da Listagem 5.10.

Listagem 5.10 – Tratamento de exceções com try – catch – finally – ex10cap5.cs

```
using System;

class ex10cap5
{
    // Exemplo10 - Capítulo 5.
    // Tratamento de exceções com try - catch - finally.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {
        // Bloco try.
        // Contém o código que pode gerar a exceção.

        try
        {
            // Declaração das variáveis.

            int divisao;

            // Entrada dos valores de x e y

            Console.Write("Digite o NUMERADOR ->");
            String Aux1=Console.ReadLine();
        }
    }
}
```

```
Console.WriteLine("Digite o DENOMINADOR ->");

String Aux2=Console.ReadLine();

// Cálculo da divisão.

divisao = Convert.ToInt32(Aux1) / Convert.ToInt32(Aux2);

// Exibição dos resultados.

Console.WriteLine("O valor da DIVISÃO     é -> {0}",divisao);
}

// Final do bloco try.

// Início do bloco catch.

// Código que será executado se uma exceção
// for gerada no bloco try.

catch (Exception e)

{

Console.WriteLine("FOI GERADA A SEGUINTE EXCEÇÃO: " + e.Message);
}

// Final do bloco catch.

// Início do bloco finally.

// Código que será executado mesmo que nenhuma exceção
// seja gerada no bloco try.

finally

{

Console.WriteLine("CÓDIGO EXECUTADO TENHA OU NÃO SIDO GERADA UMA EXCEÇÃO");
}

// Final do bloco finally.

}

}
```

Digite o exemplo da Listagem 5.10 e salve o mesmo em um arquivo chamado ex10cap5.cs, na pasta C:\ExCsharp\cap5. Compile e execute o exemplo da Listagem 5.10. Digite 10 para o numerador e 2 para o denominador. Você obterá os resultados indicados na Figura 5.15. Observe que o código do bloco finally foi executado, mesmo sem ter sido gerada nenhuma exceção. Como não foi gerada exceção, o código do bloco catch não foi executado.

```
C:\ExCSharp\cap5>csc ex10cap5.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

C:\ExCSharp\cap5>ex10cap5
Digite o NÚMERADOR ->10
Digite o DENOMINADOR ->2
O valor da DIVISÃO é -> 5
CÓDIGO EXECUTADO TENHA OU NÃO SIDO GERADA UMA EXCEÇÃO

C:\ExCSharp\cap5>
```

Figura 5.15: Executando, sem exceções, o programa ex10cap5.exe.

Agora vamos forçar uma exceção, e para isso digitaremos 0 para o segundo valor, forçando uma divisão por zero. Vamos executar novamente o programa. Digite 10 para o primeiro valor e 0 para o segundo. Na Figura 5.16 observamos que a exceção foi identificada e a mensagem respectiva foi emitida. Isto indica que o código do bloco catch foi executado e, portanto, a exceção foi tratada. Também podemos observar que o código do bloco finally foi executado, mesmo tendo sido gerada uma exceção.

```
D:\W2KSRVIN\System32\command.com

C:\EXCSHARP\CAP5>ex10cap5
Digite o NÚMERADOR ->10
Digite o DENOMINADOR ->0
FOI GERADA A SEGUINTE EXCEÇÃO: Attempted to divide by zero.
CÓDIGO EXECUTADO TENHA OU NÃO SIDO GERADA UMA EXCEÇÃO

C:\EXCSHARP\CAP5>_
```

Figura 5.16: Execução, com exceções, do programa ex10cap5.exe.

Conclusão

Neste capítulo encerramos nosso estudo sobre os aspectos básicos do C#. Estudamos os elementos da linguagem C# que serão utilizados no restante do livro. Com esse capítulo encerramos a Parte I do livro, na qual falamos sobre o Framework .NET e sobre a linguagem C#. Estes capítulos formam a base para o restante do livro. A partir do Capítulo 6, com o qual iniciamos a Parte II deste livro, estaremos tratando da criação de páginas ASP.NET.

P A R T E

2

Fundamentos do ASP.NET

Introdução

Na Parte I deste livro (Capítulos de 1 a 5) tratamos dos conceitos básicos do Framework .NET e da linguagem C#. Neste capítulo iniciamos a Parte II do livro, na qual vamos apresentar esta nova versão da tecnologia ASP (Active Server Pages) – ASP.NET.

Vamos iniciar o capítulo fazendo um rápido histórico sobre as várias versões do ASP, desde o seu lançamento com o IIS 3.0 até a nova versão ASP.NET que faz parte do Framework .NET.

Em seguida vamos falar sobre as vantagens e novas características e o porquê de precisarmos de uma nova versão, sendo que a última versão nem completou dois anos. Falaremos sobre as diversas novidades do ASP.NET e como cada uma poderá ajudar no desenvolvimento mais rápido de aplicações Web melhores, mais confiáveis e mais seguras.

Dentre as principais novidades que vamos apresentar neste capítulo e detalhar nos demais, podemos destacar as seguintes:

- ◆ Separação entre lógica e apresentação.
- ◆ Mais fácil de implementar.
- ◆ Suporte a múltiplas linguagens.
- ◆ Web Form Controls – controles mais avançados e “inteligentes”.
- ◆ Todas as vantagens oferecidas pelo Framework .NET.

Uma vez “feita a propaganda” do ASP.NET vamos verificar se todo o suporte ao ASP.NET já está instalado, para que você possa acompanhar os exemplos deste livro.

Para finalizar o capítulo apresentaremos alguns exemplos simples, apenas para termos uma idéia de como é uma página ASP.NET.

Ao final deste capítulo você deverá estar apto a relacionar as principais melhorias do ASP.NET em relação às versões anteriores. Você também será capaz de entender a estrutura de uma página ASP.NET bem como a maneira com que as mesmas são processadas pelo servidor IIS – Internet Information Services.

CAPÍTULO

6

Uma Introdução ao ASP.NET

Mais uma Versão?

ASP, ASP 2.0, ASP 3.0 e agora ASP.NET.

Menos de dois anos após o lançamento do ASP 3.0, o qual faz parte do IIS 5.0 com o Windows 2000, a Microsoft lança mais uma versão da tecnologia ASP – Active Server Pages. Na verdade não é apenas o lançamento de mais uma versão. O ASP.NET está inserido em um contexto maior, que é a iniciativa .NET da Microsoft. Além de estar inserido no contexto do .NET, a tecnologia ASP.NET apresenta “enormes” diferenças em relação ao ASP 3.0, o que não permite que a caracterizemos simplesmente como um upgrade do ASP 3.0.

A tecnologia ASP.NET segue os mesmos princípios do Framework .NET, cujo principal objetivo é facilitar o desenvolvimento de aplicações. No caso específico do ASP.NET, aplicações Web. Conforme veremos mais adiante, com ASP.NET as aplicações Web passam a usufruir de todos os recursos do Framework .NET.

A tecnologia ASP teve uma enorme aceitação por parte da comunidade de desenvolvedores. Para comprovar tal afirmação basta conferir o número de sites que utilizam ASP. Esta aceitação não se deve somente à força de mercado da Microsoft, mas também à facilidade de desenvolvimento propiciada pela utilização de ASP. Antes de falarmos sobre a nova versão – ASP.NET, vamos ver exatamente o que é a tecnologia de Active Server Pages da Microsoft.

Uma Introdução à Tecnologia ASP

A Internet deixou de ser uma rede somente para pesquisas acadêmicas e passou a ser conhecida no mundo inteiro quando recebeu uma interface gráfica. Isso foi possível graças ao desenvolvimento dos servidores Web, a utilização do protocolo HTTP (Hypertext Transfer Protocol), da linguagem HTML (Hypertext Markup Language) e de um programa conhecido como Browser (Navegador). Com essa receita, estava formada a WWW – World Wide Web ou simplesmente Web.

O conteúdo dos sites ficava armazenado em arquivos de código HTML (.htm ou .html) nos servidores Web ou servidores HTTP como também são conhecidos. O usuário utiliza um programa, o Browser, para acessar as páginas HTML. Com essa combinação, uma página HTML era capaz de exibir, além de texto, elementos gráficos. Com isso a internet passou a ser mais atraente e começou a conquistar um número cada vez maior de usuários.

Na Figura 6.1 temos uma visão geral dos elementos descritos.

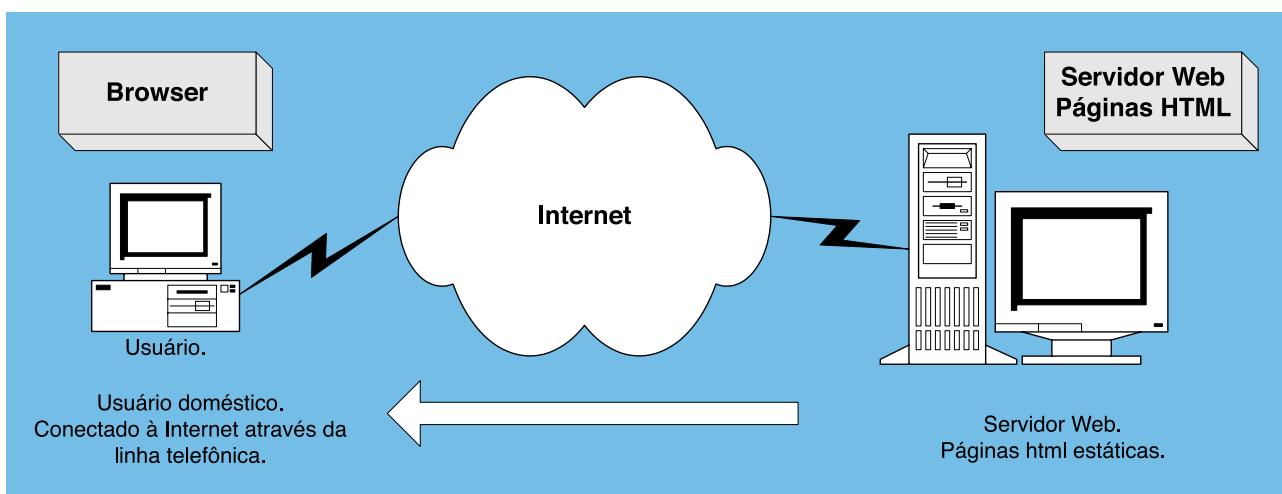


Figura 6.1: Internet com conteúdo estático.

O problema da linguagem HTML é que a mesma fornece conteúdo estático, isto é, não dinâmico. Vamos exemplificar. Se utilizamos o HTML para publicar uma lista de preços de um catálogo de produtos. Todo o conteúdo do catálogo com a lista de preços faz parte do próprio código HTML. Cada vez que tivermos que alterar um preço temos que editar o arquivo HTML e salvar as alterações. Somente após fazermos isto é que o usuário passará a ter acesso às modificações.

Com o crescimento da Internet e o uso cada vez maior da rede para operações de comércio, ficou claro que a tecnologia de páginas estáticas do HTML não seria suficiente para a criação de sites e aplicações Web que suportassem o Comércio Eletrônico.

Então começaram a surgir tecnologias para a criação dinâmica de conteúdo, a partir de informações contidas em bancos de dados e outras fontes de informações. A tecnologia pioneira foi a dobradinha CGI com a linguagem Perl.

A Microsoft, como não poderia deixar de ser, resolveu entrar nesta briga. Em primeiro lugar lançou o seu próprio servidor Web, que é o IIS – Internet Information Services. Por ser gratuito e devido à grande base Windows instalada, o IIS rapidamente foi adotado por uma grande parcela de sites. Com a versão 3 do IIS, a Microsoft lança a sua tecnologia para a criação de páginas dinâmicas, o ASP 1.0 – Active Server Pages 1.0.

Com a tecnologia ASP, ao invés de páginas .htm ou .html, criamos páginas com a extensão .asp. Estas páginas contêm, além de código HTML, o chamado código ASP, o qual pode realizar uma série de operações que tornam as páginas dinâmicas. Vamos utilizar o exemplo anterior, onde queríamos construir um catálogo de preços. Com a tecnologia ASP, podemos fazer com que a página, ao ser carregada, busque a lista de preços em um banco de dados, monte uma página com estas informações e retorne a página para o usuário, exibindo sempre os resultados mais atualizados. Na Figura 6.2 temos uma visão desta nova fase, com conteúdo dinâmico.

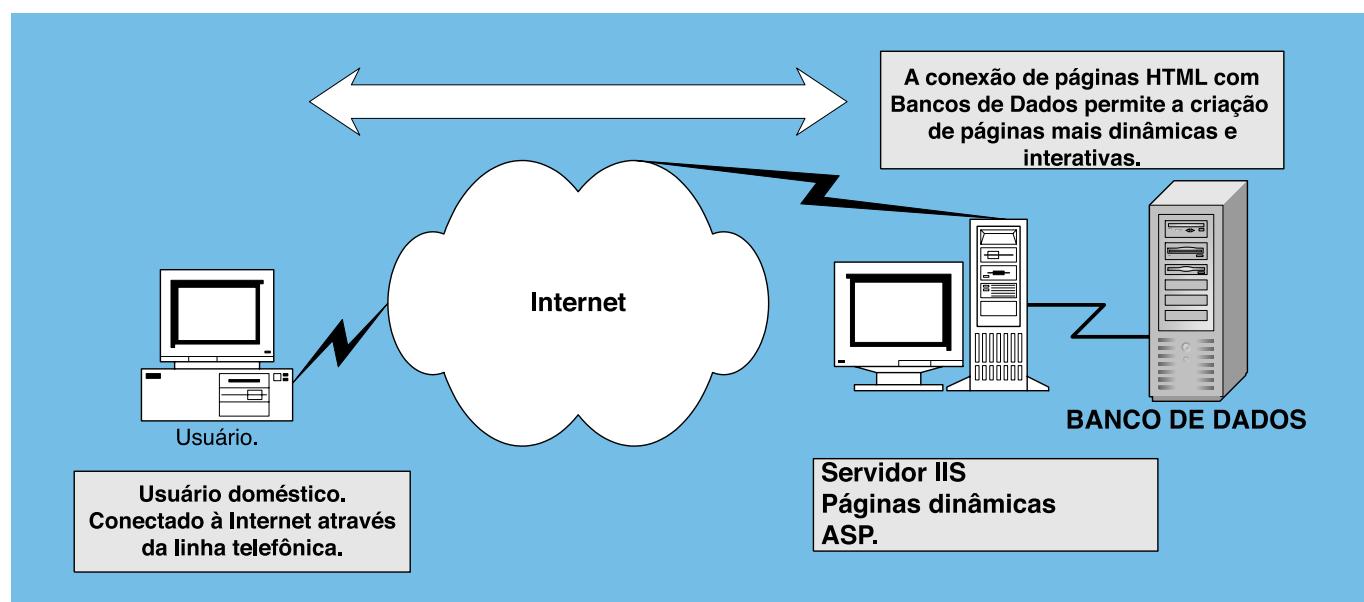


Figura 6.2: Internet com conteúdo dinâmico.

Além de criar uma página que exibe as informações mais atualizadas, podemos criar formulários interativos, onde o usuário pode inserir o código ou nome de um produto e enviar tais informações para o servidor IIS. Uma página ASP

recebe estas informações, conecta com um banco de dados e faz uma pesquisa utilizando os critérios passados pelo usuário. Uma vez encerrada a pesquisa, a página ASP retorna apenas os resultados que atendem os critérios da pesquisa. Observe que agora, além de receber informações da Internet, o usuário também pode enviar informações.

Vamos detalhar um pouco mais este processo.

Conteúdo Dinâmico na Internet

Com o crescimento da Internet e a necessidade de constantes alterações no conteúdo das páginas, surge uma segunda geração de sites na Internet, capazes de entregar conteúdo sempre atualizado, além de permitir que o usuário interaja com as páginas Web, enviando informações e não apenas recebendo. Nesta fase surge a possibilidade de ligação das páginas HTML com o conteúdo de Bancos de dados. Conforme descrevemos no tópico anterior, a tecnologia pioneira para a criação de conteúdo dinâmico foi a dobradinha CGI/Perl. No diagrama da Figura 6.2 temos uma visão geral deste processo.

Pelo diagrama, podemos ver a possibilidade de o usuário enviar informações para a Internet, e não apenas receber informações. Isto possibilitou o desenvolvimento de uma série de serviços, simplesmente impossíveis de criar, apenas com a utilização de páginas criadas somente com HTML.

Para enviar informações, o usuário preenche os campos de um formulário (o qual é criado com a utilização de HTML) e, ao clicar em um botão Enviar, os dados são enviados para o site da Web. Ao chegar no servidor Web, estes dados precisam ser “recebidos” por um programa capaz de entender o formato dos dados e armazená-los em um Banco de dados.

Os primeiros programas, capazes de realizar esta tarefa, seguiam a especificação conhecida como CGI – Common Gateway Interface. Muitas aplicações Web foram desenvolvidas, utilizando-se a especificação CGI, sendo que várias ainda continuam sendo utilizadas. A utilização de Scripts desenvolvidos na linguagem Perl é um exemplo de utilização de CGI. O Script recebe os dados enviados pelo formulário, decodifica estes dados e armazena o resultado em um Banco de dados. Embora bastante funcional, a utilização de CGI começou a apresentar alguns problemas, e com isso novas alternativas foram surgindo. Está fora do escopo deste livro, discutir os problemas da utilização de CGI.

Dentre as várias alternativas que surgiram para a geração de conteúdo dinâmico, podemos citar a tecnologia de Active Server Pages, a qual faz parte do servidor Web IIS (Internet Information Services), da Microsoft. Podemos criar uma página ASP, capaz de receber os dados enviados por um formulário e armazenar estes dados em um Banco de dados, como por exemplo o Microsoft Access ou o Microsoft SQL Server.

Com a conexão de páginas com Bancos de dados, uma série de possibilidades novas surgiu, como por exemplo:

- ◆ Criação de páginas para pesquisa em Banco de dados.
- ◆ Cadastro de usuários que acessam o site, bem como a entrega de conteúdo personalizado, de acordo com as preferências do usuário. Por exemplo, ao entrar no site, o usuário informa um nome de usuário e senha, com o qual o mesmo foi previamente cadastrado. Com isso é aberta uma página com opções e conteúdo personalizados, de acordo com preferências especificadas pelo usuário.

- ◆ Desenvolvimento de aplicações residentes em servidores Web e acessíveis através do Navegador. Devemos observar que estas aplicações ainda eram limitadas e não possuíam todas as funcionalidades das aplicações convencionais, desenvolvidas para o ambiente Windows.

Apesar das suas limitações, um novo panorama desenhava-se na Internet, com a possibilidade da criação de sites mais dinâmicos, através da conexão com Bancos de dados. Apenas para exemplificar o funcionamento de uma pesquisa em Banco de dados, através da Internet, observe a Figura 6.3.

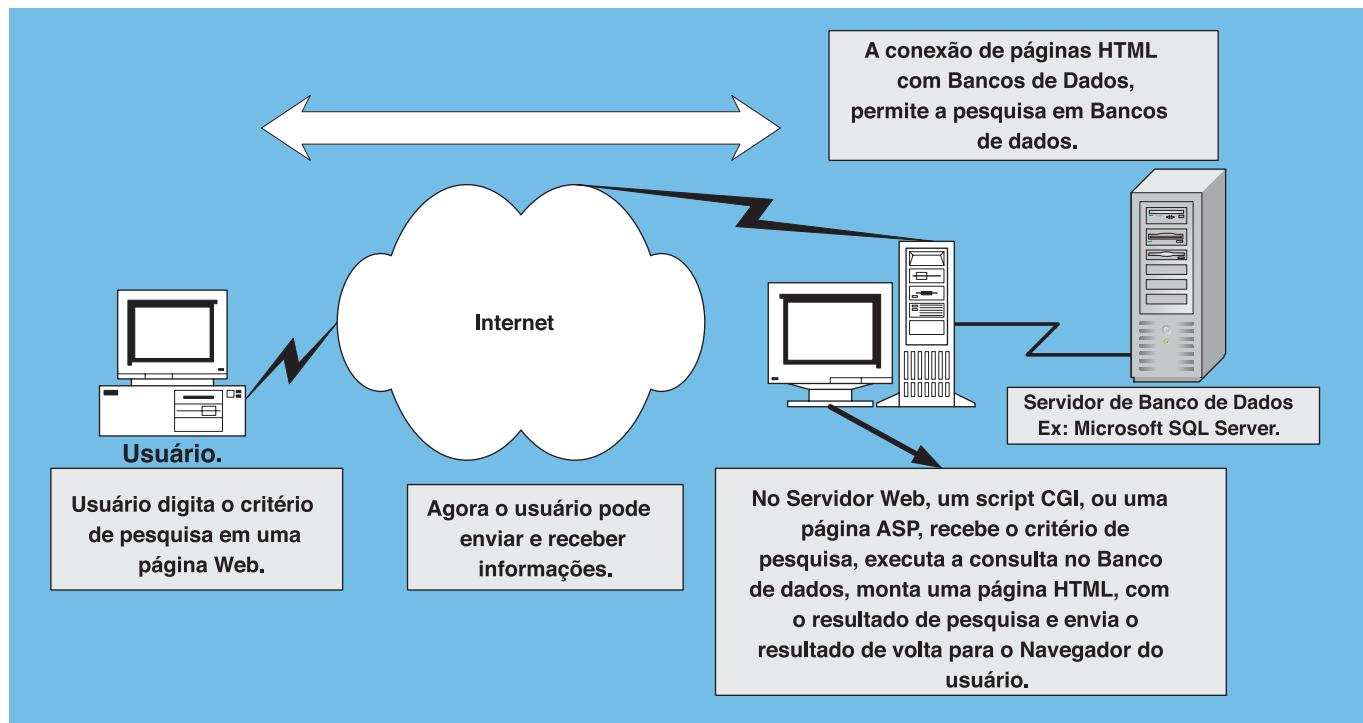


Figura 6.3: Pesquisa em um Banco de dados, através da Internet.

Vamos analisar os passos executados, desde o momento em que o usuário preenche o critério de pesquisa, até o momento em que o resultado da consulta é retornado:

- ◆ O usuário acessa a página onde temos um formulário para a digitação de um ou mais critérios de pesquisa. Por exemplo, pode ser uma consulta a uma base dos CEP's de todo o Brasil. O usuário poderia digitar o nome da Cidade, selecionar um Estado e digitar o nome ou parte do nome da Rua.
- ◆ Após preencher o(s) critério(s) de pesquisa, o usuário clica em um botão Enviar ou Pesquisar. Os dados digitados no formulário são enviados para o servidor.
- ◆ Um Script CGI, ou uma página ASP, no servidor, recebe os dados enviados pelo usuário. Com estes dados, é montado um comando (normalmente uma String SQL – Structured Query Language), o qual vai efetuar a consulta no Banco de dados. Após montado o comando, o mesmo é enviado para o Servidor de Banco de dados (o qual pode ser uma máquina separada, ou pode estar instalado no próprio servidor Web).
- ◆ O Servidor de Banco de dados recebe o comando de pesquisa, localiza um ou mais registros que atendam o(s) critério(s) de pesquisa e retorna o resultado para o Script CGI, ou para a Página ASP.

- ◆ Com o resultado retornado pelo Servidor de Banco de dados, o Script CGI, ou a Página ASP, monta uma página HTML, normalmente no formato de uma tabela, e envia esta página HTML de volta para o Navegador do cliente. Caso não seja encontrado nenhum registro que atenda o(s) critério(s) especificado(s), é retornada uma página com uma mensagem de que não foi encontrado nenhum registro.

Veja que, ao mesmo tempo em que aumentaram as possibilidades de desenvolvimento, também aumentaram as complexidades a serem gerenciadas, tanto na criação de páginas, quanto no gerenciamento dos sites. Observe que ainda nem falamos de questões tais como segurança, proteção do Banco de dados contra acessos não autorizados, ou até mesmo ataques de Hackers tentando roubar informação ou corromper a informação do Banco de dados.

A tecnologia ASP 3.0 possibilita a criação do formulário de pesquisa descrito no nosso exemplo. Utilizando ASP 3.0 em conjunto com o padrão COM/COM+ da Microsoft, podemos criar aplicações Web de 3 ou mais camadas.

Mas, se podemos criar aplicações tão sofisticadas e funcionais com ASP 3.0, porque precisamos de uma nova versão? Vamos responder a esta pergunta no próximo item, onde estaremos falando sobre as principais melhorias do ASP.NET em relação ao ASP 3.0. Nos demais capítulos do livro vamos aprender a utilizar os novos recursos do ASP.NET.

Novidades e Melhorias do ASP.NET

Neste item vamos apresentar as novidades e vantagens do ASP.NET. Mais do que uma simples justificativa para uma nova versão, procuro demonstrar os benefícios de utilizarmos ASP.NET para a criação de aplicações WEB.

Faz Parte do Framework .NET

Pode parecer óbvio, mas nunca é demais salientar este aspecto. ASP.NET é parte integrante do Framework .NET, e desta maneira tem acesso a todos os benefícios da plataforma .NET.

Em uma página ASP.NET temos acesso a todas as classes da biblioteca de classes do Framework .NET – .NET Framework Class Library. Também temos acesso a um tipo comum de dados e a todos os benefícios do Framework .NET. Só estes benefícios já justificam uma migração do ASP 3.0 para o ASP.NET.

Suporte a Múltiplas Linguagens

Com as versões anteriores do ASP estávamos limitados praticamente a duas linguagens de Script: VBScript e JScript. Além disso, por serem linguagens de Script as mesmas eram interpretadas, ou seja, ao carregar uma página ASP, o IIS precisa ler linha por linha de código, interpretar e executar.

Com ASP.NET podemos utilizar qualquer linguagem que esteja habilitada para o Framework .NET. Inicialmente são disponibilizadas as linguagens VB.NET, C# e JScript. Apesar do nome JScript, esta e as demais linguagens são compiladas, o que significa que toda página ASP.NET é compilada antes de ser executada. Na primeira vez que uma página ASP.NET é acessada, a mesma é compilada em uma Classe do Framework .NET. A versão compilada é mantida em Cache para melhorar o desempenho para os próximos acessos. Qualquer alteração na página é automaticamente

detectada pelo Framework .NET; a página é recompilada e a versão em Cache é atualizada, fazendo com que o cliente tenha sempre acesso à versão mais atualizada da página. Se não houver mudanças continua sendo utilizada a versão que está no Cache, o que torna a carga da página muito mais rápida.

Além disso, diversas empresas de Software já anunciaram que estão portando suas linguagens para que sejam compatíveis com o Framework .NET. Desta maneira você poderá utilizar, para a criação de páginas ASP.NET, a linguagem com a qual está mais familiarizado, desde que a mesma esteja habilitada ao .NET. Você prefere Delphi? Sem problemas. É fã do COBOL? Por que não?

Menos Código Para Mais Trabalho

Com o ASP.NET temos acesso a uma série de facilidades que diminuem a quantia de código que o programador precisa escrever, principalmente no trato com funções básicas. Estas funções básicas, como a manutenção de estado entre chamadas de uma página, são responsabilidade do Framework .NET. Isso faz com que o programador tenha que desenvolver apenas a lógica da aplicação, sem ter que se preocupar com aspectos básicos da Infra-estrutura.

Uma série de controles mais inteligentes e com melhor funcionalidade foi criada, os chamados Server controls. Estes controles fornecem grande funcionalidade com um mínimo de programação. Alguns artigos fazem referência à “inteligência” dos server controls. Um dos aspectos considerados inteligentes é que, no momento da carga da página, o controle detecta, por exemplo, se o navegador do cliente é o Internet Explorer ou o Netscape Navigator e adapta o seu comportamento e exibição de acordo com o navegador do cliente.

Com ASP tarefas simples, como por exemplo validar os dados digitados em formulários, exigem a criação de código específico. Com os controles “inteligentes” do ASP.NET, fazer a validação é simplesmente questão de configurar algumas propriedades do controle e pronto, a verificação será feita no momento em que a página for acessada. Nos Capítulos 7, 8 e 9 estaremos falando sobre os diversos controles disponíveis. Daremos destaque especial às seguintes categorias de controles:

- ◆ HTML Server Controls.
- ◆ Validation Server Controls.
- ◆ Web Server Controls.

Separação Entre o Código HTML e o Código ASP

Esta é uma das melhorias que eu mais aprecio. Com ASP 3.0 ou versões anteriores, nós tínhamos uma mistura (literalmente) entre o código ASP e o código HTML, o que torna o código de difícil leitura e documentação. Apenas para exemplificar, vamos apresentar uma página ASP que faz conexão com um banco de dados do Microsoft Access e exibe todos os registros de uma determinada tabela do banco de dados. Os registros são exibidos na forma de uma tabela. O Código HTML para a criação da tabela, que será exibida para o usuário, é gerado a partir de uma mescla entre código ASP e código HTML.

Na Listagem 6.1 temos um exemplo de página ASP que faz conexão com um banco de dados do Microsoft Access.

Listagem 6.1 – Uma página ASP – código ASP e HTML “misturado”.

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
<TITLE>Listagem de Clientes</TITLE>
</HEAD>

<BODY>
<H1>
<FONT color=navy>Clientes da Empresa ABC LTDA.</FONT>
</H1>

<%
' O Primeiro passo é criar a conexão com o Banco de dados
' Para isto crio um objeto do tipo Connection
' Cria um Objeto do Tipo ADODB.Connection

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

' Agora abro uma conexão utilizando OLE-DB

' O código a seguir aparece em duas linhas por questão de espaço, porém o mesmo
' deve ser digitado em uma única linha.

conn.Open "PROVIDER=MICROSOFT.JET.OLEDB.4.0;DATA SOURCE=c:\meus documentos\wind.mdb"

' O próximo passo é criar uma instrução SQL
' a qual é utilizada para criar a listagem de Clientes.
' O código a seguir aparece em duas linhas por questão de espaço, porém o mesmo
' deve ser digitado em uma única linha.

inst_sql="SELECT Clientes.CodigoCliente, Clientes.Cargo,
Clientes.Endereco, Clientes.Fone FROM Clientes"

' Esta instrução SQL retorna os campos CodigoCliente, Cargo,
' Endereco e Fone, da tabela Clientes.
' Agora criamos um Objeto RecordSet.
' Este Objeto irá executar a instrução SQL e
' receber o resultado da Consulta.

Set Clientes = Server.CreateObject("ADODB.Recordset")
```

```
'Agora executamos a Instrução SQL
'retornando os registros da tabela Clientes.

Clientes.Open inst_sql, conn, 3, 3

' Os dois últimos parâmetros serão discutidos
' no próximo Capítulo.

' Neste ponto já tenho todos os registros retornados
' pela instrução SQL. Estes registros estão armazenados
' no objeto Clientes, que é um objeto do tipo Recordset.

' Agora passo a montar a página que será retornada para o
' Navegador do Cliente. Vamos montar uma tabela com o
' resultado da consulta.

%>

<P>
<HR>

<TABLE bgColor=gray border=1 borderColor=navy cellPadding=1 cellSpacing=1
width="100%">

<TR>
    <TD align=middle bgColor=gray>Código</TD>
    <TD align=middle bgColor=gray>Cargo</TD>
    <TD align=middle bgColor=gray>Endereço</TD>
    <TD align=middle bgColor=gray>Fone</TD>
</TR>
<%
' Inicio um Loop para percorrer todos os registros
' do RecordSet Clientes, exibindo um Registro em
' cada Linha da tabela.
Do Until Clientes.eof %>

<TR>
    <TD align=middle bgColor=gray><%=Clientes.Fields("CodigoCliente")%></TD>
    <TD align=middle bgColor=gray><%=Clientes.Fields("Cargo")%></TD>
    <TD align=middle bgColor=gray><%=Clientes.Fields("Endereco")%></TD>
    <TD align=middle bgColor=gray><%=Clientes.Fields("Fone")%></TD>
</TR>
```

<%

```

Clientes.MoveNext
loop %>
</TABLE>
</P>
<HR>
</BODY>
</HTML>

```

Na Figura 6.4 temos esta página carregada no IE – Internet Explorer.

Observe que por mais organizados que sejamos, inserindo comentários e endentações no código, não é nada fácil a leitura e interpretação de uma página ASP. Isso acontece devido, principalmente, à intercalação de código ASP e código HTML. Em uma mesma página ASP posso ter várias seções de código ASP. Cada seção inicia com <% e encerra com %>. Tudo o que não estiver entre estes dois marcadores é código HTML.

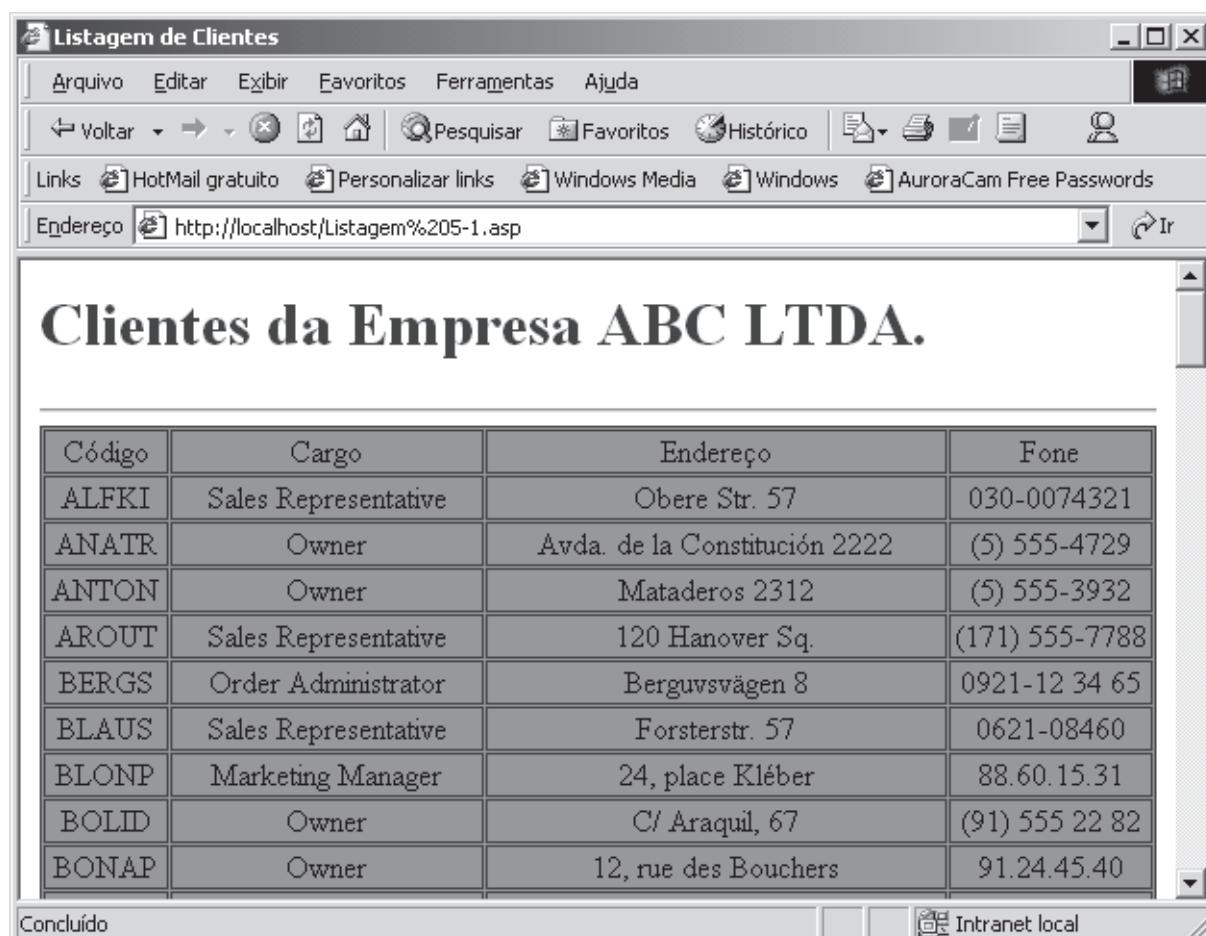


Figura 6.4: Página ASP de exemplo.

Já com ASP.NET isto não acontece. A parte de apresentação da página é separada da parte de processamento, isto é, da parte onde está o código responsável pela lógica de processamento da página. Com isto estamos separando a lógica da apresentação, o que facilita a manutenção das páginas.

Maiores Facilidades Para Criação e Utilização de Componentes

Esta é mais uma das vantagens propiciadas pelo Framework .NET. Conforme descrevemos nos Capítulos 1 e 2, a criação de componentes com o Framework .NET ficou muito mais fácil. Mais do que a criação, a distribuição e utilização destes componentes ficou extremamente simplificada. Não existe mais a necessidade de registrar um determinado componente como acontece com os componentes COM/COM+.

Outra novidade importante é a possibilidade de diferentes versões de um componente executarem em um mesmo servidor. É possível, inclusive, a execução simultânea de diferentes versões de um mesmo componente, o que é conhecido por execução “syde by syde”. Ao facilitar a criação e utilização de componentes, o Framework .NET incentiva a reutilização de código.

Compatibilidade com Qualquer Navegador

O resultado da execução de uma página ASP.NET é código HTML retornado para o cliente. Qualquer navegador é capaz de interpretar corretamente HTML, em tese. Na prática o que acontece é que existem pequenas diferenças e até mesmo “inconsistências” entre a maneira como o código HTML é interpretado. Para minimizar este problema, o ASP.NET conta com os controles que rodam no servidor e são capazes de fornecer diferentes saídas, dependendo do navegador que fez a solicitação da página. Na prática isto significa que um controle da interface de uma página ASP.NET, que roda no servidor, é capaz de gerar diferentes saídas, dependendo do navegador do cliente. Esta característica é conhecida por AUI – Adaptive User Interface.

Check List Para Acompanhar os Exemplos Deste Livro

Na Introdução deste livro descrevi as características do computador que estou utilizando. A seguir, repito estas informações.

Para criar os exemplos deste livro, utilizei um servidor com as seguintes configurações:

- ◆ Pentium 450
- ◆ 256 MB RAM
- ◆ Windows 2000 Server em Português
- ◆ IIS 5.0
- ◆ Microsoft .NET Framework SDK
- ◆ Nome do servidor: Servidor
- ◆ Endereço IP: 10.204.123.1

- ◆ Máscara de sub-rede: 255.255.255.0
- ◆ Domínio DNS: groza.com
- ◆ Pasta padrão: D:\Inetpub\wwwroot

O Framework .NET está disponível para várias versões do Windows. ASP.NET é suportado no Windows 2000 e no Windows NT 4 com Service Pack 6a. A utilização de Web Services é suportada em todas as plataformas suportadas pelo Microsoft .NET Framework SDK, com exceção do Windows 95.

O endereço para acessar a página inicial do servidor é o seguinte: <http://www.groza.com> ou <http://localhost>. Sempre que for feita referência a um destes dois endereços, você deve substituir pelo endereço do equipamento que você está utilizando para acompanhar os exemplos deste livro.

Ao instalarmos o IIS é criada uma pasta conhecida por Home Directory. O Home Directory é a pasta padrão que é acessada quando digitamos o endereço do servidor IIS, como por exemplo: <http://localhost>. Dentro da pasta padrão podemos definir um documento padrão, como por exemplo index.asp. Com isso ao acessar o servidor <http://localhost>, a página padrão (index.asp) do diretório padrão será automaticamente carregada. O diretório padrão é criado na pasta \Inetpub\wwwroot do drive onde o Windows 2000 está instalado. Ao instalar o IIS o usuário pode alterar a localização da pasta padrão. No computador que estou utilizando, o Windows 2000 Server está instalado em D:\Win2k. Por isso o Home Directory do servidor está em D:\Inetpub\wwwroot.

Dentro da pasta wwwroot irei criar uma pasta para cada capítulo, conforme indicado na Tabela 6.1.

Tabela 6.1 Pastas criadas no Home Directory do servidor IIS

Capítulo	Pasta
Capítulo6	D:\Inetpub\wwwroot\chap6
Capítulo7	D:\Inetpub\wwwroot\chap7
Capítulo8	D:\Inetpub\wwwroot\chap8
Capítulo9	D:\Inetpub\wwwroot\chap9
Capítulo10	D:\Inetpub\wwwroot\chap10
Capítulo11	D:\Inetpub\wwwroot\chap11
Capítulo12	D:\Inetpub\wwwroot\chap12
Capítulo13	D:\Inetpub\wwwroot\chap13
Capítulo14	D:\Inetpub\wwwroot\chap14
Capítulo15	D:\Inetpub\wwwroot\chap15

Criarei os exemplos de cada capítulo dentro da respectiva pasta. Por exemplo, ao criar o exemplo 1 deste capítulo, vou salvá-lo na pasta D:\Inetpub\wwwroot\chap6, com o nome de chap6ex1.aspx. O exemplo 2 deste capítulo será salvo como chap6ex2.aspx e assim por diante.

Um detalhe importante é sobre o endereço para acessarmos e testarmos os exemplos de cada capítulo. Vamos considerar a página chap6ex1.aspx. Ao criarmos uma pasta, dentro do Home Directory, a pasta passa a fazer parte do endereço. Então, para acessarmos uma página que está dentro da pasta chap6, precisamos adicionar “chap6” ao endereço. Por último é só adicionar o nome da página a ser acessada. Para acessar a página chap6ex1.aspx, utilizariamos o seguinte endereço: <http://localhost/chap6/chap6ex1.aspx>.

Para acessarmos uma página chamada ex10chap13.aspx, gravada na pasta chap13, utilizamos o seguinte endereço: <http://localhost/chap13/ex10chap13.aspx>.

NOTA: As páginas ASP.NET têm a extensão .aspx ao invés de .asp. Falaremos mais sobre as diferenças do ASP.NET em relação ao ASP, no próximo item.

Lembrando sempre que localhost é uma referência ao servidor local, ou seja, na própria máquina onde estamos trabalhando. Ao invés de localhost podemos utilizar o nome DNS completo da máquina, no meu caso: <http://www.groza.com>.

Também cabe lembrar que, para que as páginas ASP.NET possam ser processadas, é necessário que você tenha instalado o Framework .NET conforme descrito na introdução, no seguinte item:

Como obter e instalar o Microsoft .NET Framework SDK.

IMPORTANTE: Se você estiver utilizando uma configuração diferente para testar os exemplos deste livro, utilize as configurações do equipamento que estiver utilizando em substituição às descritas neste item. Por exemplo, sempre que for solicitado acesso <http://www.groza.com>, substitua este endereço pelo endereço que você estiver utilizando.

ASP e ASP.NET Podem Rodar no Mesmo Servidor?

Sim. É importante salientar este ponto. Ao instalarmos o Framework .NET, estamos habilitando o suporte a páginas ASP.NET, porém as páginas ASP tradicionais continuam sendo processadas normalmente. Ou seja, continuamos tendo suporte ao código ASP. Desta maneira podemos migrar as aplicações já desenvolvidas, aos poucos para ASP.NET.

Podemos inclusive fazer com que um formulário de uma página ASP seja processado por uma página ASP.NET e vice-versa, isto tudo no mesmo servidor.

Enfim Vamos Iniciar com ASP.NET

Vamos conhecer as principais características de uma página ASP.NET.

A Extensão do Arquivo Mudou – .aspx

A primeira diferença que notamos é na extensão dos arquivos. As páginas ASP, desde a versão inicial até a versão 3.0, têm a extensão .asp. Esta extensão é obrigatória, para que o IIS reconheça a página como uma página ASP e passe o processamento para o interpretador ASP (ASP.DLL). Já as páginas ASP.NET têm a extensão .aspx, a qual é reconhecida pelo IIS (após a instalação do Framework .NET). O processamento da página ASP.NET é passado para o controle do Framework .NET. O serviço responsável pelo processamento das páginas ASP.NET está contido na DLL – XSPISAPI.DLL.

Inserindo Código ASP.NET

Para inserirmos código em uma página ASP.NET utilizamos a tag <SCRIPT>, com o atributo RUNAT igual a server. A linguagem padrão para a criação de páginas ASP.NET é o VB.NET. Observe bem que não é o VBScript e sim o Visual Basic .NET – VB.NET. No exemplo a seguir estamos definindo uma seção de código, na qual utilizaremos o VB.NET:

```
<script language="VB" runat="server">  
    comando1  
    comando2  
    ...  
    comandoN  
</script>
```

Na Listagem 6.2 colocamos a estrutura básica de uma página ASP.NET, onde temos uma seção de código e uma seção com os elementos que formam a parte visual, ou de apresentação da página.

Listagem 6.2 – A estrutura básica de uma página ASP.NET.

```
<html>  
<script language="VB" runat="server">  
    comando1  
    comando2  
    ...  
    comandoN  
</script>  
  
<body>  
    controles e  
    demais  
    elementos  
    da interface  
    - Server controls  
    - Web Forms  
    - etc  
</body>  
</html>
```

Toda página ASP.NET tem esta estrutura. Uma parte inicial onde temos o código, responsável pela lógica da página. Na seção de código, como é chamada, podemos colocar comandos para fazer a conexão com um banco de dados, para realizar cálculos, para responder a eventos que acontecem na página, como por exemplo um clique do usuário em um

botão, enfim, toda a lógica de programação necessária ao funcionamento da página. A seção de código é representada pelo seguinte trecho.

```
<script language="VB" runat="server">
    comando1
    comando2
    ...
    comandón
</script>
```

Na segunda parte da página, a partir da tag <body>, colocamos os elementos de apresentação da interface. Nesta parte podemos colocar desde código HTML básico, até controles mais avançados como os disponibilizados pelo ASP.NET. Nos Capítulos 7, 8 e 9 estaremos estudando os controles disponíveis com o ASP.NET.

Tudo o que você já conhece de HTML pode ser utilizado na seção de apresentação da página.

Se ao invés de utilizarmos o VB quiséssemos utilizar a linguagem C#, teríamos que definir a tag <script>, da seguinte maneira:

```
<script language="C#" runat="server">
    comando1
    comando2
    ...
    comandón
</script>
```

Observe que a estrutura de uma página ASP.NET deixa bem clara a separação entre código ASP.NET e código de apresentação, bem diferente do que acontecia com o ASP 3.0 onde seções de código ASP são intercaladas com seções de código HTML.

A estrutura apresentada na Listagem 6.2 é o que chamamos de “Code Inline”, ou seja, o código na própria página ASP.NET, embora em uma seção de código, separada da seção de apresentação.

NOTA: Nos exemplos deste livro estaremos utilizando o C#. Para uma introdução à linguagem C# consulte os Capítulos 3, 4 e 5.

Outra maneira de procedermos a separação entre código e apresentação é através da técnica chamada de “Code Behind”. Com esta técnica colocamos o código em um arquivo separado da página ASP.NET. Na página ASP.NET colocamos um comando para acessar o arquivo onde está o código.

Para utilizar Code Behind utilizamos o seguinte comando:

```
<%@Page Inherits="nome_da_classe" Src="Caminho para o arquivo com o código" %>
```

Por exemplo, vamos criar o código utilizando a linguagem C# e salvar o código em um arquivo chamado codex1.cs, na mesma pasta onde está a página ex1.aspx. No início da página ex1.aspx, devemos colocar o seguinte comando, para termos acesso ao código do arquivo codex1.cs:

```
<%@Page Inherits="codex1" Src="codex1.cs" %>
```

Esta deve ser a primeira linha da página, antes inclusive da tag <html>.

A utilização de Code Behind facilita o reaproveitamento de código. Vamos supor que estejamos criando uma série de páginas que acessam um banco de dados no SQL Server 2000. Podemos colocar o código de acesso ao banco de dados em um arquivo separado e utilizar a diretiva @Page para ter acesso a este código em cada página onde o acesso ao SQL Server 2000 for necessário. Se por algum motivo a forma de acesso tiver que ser modificada, basta alterar o arquivo com o código e, pronto, as páginas que fazem uso deste arquivo passarão a acessar a versão atualizada. Se a página já tiver sido acessada anteriormente e estiver no cache de páginas no servidor IIS, o Framework .NET detecta que houve mudanças em um dos elementos da página. Na primeira vez que a página for solicitada após as mudanças, uma nova versão da página será compilada e enviada para o usuário. A nova versão será mantida no cache para melhorar o desempenho. Se novas mudanças forem detectadas, o processo todo se repete.

Um Pequeno Exemplo, só Para Começar

Vamos apresentar alguns exemplos, para que você possa ver o ASP.NET em funcionamento. Neste momento não entrarei em maiores detalhes sobre o código. Irei detalhar mais a estrutura, a disponibilização e o acesso às páginas no servidor IIS.

Exemplo 1

Vamos criar um exemplo bastante simples. A nossa página contém dois controles do tipo texto, onde o usuário pode digitar informações. A página também contém um botão de comando. Ao clicar no botão de comando, é gerado o evento Click do respectivo botão. Vamos utilizar este evento. No evento Click do botão de comando vamos verificar os valores digitados nos campos usuário e senha e compará-los com valores previamente definidos, no próprio código do evento. Se os valores digitados forem iguais aos valores previamente definidos, emitimos uma mensagem: “LOGON EFETUADO COM SUCESSO !!!”; caso contrário emitimos a mensgem “LOGON FALHOU, TENTE NOVAMENTE!!!”.

Na Listagem 6.3 temos o código para a criação da página proposta.

Listagem 6.3 – O primeiro exemplo de página ASP.NET – chap6ex1.aspx.

```
<html>
```

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

    public void Botao_Click(Object sender, EventArgs e)
    {
        if (Nome.Value == "user1" && Senha.Value == "senha123")
        {
            Message.InnerHtml = "LOGON EFETUADO COM SUCESSO !!!!";
        }
    }
</script>
```

```

        }

    else
    {
        Message.InnerHtml = "LOGON FALHOU, TENTE NOVAMENTE !!!";
    }
}

</script>

<body>

<form method=post runat="server">

    <h3>Digite o nome:</h3>
    <input id="Nome" type=text size=40 runat="server">

    <h3>Digite a senha:</h3>
    <input id="Senha" type=password size=40 runat="server">
    <input type=submit value="Enter" OnServerClick="Botao_Click"
runat="server">

    <h1>
        <span id="Message" runat="server"> </span>
    </h1>
</form>

</body>
</html>

```

Digite o código da Listagem 6.3 e salve o mesmo em um arquivo chamado chap6ex1.aspx, na pasta chap6, dentro da pasta wwwroot, conforme descrito anteriormente. Para acessar esta página utilize o seguinte endereço:

<http://localhost/chap6/chap6ex1.aspx>

Ao carregarmos esta página no Internet Explorer obtemos o resultado indicado na Figura 6.5.

Digite user1 para o nome e senha123 para a senha. Dê um clique no botão Enter. Você receberá uma mensagem dizendo que o logon foi efetuado com sucesso. Agora digite jose para o nome e senha123 para a senha. Dê um clique no botão Enter. Você receberá uma mensagem informando que o logon falhou e que você deve tentar novamente, conforme indicado na Figura 6.6.

Sem detalhar o código vamos comentar os principais componentes do nosso exemplo.

- ◆ Criação da interface – Para criarmos a interface da página utilizamos alguns controles HTML chamados de “Controles HTML do Servidor – HTML Server Controls”. Observe o atributo runat=”server” destes controles, indicando que os mesmos são HTML Server Controls.

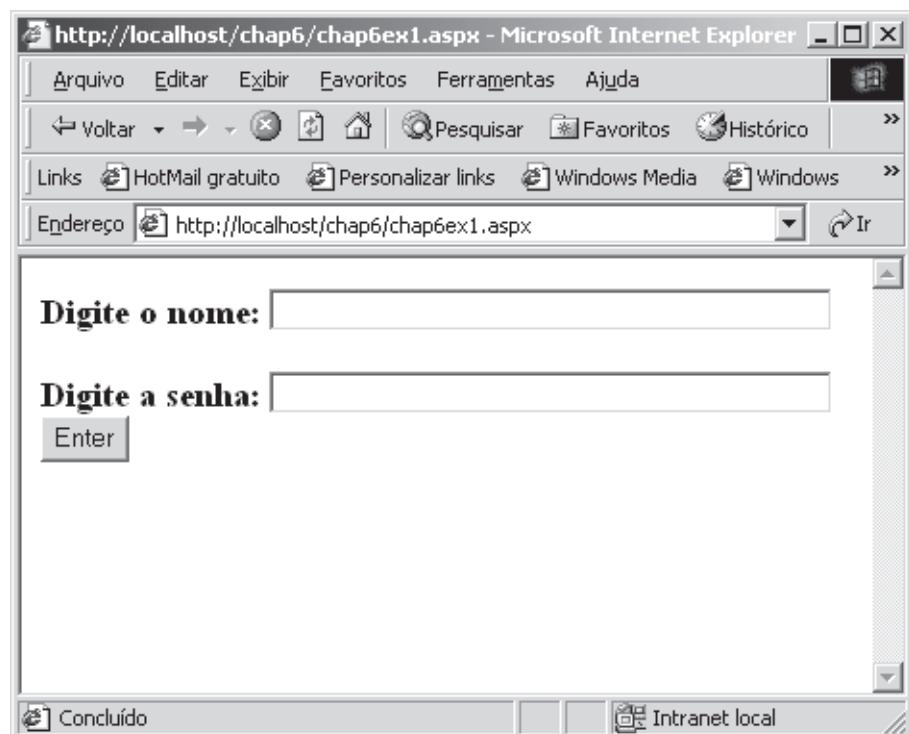


Figura 6.5: O primeiro exemplo – chap6ex1.aspx.

Por padrão, as tags HTML normais são tratadas como texto e não temos acesso às propriedades das mesmas, através de código. Com o uso de HTML Server Controls os controles HTML são tratados como elementos do servidor e podemos ter acesso a estes controles através de programação. No exemplo da Listagem 6.2 estamos acessando a propriedade Value dos controles Nome e Senha, para verificar os valores digitados pelo usuário. Não teríamos como acessar estas propriedades, utilizando os controles HTML tradicionais.

Por exemplo, para acessar o valor digitado no controle Nome, acessamos a propriedade Value deste controle, utilizando a seguinte sintaxe:

Nome.Value

- ◆ Código para responder a um evento.

A lógica do exemplo da Listagem 6.2 pode ser resumida da seguinte maneira:

“Quando o usuário clica no botão Enter é disparado o evento Click deste botão, o qual dispara o código contido no procedimento Botao_Click. Na própria definição do controle indicamos o nome do procedimento e do evento, conforme indicado a seguir:

```
<input type=submit value="Enter" OnServerClick="Botao_Click"
runat="server">
```

NOTA: Teremos um capítulo completo sobre HTML Server Controls – Capítulo 7.



Figura 6.6: Utilizando o primeiro exemplo – chap6ex1.aspx.

O OnServerClick indica o evento “Ao Clicar”, o qual é disparado quando o usuário clica no botão. O valor atribuído à propriedade OnServerClick, que no nosso exemplo foi Botao_Click, define o nome do procedimento que será executado em resposta ao evento. Este evento pode estar na Seção de Código da página, que é o caso do nosso exemplo, ou pode estar em uma classe herdada, caso estejamos utilizando Code Behind, conforme descrito anteriormente.

- ◆ A seção de código – Na seção de código da página chap6ex1.asp, logo após a tag <html>, temos o procedimento Botao_Click. Este procedimento está escrito em linguagem C#, conforme definido pela tag <script> no início da página:
- ```
<script language="C#" runat="server">
```

No procedimento Botao\_Click testamos os valores digitados pelo usuário nos campos Nome (Nome.Value) e Senha (Senha.Value). Dependendo dos valores utilizados retornamos uma ou outra mensagem. Observe que o código no procedimento Botao\_Click é código C# padrão. Temos um teste if, no qual são feitos dois testes ligados pelo operador lógico AND (&&).

Em resumo acontece o seguinte:

1. A página chap6ex1.aspx é carregada.
2. O usuário preenche os campos Nome e Senha e dá um clique no botão Enter.
3. O evento Click do botão é disparado.
4. Em resposta ao evento Click, o procedimento Botao\_Click é executado.

**NOTA:** Na prática, se estivéssemos criando um formulário para logon, o nosso evento Botao\_Click deveria fazer conexão com um banco de dados ou com o Active Directory do Windows 2000, para verificar as credenciais fornecidas. Se o nome de usuário e senha estiver correto, o acesso é liberado; caso contrário uma mensagem de erro é retornada para o usuário. Embora sem utilidade prática, o exemplo chap6ex1.aspx serve, didaticamente, para

5. Dependendo dos valores digitados pelo usuário, uma ou outra mensagem será retornada.

Agora que apresentamos um exemplo bastante simples, apenas para começarmos com o ASP.NET, vamos ver um exemplo que faz conexão com um banco de dados do SQL Server 2000.

## Banco de Dados do SQL Server 2000 Para os Exemplos Deste Livro

Ao instalarmos o Framework .NET é instalada uma miniversão do SQL Server 2000, na qual estão disponíveis diversos bancos de dados de exemplo. Se você já tiver o SQL Server 2000 instalado, as suas configurações serão mantidas. Para os exemplos deste livro estaremos utilizando os seguintes bancos de dados:

- ◆ pubs
- ◆ Northwind

Esta miniversão do SQL Server se instala como uma instância chamada NETSDK. Para fazer referência a esta instância do SQL Server, onde estão os bancos que utilizaremos nos exemplos deste livro, precisamos utilizar o nome do servidor e o nome da instância. Por exemplo, estou trabalhando em um computador cujo nome é SERVIDOR. Para fazer referência à instância NETSDK utilizo a seguinte nomenclatura:

**SERVIDOR\NETSDK**

Se o computador que você está utilizando tem o nome SERVNET, para fazer referência a instância NETSDK você deve utilizar a seguinte nomenclatura:

**SERVNET\NETSDK**

**NOTA:** Para um curso completo de SQL Server 2000 consulte o livro: "SQL SERVER 2000 Administração & Desenvolvimento – Curso Completo", de minha autoria, publicado pela editora Axcel Books ([www.axcel.com.br](http://www.axcel.com.br)).

## O Segundo Exemplo – Conectando com um Banco de Dados

O nosso segundo exemplo é um pouco mais complexo do que o exemplo anterior. Vamos apresentar uma página ASP.NET que faz conexão com um banco de dados do SQL Server e exibe todos os registros de uma determinada tabela.

A primeira diferença que você irá notar em relação ao ASP 3.0 é que não temos mais o objeto Recordset, o qual é bastante utilizado com o ASP 3.0, para conexões de páginas com bancos de dados.

Na Listagem 6.4 temos o código para a criação da página proposta.

**Listagem 6.4 – O segundo exemplo: conectando com o banco de dados pubs – chap6ex2.aspx.**

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

**NOTA:** A partir do Capítulo 10 estaremos tratando em detalhes da conexão de páginas ASP.NET com banco de dados.

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

 protected void Page_Load(Object Src, EventArgs E)
 {

 // Crio uma conexão com o banco de dados pubs localizado no servidor local.
 // Vamos acessar a instância SERVIDOR\NETSDK.

 SqlConnection myConnection = new SqlConnection("server=SERVIDOR\\NETSDK;" +
 " uid=sa;pwd=;database=pubs");

 // Conectamos com o banco de dados utilizando um comando SQL,
 // o qual retorna todos os dados da tabela "Authors", do banco de
 // dados pubs.

 SqlCommand myCommand = new SqlCommand("SELECT " +
 " * FROM Authors", myConnection);

 // Criamos e preenchemos um objeto DataSet.
 // Observe que não temos mais o objeto Recordset,
 // como era de praxe com o ASP 3.0.

 DataSet ds = new DataSet();
 myCommand.Fill(ds);

 // Conectamos um controle DataGrid com o DataSet criado anteriormente.
 // MyDataGrid é o id (nome) de um controle do tipo
 // DataGrid que está na seção de apresentação da página.

 DataView source = new DataView(ds.Tables[0]);
 MyDataGrid.DataSource = source ;
 MyDataGrid.DataBind();

 }

</script>

<body>
 <%-- Exibe as informações do DataGrid no corpo da página. --%>

 <h3>Registros da tabela Authors!</h3>
 <%-- Os registros da tabela Authors são exibidos no DataGrid. --%>
```

```

<ASP:DataGrid id="MyDataGrid" runat="server"
 Width="700"
 BackColor="#ccccff"
 BorderColor="black"
 ShowFooter="false"
 CellPadding=3
 CellSpacing="0"
 Font-Name="Verdana"
 Font-Size="8pt"
 HeaderStyle-BackColor="#aaaadd"
 MaintainState="false"
/>
</body>

</html>

```

**Registros da tabela Authors!**

| au_id       | au_lname       | au_fname    | phone        | address              | city           | state | zip   | contract |
|-------------|----------------|-------------|--------------|----------------------|----------------|-------|-------|----------|
| 172-32-1176 | White          | Johnson     | 408 496-7223 | 10932 Bigge Rd.      | Menlo Park     | CA    | 94025 | True     |
| 213-46-8915 | Green          | Marjorie    | 415 986-7020 | 309 63rd St. #411    | Oakland        | CA    | 94618 | True     |
| 238-95-7766 | Carson         | Cheryl      | 415 548-7723 | 589 Darwin Ln.       | Berkeley       | CA    | 94705 | True     |
| 267-41-2394 | O'Leary        | Michael     | 408 286-2428 | 22 Cleveland Av. #14 | San Jose       | CA    | 95128 | True     |
| 274-80-9391 | Straight       | Dean        | 415 834-2919 | 5420 College Av.     | Oakland        | CA    | 94609 | True     |
| 341-22-1782 | Smith          | Meander     | 913 843-0462 | 10 Mississippi Dr.   | Lawrence       | KS    | 66044 | False    |
| 409-56-7008 | Bennet         | Abraham     | 415 658-9932 | 6223 Bateman St.     | Berkeley       | CA    | 94705 | True     |
| 427-17-2319 | Dull           | Ann         | 415 836-7128 | 3410 Blonde St.      | Palo Alto      | CA    | 94301 | True     |
| 472-27-2349 | Gringlesby     | Burt        | 707 938-6445 | PO Box 792           | Covelo         | CA    | 95428 | True     |
| 486-29-1786 | Locksley       | Charlene    | 415 585-4620 | 18 Broadway Av.      | San Francisco  | CA    | 94130 | True     |
| 527-72-3246 | Greene         | Morningstar | 615 297-2723 | 22 Graybar House Rd. | Nashville      | TN    | 37215 | False    |
| 648-92-1872 | Blotchet-Halls | Reginald    | 503 745-6402 | 55 Hillsdale Bl.     | Corvallis      | OR    | 97330 | True     |
| 672-71-3249 | Yokomoto       | Akiko       | 415 935-4228 | 3 Silver Ct.         | Walnut Creek   | CA    | 94595 | True     |
| 712-45-1867 | del Castillo   | Innes       | 615 996-8275 | 2286 Cram Pl. #86    | Ann Arbor      | MI    | 48105 | True     |
| 722-51-5454 | DeFrance       | Michel      | 219 547-9982 | 3 Balding Pl.        | Gary           | IN    | 46403 | True     |
| 724-08-9931 | Stringer       | Dirk        | 415 843-2991 | 5420 Telegraph Av.   | Oakland        | CA    | 94609 | False    |
| 724-80-9391 | MacFeather     | Stearns     | 415 354-7128 | 44 Upland Hts.       | Oakland        | CA    | 94612 | True     |
| 756-30-7391 | Karsen         | Livia       | 415 534-9219 | 5720 McAuley St.     | Oakland        | CA    | 94609 | True     |
| 807-91-6654 | Panteley       | Sylvia      | 301 946-8853 | 1956 Arlington Pl.   | Rockville      | MD    | 20853 | True     |
| 846-92-7186 | Hunter         | Sheryl      | 415 836-7128 | 3410 Blonde St.      | Palo Alto      | CA    | 94301 | True     |
| 893-72-1158 | McBadden       | Heather     | 707 448-4982 | 301 Putnam           | Vacaville      | CA    | 95688 | False    |
| 899-46-2035 | Ringer         | Anne        | 801 826-0752 | 67 Seventh Av.       | Salt Lake City | UT    | 84152 | True     |
| 998-72-3567 | Ringer         | Albert      | 801 826-0752 | 67 Seventh Av.       | Salt Lake City | UT    | 84152 | True     |

Figura 6.7: Os registros da tabela Authors – chap6ex2.aspx.

Digite o código da Listagem 6.4 e salve o mesmo em um arquivo chamado chap6ex2.aspx, na pasta chap6, dentro da pasta wwwroot, conforme descrito anteriormente. Para acessar esta página utilize o seguinte endereço: <http://localhost/chap6/chap6ex2.aspx>

Ao carregarmos esta página no Internet Explorer obtemos o resultado indicado na Figura 6.7

Alguns comentários sobre o exemplo chap6ex2.aspx, sem entrar em maiores detalhes sobre o código do exemplo.

## Cadê o meu Objeto Recordset?

Nada de objetos do tipo Recordset. Com ASP.NET utilizamos ADO.NET para fazer a conexão com banco de dados. O ADO.NET utiliza o objeto Dataset ao invés do objeto Recordset. Para a conexão com bancos de dados utilizamos uma série de classes do namespace System.Data, o qual faz parte do .NET Framework Class Library.

Para ter acesso às classes de um determinado namespace, precisamos adicionar uma referência ao mesmo, no início da página ASP.NET, utilizando a diretiva Import, conforme indicado no trecho de código a seguir:

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

Observe que estamos fazendo referência a dois namespaces: System.Data e System.Data.SqlClient. Vamos tratar exaustivamente dos membros destes namespaces nos capítulos 10 e 11.

## Conectando com o Banco de Dados

Na seção de código da página temos os comandos para conexão com o banco de dados pubs da instância SERVIDOR\NETSDK. Um detalhe importante é a maneira como informamos o nome da instância, conforme indicado pelo trecho de código a seguir:

```
SqlConnection myConnection = new SqlConnection("server=SERVIDOR\\NETSDK;" +
 " uid=sa;pwd=;database=pubs");
```

Observe que utilizamos duas barras invertidas ao invés de uma única barra. Isto é necessário porque a barra invertida é considerada um caractere de escape com significado especial para a linguagem C#. Quando, ao invés do caractere de escape, queremos que o C# entenda o mesmo como um caractere normal, temos que precedê-lo de uma barra invertida. Na prática as duas barras invertidas são interpretadas, pelo C#, como uma única barra, o que produz o resultado desejado: SERVIDOR\NETSDK.

Neste exemplo utilizamos as seguintes classes do namespace System.Data.SqlClient:

- ◆ SqlConnection
- ◆ SqlDataAdapter

Também utilizamos as seguintes classes do namespace System.Data:

- ◆ DataSet
- ◆ DataView

## Um Controle Poderoso Para Exibir os Dados

Na seção de apresentação utilizamos o controle DataGrid para fazer a exibição dos registros da tabela Authors. Observe o quanto poderoso é este controle. Simplesmente através da configuração da sua propriedade DataSource, na seção de código, definimos os registros a serem exibidos pelo controle. Esta definição é feita nas seguintes linhas de código:

```
MyDataGrid.DataSource = source ;
MyDataGrid.DataBind();
```

Veja que não precisamos utilizar um laço While para percorrer todos os registros da tabela Authors, como tínhamos que fazer com o objeto RecordSet.

O controle DataGrid faz parte dos chamados “Web Server Controls”. No Capítulo 8 aprenderemos a utilizar os diversos Web Server Controls disponíveis.

## Posso Ter um Comando que Ultrapassa uma Linha?

A resposta é um sonoro Sim.

Diferente do ASP 3.0, em nossas páginas ASP.NET, os comandos da seção de código podem estar em mais do que uma linha, sem problemas. Inclusive na Listagem 6.3 temos comandos que ultrapassam uma linha e a página carrega sem maiores problemas, conforme exemplificado no trecho a seguir:

```
SqlDataAdapter myCommand = new SqlDataAdapter("SELECT " +
 " * FROM Authors", myConnection);
```

Isso é possível pois cada comando do C# é finalizado com um ponto-e-vírgula (;).

## O Bom e Velho SQL Continua o Mesmo?

A resposta é outro sonoro Sim.

Você deve ter observado que utilizamos um comando SQL para retornar os dados da tabela authors. Utilizamos um comando SELECT, conforme indicado a seguir:

```
SqlConnection myConnection = new SqlConnection("server=SERVIDOR\\NETSDK;" +
 " uid=sa;pwd=;database=pubs");
```

Vamos alterar o comando SQL para introduzir uma condição. Queremos que sejam retornados somente os registros para os autores em que a cidade (city) é Oakland. Para isso basta que seja utilizada uma cláusula where para especificar a condição city='Oakland', conforme indicado a seguir:

```
SqlDataAdapter myCommand = new SqlDataAdapter("SELECT " +
 " * FROM Authors where city='Oakland'", myConnection);
```

Faça esta alteração na página chap6ex2.aspx e salve a nova versão da página na pasta chap6, com o nome de chap6ex3.aspx. Ao carregar esta nova versão, obteremos o resultado indicado na Figura 6.8.

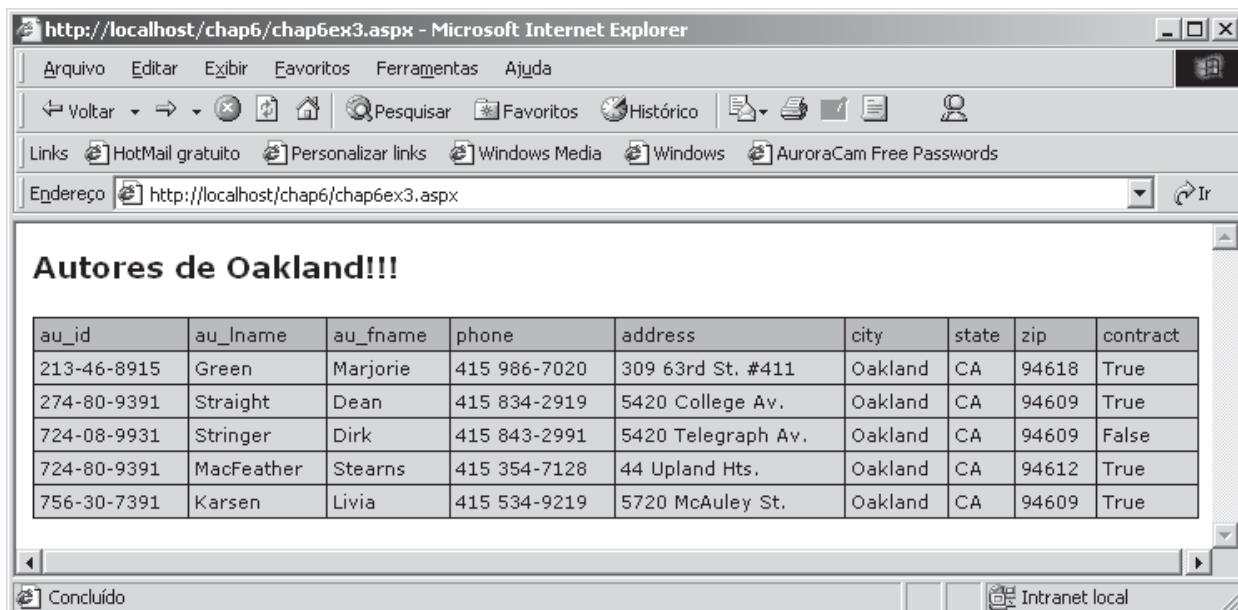


Figura 6.8: Definindo um critério com a cláusula where – chap6ex3.aspx.

## Conclusão

Neste capítulo apresentamos o ASP.NET.

Iniciamos o capítulo justificando o porquê de mais uma versão para a tecnologia ASP – Active Server Pages. Vimos que ASP.NET não é apenas mais uma versão, mas sim uma mudança de paradigma, uma vez que com ASP.NET temos acesso a todas as vantagens do Framework .NET. Dentre as principais vantagens do ASP.NET, destacamos:

- ◆ O Framework .NET.
- ◆ Suporte a múltiplas linguagens.
- ◆ O programador precisa escrever menos código.
- ◆ Separação entre o código HTML e o código ASP.NET.
- ◆ Maior facilidade na criação e utilização de componentes, o que facilita a reutilização de código.
- ◆ Um conjunto de controles avançados:
  1. HTML Server Controls.
  2. Web Server Controls.
  3. Validation Server Controls.

**NOTA:** No Anexo III apresento uma revisão dos conceitos básicos da linguagem SQL – Structured Query Language. Antes de estudar os capítulos 10 e 11 seria interessante dar uma olhada no Anexo II – O modelo de dados relacionais, e no Anexo III – Fundamentos da linguagem SQL.

Em seguida fizemos um Check List para verificar se o computador que você está utilizando está apto a rodar páginas ASP.NET. Também falamos sobre o Home Directory do servidor IIS e como acessar páginas gravadas em subpastas do Home Directory.

Na seqüência apresentamos a estrutura básica de uma página ASP.NET. Também aprendemos a definir uma seção de código através da tag <script>.

Para finalizar o capítulo apresentamos dois exemplos, sem detalhar o código dos exemplos. No primeiro exemplo apresentamos alguns HTML Server Controls. No segundo exemplo utilizamos algumas classes do Framework .NET, mais especificamente dos namespaces System.Data e System.Data.SqlClient, para fazer acessar os dados do banco de dados pubs do SQL Server 2000.

No próximo capítulo estudaremos os HTML Server Controls disponíveis. Veremos um a um, todos os controles disponíveis, bem como as propriedades de cada um. Também apresentaremos alguns exemplos práticos para ilustrar o uso dos HTML Server Controls.

## Introdução

Uma das inovações mais bem-vindas do ASP.NET é a disponibilização de uma série de controles mais “poderosos”, flexíveis e “inteligentes”. Com ASP.NET temos acesso a diversos controles que permitem a criação de páginas com uma aparência gráfica mais parecida com os programas tradicionais do Windows, além de termos acesso a um rico modelo de eventos, o que já estava disponível em aplicações tradicionais do Windows, a um bom tempo.

Com as versões anteriores, criar páginas com uma aparência gráfica mais elaborada não era tarefa das mais fáceis. Para alcançar uma aparência mais profissional, tínhamos que lançar mão de uma série de tecnologias diferentes, o que era bastante trabalhoso. Com o ASP.NET temos uma série de elementos, conhecidos como controles de servidor, que nos ajudam na tarefa de criar páginas com uma aparência mais profissional, e que são exibidas da mesma maneira nos diferentes navegadores.

Com as tags HTML tradicionais, o que temos são elementos estáticos em uma página. Em outras palavras, uma tag do tipo HTML <INPUT>, colocada em um formulário HTML tradicional, utilizada para que o usuário faça uma entrada de texto, é um elemento estático. Não existem eventos para este campo e não temos como acessar as propriedades do campo através de programação. Com o modelo de página das versões anteriores do ASP, tínhamos algumas limitações. O exemplo de limitação mais típico que tínhamos na criação de páginas ASP era a situação na qual precisávamos que a página fosse recarregada em períodos determinados. Por exemplo, uma página que fornece informações sobre cotações de ações precisa ser atualizada de minuto em minuto. Para atualizar a página, o usuário precisava utilizar o comando de atualização do seu Browser. Já com ASP.NET e o modelo baseado em eventos podemos fazer com que seja disparado um evento, por exemplo, de minuto em minuto. No código do evento podemos inserir os comandos necessários para que a página seja recarregada, automaticamente, com a versão mais atualizada, de minuto em minuto.

Com a utilização dos controles de servidor (Server Controls) temos uma mudança na maneira como criamos nossas páginas. O novo modelo utiliza a programação baseada em eventos. Ações que acontecem em um ou mais controles da página podem ser detectadas pelo servidor e uma ação pode ser tomada em resposta ao evento. Conforme você já deve ter notado, é o tradicional modelo de programação baseado em eventos, só que agora trazido para o mundo do desenvolvimento de aplicações e páginas Web.

Existem diversos tipos de controles de servidor, dentre os quais podemos destacar os seguintes tipos:

- ◆ HTML Server Controls.
- ◆ Validation Server Controls.
- ◆ Web Server Controls.

Neste capítulo vamos estudar os HTML Server Controls. Vamos apresentar os controles existentes, bem como as diversas propriedades de cada controle. Também vamos apresentar uma série de páginas que ilustram a utilização destes controles.

Antes de iniciarmos com a apresentação dos controles, veremos o que mudou na maneira como uma página é carregada no servidor e disponibilizada para o usuário. Veremos que o modelo das páginas mudou significativamente no ASP.NET, em relação às versões anteriores. Também falaremos sobre o cache automático de páginas no servidor.

## Uma Classe Chamada Page

A classe Page faz parte do namespace System.Web.UI. Este namespace possui uma série de classes e interfaces, as quais permitem a criação de controles e páginas, os quais formam a interface com o usuário em nossas aplicações Web. Temos, por exemplo, uma classe chamada Control, a qual disponibiliza todos os controles que podem ser utilizados em páginas ASP.NET. Também existe um controle chamado Page, o qual é automaticamente criado toda vez que é feita a requisição de uma página ASP.NET. Fazer a requisição significa acessar a página. Também estão disponíveis classes que nos possibilitam a ligação de dados com controles de um formulário e a manutenção de estado entre diferentes requisições à mesma página.

## Eventos ao Carregar uma Página ASP.NET

Quando acessamos uma página .aspx, como por exemplo: <http://www.microsoft.com/net/default.aspx>, uma série de eventos é disparada. É semelhante ao carregamento de uma aplicação Windows tradicional. Ao carregarmos uma aplicação Windows desenvolvida, por exemplo, em Visual Basic, são disparados diversos eventos. Por exemplo, ao carregar o formulário principal da aplicação, é disparado o evento OnLoad; ao exibir o formulário na tela é disparado o evento OnActivate e assim por diante. Ao carregarmos uma página .aspx são disparados, em seqüência, os seguintes eventos:

- ◆ Page\_Init: Disparado quando a página é inicializada.
- ◆ Page\_Load: Disparado quando a página é carregada.
- ◆ Contorl Event: É disparado se um evento associado com um controle da página faz com que a mesma tenha que ser recarregada.
- ◆ Page\_Unload: Disparado quando a página é retirada da memória.

A diferença básica entre os eventos Page\_Init e Page\_Load é em relação aos controles da página. Quando o evento Page\_Init dispara, os controles da página ainda estão com seus valores padrão, pois o estado dos mesmos, caso tenham sido alterados, ainda não foi carregado. Já quando dispara o evento Page\_Load, os controles estão com seus estados atualizados, sendo que já temos acesso aos reais valores de cada controle.

Esta seqüência de eventos reforça a idéia de que o modelo de programação baseado em eventos foi transportado para o mundo das aplicações Web, pelo ASP.NET.

## A Classe Page

Toda página .aspx acessada a partir de um servidor onde está instalado o Framework .NET, quer a página contenha código ASP.NET ou somente código HTML, é compilada e é criado um objeto do tipo Page, o qual fica armazenado no cache de memória do servidor. Em outras palavras, ao acessarmos uma página .aspx, a mesma é compilada, sendo gerada uma instância da classe Page, sendo esta instância armazenada no cache do servidor para melhorar a velocidade de acesso à página. Qualquer alteração na página é, automaticamente, detectada pelo Framework .NET, a nova versão é compilada e armazenada no cache de memória do servidor, em substituição à versão anterior.

Na classe Page estão definidos as propriedades, métodos e eventos comuns a todas as páginas processadas pelo Runtime do ASP.NET. A classe Page funciona como um Container para todos os componentes que fazem parte da página. Como analogia, nas aplicações Windows, temos a figura do Formulário (Form), o qual é um Container para todos os elementos da aplicação.

## Os Eventos da Classe Page

Na Tabela 7.1 temos a descrição dos eventos da classe Page.

**Tabela 7.1** Eventos da classe Page.

| Evento            | Descrição                                                                                                                                    |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| AbortTransaction  | Ocorre quando uma transação é cancelada.                                                                                                     |
| CommitTransaction | Ocorre quando uma transação é finalizada com sucesso.                                                                                        |
| DataBinding       | Ocorre quando um controle de servidor (Server Control) é vinculado a uma fonte de dados.                                                     |
| Error             | Ocorre quando é detectada uma exceção que não foi tratada.                                                                                   |
| Init              | Ocorre quando a página é inicializada.                                                                                                       |
| Load              | Ocorre quando a página é carregada e todos os seus controles tiverem sido carregados.                                                        |
| PreRender         | Ocorre antes que qualquer informação seja enviada para o navegador do cliente.                                                               |
| Unload            | Ocorre quando o processamento da página é finalizado. Isto ocorre após todas as informações terem sido enviadas para o navegador do cliente. |

Vamos a um exemplo no qual utilizamos o evento Load da página, para emitir uma mensagem.

Na Listagem 7.1 temos um exemplo de página, onde definimos o valor de um controle da página – controle Mensagem, durante o evento Load da página.

**Listagem 7.1 – Utilizando o evento Load – chap7ex1.aspx.**

```

<html>

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

 void Page_Load(Object Src, EventArgs E)
 {
 Mensagem.Value = "Último acesso em: " + DateTime.Now;
 }
</script>

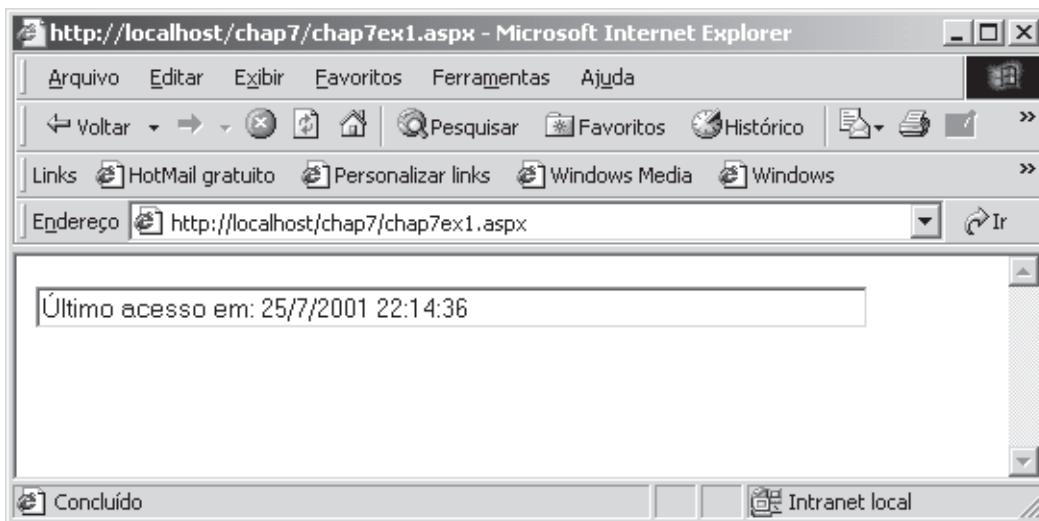
<body>
 <input id="Mensagem" type=text size=60 runat="server">
</body>

</html>

```

Digite o código da Listagem 7.1 e salve o mesmo em um arquivo chamado chap7ex1.aspx, na pasta chap7, dentro da pasta wwwroot, conforme descrito anteriormente. Para acessar esta página utilize o seguinte endereço: <http://localhost/chap7/chap7ex1.aspx>

Ao carregarmos esta página no Internet Explorer obtemos o resultado indicado na Figura 7.1.



**Figura 7.1: O evento Load da página – chap7ex1.aspx.**

Em primeiro lugar quero reforçar o fato de termos à disposição um modelo de programação baseado em eventos, o que para o ambiente Web é um avanço significativo em relação às versões anteriores.

O segundo detalhe para o qual gostaria de chamar a atenção é a utilização de um controle de servidor. Mais especificamente, utilizamos um HTML Server Control, para exibir a mensagem definida no evento Load da página. Utilizamos o controle Input, conforme indicado no trecho de código a seguir:

```
<input id="Mensagem" type="text" size=60 runat="server">
```

Observe a propriedade runat="server" definindo que este é um controle de servidor e que, portanto, temos acesso às suas propriedades através do código da página. Foi exatamente isto que fizemos no evento Load, quando definimos o valor da propriedade Load do evento Mensagem, conforme indicado a seguir:

```
void Page_Load(Object Src, EventArgs E)
{
 Mensagem.Value = "Último acesso em: " + DateTime.Now;
}
```

Podemos utilizar os eventos da classe Page, para uma série de funções, tais como:

- ◆ Código para conexão com banco de dados.
- ◆ Código para verificação das permissões de acesso.
- ◆ Código para configuração dos controles da página.

## As Propriedades da Classe Page

Na Tabela 7.2 temos a descrição das propriedades da classe Page.

**Tabela 7.2 Propriedades da classe Page.**

Propriedade	Descrição
Application	Faz referência a um objeto do tipo Application. Um único objeto do tipo Application é criado para cada aplicação Web. A instância do objeto Application é compartilhada por todos os clientes que acessam páginas da aplicação Web.
Cache	Retorna uma referência a um objeto do tipo Cache, o qual pode ser utilizado para armazenar dados que são utilizados nas próximas solicitações da página. O objeto cache utiliza campos do tipo oculto para manter o estado dos elementos de uma página entre uma chamada e outra da página.
ClientTarget	Esta propriedade permite alterar a detecção automática do Browser do cliente, que é feita pelo Framework .NET, e especificar o Browser que deve ser considerado para a montagem da página de retorno para o usuário.
EnableViewState	É um valor Booleano (True ou False). O valor desta propriedade indica se os controles da página devem ou não manter o seu estado entre uma chamada e outra da página. Esta propriedade afeta todos os controles da página. Por padrão é definida em True.

Propriedade	Descrição
ErrorPage	Pode ser utilizada para retornar ou definir a página que deve ser carregada, caso seja detectada uma exceção não tratada.
IsPostBack	É um valor Booleano. Se for verdadeiro (True), indica que a página está sendo recarregada pelo usuário, o que é conhecido como um “round-trip” (ida e volta). Quando for False, significa que a página está sendo carregada pela primeira vez e, consequentemente, não existem valores de estado armazenados no cache do servidor, para os controles da página. Podemos utilizar esta propriedade para detectar se está sendo feita a carga inicial da página e, em caso afirmativo, utilizar o evento Page_Load para definir os valores iniciais, para os controles da página.
IsValid	Com o ASP.NET temos os chamados Validation Controls. Estes controles são capazes de fazer a validação dos dados digitados em um formulário. Se a validação de todos os controles ocorrer com sucesso, a propriedade IsValid é definida como True; caso contrário, a propriedade será definida como False. Se a página não tiver nenhum controle de validação, a propriedade retornará True.
Request	Permite que tenhamos acesso aos dados enviados com a requisição HTTP da página. Esta propriedade retorna uma referência ao objeto Request. Estudaremos este objeto em detalhes, ainda neste capítulo.
Response	Permite que tenhamos acesso aos dados enviados com HTTP response da página. Esta propriedade retorna uma referência ao objeto Response. Um Response está, geralmente, associado com um formulário. Estudaremos este objeto em detalhes, ainda neste capítulo.
Server	Retorna uma referência ao objeto Server.
Session	Retorna uma referência ao objeto Session.
TraceEnabled	É uma propriedade Booleana. Retorna True se o Trace para a página estiver habilitado e False, em caso contrário. O objeto Trace é utilizado para depuração de páginas.
Trace	Retorna uma referência ao objeto Trace, caso o mesmo tenha sido habilitado.
User	Obtém informações a respeito do usuário que está fazendo a requisição da página.
Validators	Retorna uma coleção que contém todos os controles de validação da página.

Vamos a um exemplo no qual exibimos o valor de diversas propriedades da classe Page.

Na Listagem 7.2 temos um exemplo de página, onde são exibidos os valores de diversas propriedades da classe page.

#### Listagem 7.2 – Propriedades da classe Page – chap7ex2.aspx.

```
<%@ Import Namespace="System" %>
<html>
<script language="C#" runat="server">
```

```

void Page_Load(Object Src, EventArgs E)
{
 PrApplication.Value = Convert.ToString(Page.Application);
 PrCache.Value = Convert.ToString(Page.Cache);
 PrClientTarget.Value = Convert.ToString(Page.ClientTarget);
 PrEnableViewState.Value = Convert.ToString(Page.EnableViewState);
 Page.ErrorPage = "http://localhost/erros/erro.aspx";
 PrErrorMessage.Value = Convert.ToString(Page.ErrorMessage);
 PrIsPostBack.Value = Convert.ToString(Page.IsPostBack);
 PrIsValid.Value = Convert.ToString(Page.IsValid);
 PrRequest.Value = Convert.ToString(Page.Request);
 PrResponse.Value = Convert.ToString(Page.Response);
 PrServer.Value = Convert.ToString(Page.Server);
 PrSession.Value = Convert.ToString(Page.Session);
 PrTrace.Value = Convert.ToString(Page.Trace);
 PrUser.Value = Convert.ToString(Page.User);
}

</script>
<body>

<H1>Propriedades da classe Page !!! </H1>

<TABLE>

<TR>
 <TD>
 Propriedade
 </TD>
 <TD>
 Valor retornado.
 </TD>
</TR>

<TR>
 <TD>
 Application:
 </TD>

```

```
</TD>

<TD>

 <input id="PrApplication" type="text" size=60 runat="server">

</TD>

</TR>

<TR>

<TD>

 Cache:

</TD>

<TD>

 <input id="PrCache" type="text" size=60 runat="server">

</TD>

</TR>

<TR>

<TD>

 ClientTarget:

</TD>

<TD>

 <input id="PrClientTarget" type="text" size=60 runat="server">

</TD>

</TR>

<TR>

<TD>

 EnableViewState:

</TD>

<TD>

 <input id="PrEnableViewState" type="text" size=60 runat="server">

</TD>

</TR>

<TR>

<TD>

 ErrorPage:

</TD>
```

```
<TD>

 <input id="PrErrorPage" type="text" size=60 runat="server">

</TD>

</TR>

<TR>

<TD>

 IsPostBack:

</TD>

<TD>

 <input id="PrIsPostBack" type="text" size=60 runat="server">

</TD>

</TR>

<TR>

<TD>

 IsValid:

</TD>

<TD>

 <input id="PrIsValid" type="text" size=60 runat="server">

</TD>

</TR>

<TR>

<TD>

 Request:

</TD>

<TD>

 <input id="PrRequest" type="text" size=60 runat="server">

</TD>

</TR>

<TR>

<TD>

 Response:

</TD>

<TD>

 <input id="PrResponse" type="text" size=60 runat="server">
```

```
</TD>

</TR>

<TR>
<TD>
Server:
</TD>
<TD>
<input id="PrServer" type="text" size=60 runat="server">
</TD>
</TR>

<TR>
<TD>
Session:
</TD>
<TD>
<input id="PrSession" type="text" size=60 runat="server">
</TD>
</TR>

<TR>
<TD>
Trace:
</TD>
<TD>
<input id="PrTrace" type="text" size=60 runat="server">
</TD>
</TR>

<TR>
<TD>
User:
</TD>
<TD>
<input id="PrUser" type="text" size=60 runat="server">
```

```

</TD>
</TR>

</TABLE>

</body>
</html>

```

Digite o código da Listagem 7.2 e salve o mesmo em um arquivo chamado chap7ex2.aspx, na subpasta chap7, dentro da pasta wwwroot, conforme descrito anteriormente. Para acessar esta página utilize o seguinte endereço: <http://localhost/chap7/chap7ex2.aspx>

Ao carregarmos esta página no Internet Explorer obtemos o resultado indicado na Figura 7.2.

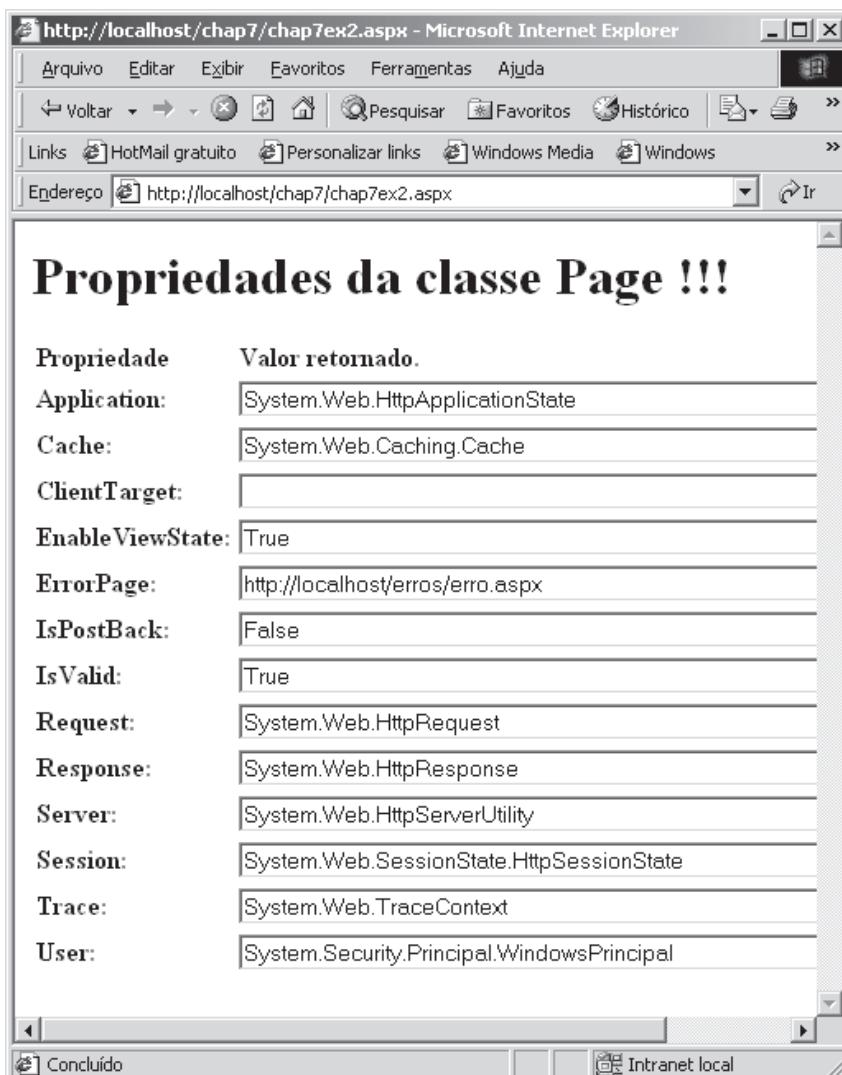


Figura 7.2: Propriedades da classe Page – chap7ex2.aspx.

Alguns comentários sobre o exemplo.

No início da página utilizamos uma diretiva Import:

```
<%@ Import Namespace="System" %>
```

Utilizamos esta diretiva para termos acesso à classe Convert, do namespace System. Utilizamos o método ToString da classe Convert – Convert.ToString, para converter os valores retornados pelas propriedades da classe Page, em valores de texto, e atribuir os valores de texto para controles do tipo input na seção de apresentação da página.

Na seção de código acessamos os valores para as propriedades da classe page. Por exemplo, para a propriedade Application, utilizamos o seguinte código:

```
PrApplication.Value = Convert.ToString(Page.Application);
```

Page.Application retorna o valor da propriedade Application. O valor retornado é convertido para String pelo método Convert.ToString. O valor já convertido para String é atribuído à propriedade Value do controle PrApplication, na seção de apresentação da página. O resultado prático disso tudo é que o valor da propriedade Application será exibido no controle PrApplication, na seção de apresentação.

Na seção de apresentação utilizamos uma tabela para exibir os controles melhor alinhados. Para isso utilizamos as tags do HTML para construção de tabelas:

- ◆ <TABLE> </TABLE> para a criação da tabela.
- ◆ <TR> </TR> para a criação de uma nova linha.
- ◆ <TD> </TD> para a criação de uma nova célula em cada linha.

**NOTA:** Para maiores detalhes sobre os comandos básicos da linguagem HTML consulte o Anexo I.

Para acessar as propriedades do objeto page, utilizamos a sintaxe padrão para acesso aos componentes de um objeto:

```
nome_do_objeto.nome_componente
```

Por exemplo, para acessar a propriedade Cache, do objeto page, utilizamos a seguinte sintaxe:

```
Page.Cache.
```

## Métodos da Classe Page

Na Tabela 7.3 temos a descrição dos métodos da classe Page.

**IMPORTANTE:** Quando uma página .aspx é carregada, a mesma é compilada como um objeto que é uma instância da classe Page. Sendo um objeto, temos acesso a suas propriedades, métodos e eventos.

**Tabela 7.3** Métodos da classe Page.

Métodos	Descrição
<b>DataBind</b>	Executa uma operação de DataBind para todos os controles da página.
<b>FindControl</b>	Permite que localizemos um determinado controle, na página.
<b>MapPath</b>	Retorna o caminho físico para um determinado diretório virtual.
<b>Validate</b>	Faz com que os controles de validação da página façam a validação de seus conteúdos.

## O Processamento de uma Página ASP.NET

Vamos analisar mais alguns detalhes sobre a maneira como uma página ASP.NET é processada no servidor e retornada para o usuário.

Para acessar uma página, o usuário digita o endereço da página no seu navegador, como por exemplo: <http://www.juliobattisti.com.br/certificacao.aspx>.

O navegador envia a requisição da página para o servidor IIS, através de uma requisição HTTP (HTTP Request). O servidor localiza os dados solicitados, envia de volta para o usuário e fecha a conexão. Este aspecto básico não mudou, até porque faz parte da definição do protocolo HTTP, independente do conteúdo que está sendo carregado. Porém, com ASP.NET, tivemos muitas melhorias em relação à maneira como a página é carregada e como as informações são mantidas entre uma requisição e outra.

Com o modelo atual, parte do processamento acontece no navegador do cliente e parte no servidor. Normalmente no cliente são colocados os elementos visuais da página. No servidor são feitos os processamentos do código de script da página. Porém o processamento é do tipo “tudo de uma só vez”, ou seja, o cliente envia a requisição, o servidor processa todos os elementos da página, formata uma saída e retorna a saída formatada para o cliente. Para que possa haver um novo processamento, uma nova requisição da página precisa ser feita.

Com ASP.NET temos uma mudança neste modelo, devido à criação dos chamados Server Controls. Um controle do tipo Server Control fornece elementos visuais para uma página ASP.NET, ao mesmo tempo que é processado no servidor. Estes controles têm, inclusive, um modelo de eventos que pode ser acessado via código. Com isso temos disponível um modelo de desenvolvimento baseado em eventos, semelhante ao desenvolvimento de aplicações Windows tradicionais.

Outra mudança que é possível, graças aos Server Controls, é em relação à iteração entre a página e o servidor. Como os Server Controls possuem eventos, podemos escrever código para ser executado em resposta a estes eventos. Este código de eventos é processado no servidor. Dentro de uma mesma requisição de página, podem ser disparados diversos eventos, à medida que o usuário vai trabalhando na página. Para cada evento, o código respectivo, caso exista, é processado no servidor. Observem que o cliente fica “indo e voltando” do servidor, para processar o código associado a eventos. Cada ida e vinda equivale a uma requisição. Este processo de ida e vinda é conhecido como “server round-trip.”

Para que um round-trip seja disparado, basta que o usuário realize alguma ação na página, ação esta que dispara um evento. Por exemplo, clicar em um campo para entrada de texto, para provocar um evento Click do respectivo controle.

## Manutenção de Estado é Fundamental

Conforme descrevemos no item anterior, o processo de round-trip faz com que diversas requisições sejam enviadas para uma mesma página .aspx. Se para cada requisição, toda a página tivesse que ser novamente processada, o desempenho deixaria muito a desejar. Além disso o protocolo HTTP é stateless, ou seja, não é mantida nenhuma informação entre uma requisição e outra. Para resolver este problema e manter o estado entre requisições, o ASP.NET utiliza um processo conhecido como ViewState, para manter todas as informações da página entre uma requisição e outra. O ViewState contém o estado (valores, propriedades, aparência, etc.) para todos os controles de usuário de uma página ASP.NET. A informação de estado é armazenada na forma de pares de valores. O objeto responsável pela manutenção de estado é derivado da classe System.Web.UI.StateBag. Todas as informações de estado são armazenadas como uma string, a qual é enviada de volta para o cliente. O navegador do cliente recebe a string com a informação sobre os controles e configura cada um com os mesmos valores que tinham no momento da requisição. A string com as informações de estado são retornadas na forma de um campo oculto, o qual, evidentemente, não é exibido na página. Podemos ver o conteúdo desta string, verificando o código-fonte da página que é retornada. Por exemplo, no IE, podemos utilizar o comando Exibir -> Código-fonte.

Conforme descrevemos anteriormente, o processo de ViewState é habilitado por padrão. Se por algum motivo quisermos desabilitar este processo, podemos utilizar a propriedade EnableViewState. Para desabilitar o mecanismo de ViewState utilize o seguinte comando no início da página:

```
<%@Page EnableViewState="false" %>
```

Com o mecanismo de round-trip e ViewState, o ASP.NET traz para o mundo stateless do protocolo HTTP um modelo capaz de fazer a manutenção de estado para aplicações Web. Este é um grande avanço, pois com as versões anteriores o programador era obrigado a fazer verdadeiros “malabarismos” para conseguir uma manutenção de estado eficiente.

Com ASP.NET o próprio Framework .NET nos disponibiliza mecanismos eficientes para a manutenção de estado. Esta é mais uma das características que comprovam o objetivo do Framework .NET em oferecer o máximo de funcionalidade, deixando para o desenvolvedor apenas o código relacionado com a lógica e a apresentação da aplicação.

**IMPORTANTE:**Também podemos desabilitar o mecanismo de ViewState em nível de controle. Veremos isso mais adiante.

## HTML Server Controls

Agora que já conhecemos um pouco mais sobre o processamento de páginas ASP.NET e sobre a classe page, estamos prontos para começar a estudar os HTML Server Controls. Vamos iniciar o nosso estudo entendendo o que são HTML Server Controls.

## Uma Definição Para HTML Server Controls

Em primeiro lugar são controles de servidor, isto é, são processados no servidor. Para definir um controle como sendo de servidor, devemos definir a sua propriedade runat como sendo igual a server, conforme o exemplo a seguir:

```
<input id="Senha" type=password size=40 runat="server">
```

A propriedade runat="server" faz com que o controle seja compilado juntamente com a página e executado, no servidor, cada vez que a página é requisitada. Observe que esta forma de processamento demanda mais recursos do servidor do que os controles HTML tradicionais.

Em segundo lugar os controles disponibilizam um modelo de programação baseado em eventos. Esta é uma mudança importante em relação às versões anteriores. Por exemplo, quando o usuário clica em um botão Enviar, os valores do formulário são enviados para o servidor e o evento Click do botão é disparado. Podemos escrever o código que executa quando o evento é disparado. Nós já fizemos uso desta técnica no Exemplo da Listagem 6.1 – chap6ex1.aspx.

Na seção de apresentação do exemplo citado, temos um controle do tipo Submit – um botão de comando. Este controle é criado com a linha de código a seguir:

```
<input type=submit value="Enter" OnServerClick="Botao_Click" runat="server">
```

Observe a propriedade OnServerClick="Botao\_Click". Esta propriedade define o nome do procedimento que será executado, no servidor, em resposta ao evento Click do botão.

Na seção de código da página criamos um procedimento chamado Botao\_Click, o qual é executado em resposta ao evento Click do botão de comando, conforme indicado no código a seguir:

```
<script language="C#" runat="server">
 public void Botao_Click(Object sender, EventArgs e)
 {
 if (Nome.Value == "user1" && Senha.Value == "senha123")
 {
 Message.InnerHtml = "LOGON EFETUADO COM SUCESSO !!!!";
 }
 else
 {
 Message.InnerHtml = "LOGON FALHOU, TENTE NOVAMENTE !!!";
 }
 }
</script>
```

Cada controle de servidor é considerado, pelo Framework .NET, como um objeto da página. Como um objeto, o controle possui propriedades, métodos e eventos, os quais são acessíveis através do código de programação.

HTML Server controls devem ser colocados dentro de um formulário, na página ASP.NET. Criamos um formulário, conforme veremos mais adiante, com as tags <FORM> </FORM>. Além disso, a propriedade runat, do formulário, deve ser definida como: runat="server".

Como a utilização de Server Controls exige mais recursos do servidor do que o uso de controles tradicionais, somente devemos utilizar Server Controls, quando as suas características forem necessárias. Porém com o avanço e sofisticação das aplicações Web, vai ser difícil acharmos uma situação em que possamos dispensar o modelo baseado em eventos, disponibilizado pelos Server Controls.

Em algumas situações simples pode ser dispensável a utilização de Server Controls. Como por exemplo, quando o elemento é um link para outra página e não precisamos processar a informação do link, no servidor. Agora vamos estudar os diversos HTML Server Controls disponíveis.

## HTML Server Controls Disponíveis

Existem algumas propriedades que são comuns a todos os HTML Server Controls disponíveis. Isto acontece porque os mesmos herdam estas propriedades a partir de uma classe comum a todos.

### A Classe Base System.Web.UI.HtmlControls.HtmlControl

A classe base para todos os HTML Server Controls é System.Web.UI.HtmlControls.HtmlControl. Esta classe expõe uma série de propriedades e métodos comuns a todos os HTML Server Controls.

Na Tabela 7.4 temos a descrição das principais propriedades da classe System.Web.UI.HtmlControls.HtmlControl

**Tabela 7.4** Principais propriedades da classe System.Web.UI.HtmlControls.HtmlControl.

Propriedade	Descrição
Attributes	Retorna uma coleção de todos os atributos do tipo pares de nome/valor contidos na página .aspx, na qual está o controle.
ClientID	Retorna a identificação do controle. Esta identificação é gerada pelo ASP.NET e utilizada, dentre outras coisas, para manter o estado do controle.
Disabled	Utilizado para retornar ou definir um valor que indica se o controle está ou não desabilitado.
EnableViewState	Utilizado para retornar ou definir um valor booleano que indica se o controle de estado está habilitado (true) ou desabilitado (false), para o controle.
ID	Utilizado para retornar ou definir um nome associado ao controle. Este nome é utilizado no código para acessar as propriedades e métodos do controle.
Page	Retorna uma referência para a página que contém o controle.

Propriedade	Descrição
Site	Retorna informação a respeito do Web site.
Visible	Utilizado para retornar ou definir um valor que indica se o controle é ou não visível.

Vamos, finalmente, começar a estudar os HTML Server Controls disponíveis.

## HTMLForm Control

Este controle é utilizado para a criação de formulários. É o tradicional `<form> </form>`. Na prática, todo controle de servidor deve ser colocado dentro de um formulário. O que define o controle como um Server Control é o atributo `runat="server"`.

Sintaxe para o controle Form:

```

<form
 runat="server"
 id="identificação_no_código"
 method=POST | GET
 action="endereço"

>
 Controles que definem os elementos do formulário
</form>

```

Onde temos as seguintes propriedades:

- ◆ **runat="server"**: Define que é um controle de servidor, ou seja, que será processado no servidor.
- ◆ **id="identificação\_no\_código"**: Um nome que atribuímos ao controle, nome este que será usado para acessar as propriedades e métodos do controle, através de programação.
- ◆ **method**: Existem dois métodos para enviar um formulário para o servidor: POST e GET. O método GET é mais limitado, sendo que na grande maioria das vezes utilizamos o método POST.
- ◆ **action**: O endereço de uma página que irá receber os dados digitados no formulário e realizar alguma ação com os dados. Por exemplo, podemos criar um formulário de cadastro, onde o formulário digita os seus dados para o cadastramento. O formulário de cadastro é criado como uma página ASP.NET chamada `cadastro.aspx`. O usuário preenche os dados e clica em um botão Enviar. Ao clicar no botão Enviar, os dados digitados pelo usuário serão passados para a página definida na propriedade `action` – vamos chamá-la de `insere.aspx`. A página `insere.aspx` recebe os dados digitados pelo usuário, insere os mesmos em uma tabela de um banco de dados do SQL Server 2000 e retorna uma mensagem para o usuário. As etapas do exemplo descrito estão ilustradas na Figura 7.3.

**NOTA:** Para uma referência a todas as propriedades e métodos da classe `System.Web.UI.HtmlControls.HtmlControl`, consulte a documentação do Framework .NET, no item ".NET Framework Reference".

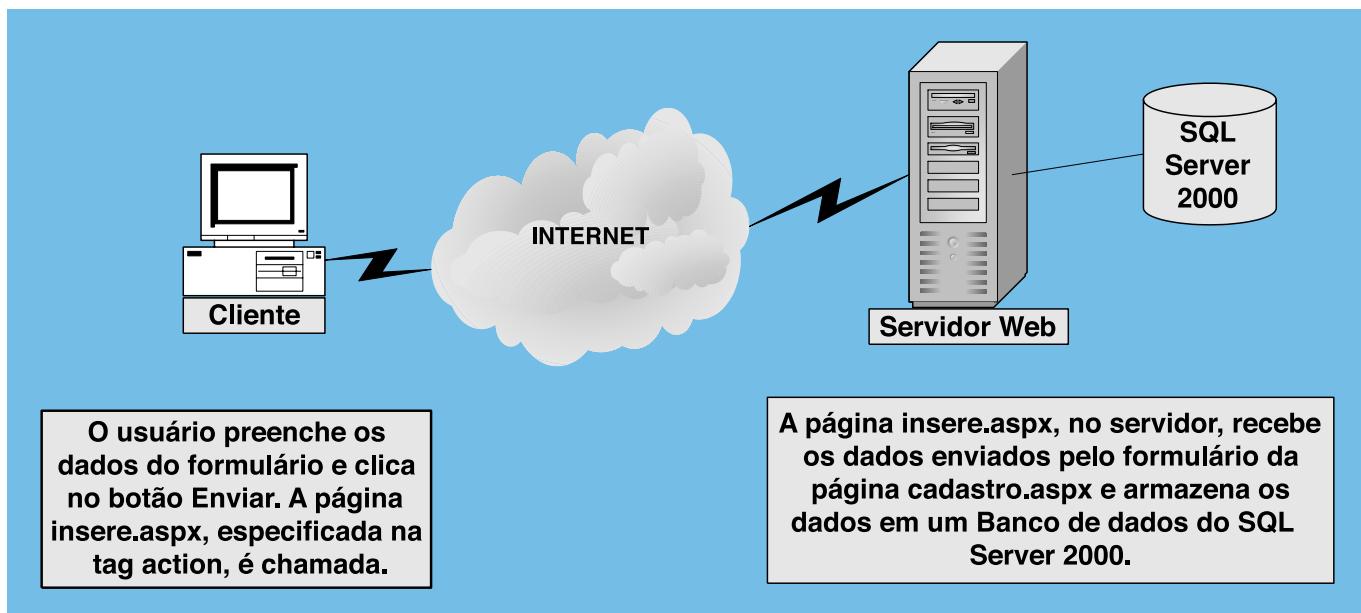


Figura 7.3: Dados de um formulário sendo processados.

Vamos apresentar um exemplo.

Exemplo: Neste exemplo criaremos uma página com um controle HtmlForm. No formulário colocaremos dois controles HtmlInputText (detalharemos este controle mais adiante) e um controle HtmlButton (também detalharemos este controle mais adiante). Ao carregar a página, o usuário digita um valor no primeiro campo e clica no botão Enviar. O valor digitado pelo usuário, no primeiro campo, é repetido no segundo campo.

Na Listagem 7.3 temos a página do exemplo proposto.

#### Listagem 7.3 – Um formulário simples – chap7ex3.aspx.

```
<html>
<script language="C#" runat="server">
 public void Enviar_Click(Object sender, EventArgs e)
 {
 Voltou.Value = Digitou.Value;

 }
</script>
<body>

<form method=post runat="server">
<table>
<tr>
```

**NOTA:** Não podemos incluir mais do que um controle HtmlForm em uma página. Se você inserir mais do que um HtmlForm, uma exceção será gerada. Por padrão, method é definido como POST e a propriedade action é definida com o endereço da própria página.

```

<td><h3>Digite um texto:</h3> </td>
<td><input id="Digitou" type=text size=40 runat="server"></td>
</tr>

<tr>
 <td><h3>Você digitou: </h3> </td>
 <td><input id="Voltou" type=text size=40 runat="server"> </td>
</tr>

<tr>
 <td><h3>Clique no botão --></h3></td>
 <td><input type=submit value="Enviar" OnServerClick="Enviar_Click"
runat="server"></td>
</tr>
</table>
</form>
</body>
</html>

```

Digite o código da Listagem 7.3 e salve o mesmo em um arquivo chamado chap7ex3.aspx, na pasta chap7, dentro da pasta wwwroot, conforme descrito anteriormente. Para acessar esta página utilize o seguinte endereço: http://localhost/chap7/chap7ex3.aspx

Ao carregarmos esta página no Internet Explorer obtemos um formulário com dois campos para digitação de texto. Digite a mensagem: “Exemplo de formulário”, no primeiro controle e dê um clique no botão Enviar. A mensagem “Exemplo de formulário” será exibida no segundo campo, conforme indicado na Figura 7.4.

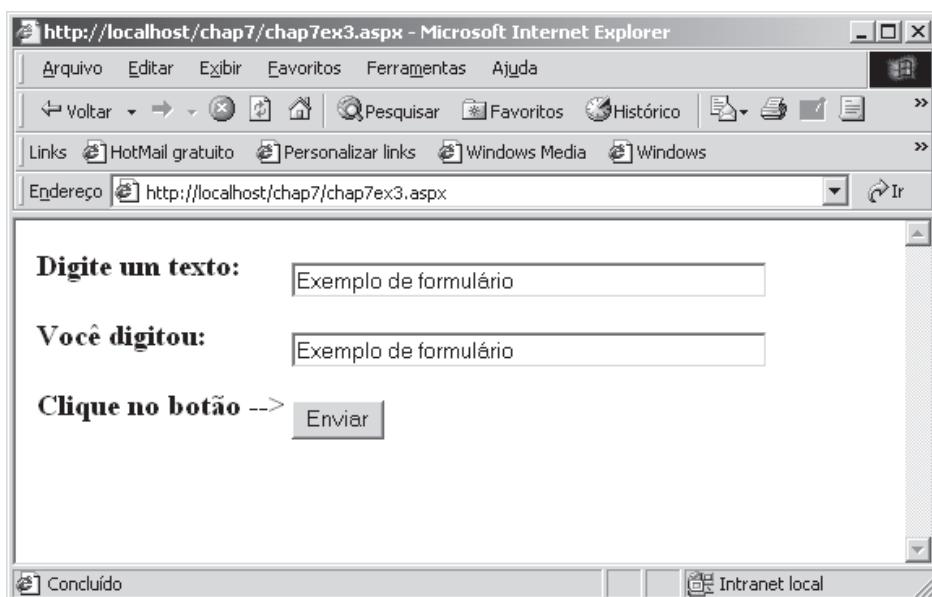


Figura 7.4: Um exemplo de formulário – chap7ex3.aspx.

Agora uma pequena demonstração de como o ASP.NET mantém o estado dos controles. No nosso exemplo vamos provar que o valor contido nos campos do formulário é mantido no servidor. Pressione F5 para atualizar a página do nosso exemplo (se você estiver utilizando o Netscape pressione Ctrl+R). Com o ASP 3.0 e versões anteriores, ao pressionarmos F5, a página seria recarregada e os campos apareceriam em branco ou com os seus valores padrão. No caso do ASP.NET são exibidos os valores atualmente definidos, ou seja, o estado atual da página foi mantido. Pressione F5 e observe. O texto “Exemplo de formulário” será exibido nos dois campos, o que comprova a manutenção automática de estado com o ASP.NET.

**NOTA:** Apenas para lembrar, você pode desabilitar a manutenção automática de estado utilizando a seguinte diretiva, no início da página:  
`%@Page EnableViewState="false" %>

Algumas observações sobre o código do nosso exemplo.

Não definimos a propriedade action do formulário. Com isso, ao clicar no botão Enviar, a página chama a si mesma, pois, conforme descrito, o valor padrão para a propriedade action é o endereço da própria página, onde está o formulário.

No formulário definimos um evento a ser disparado em resposta ao clique no botão enviar. Esta definição é feita na seguinte linha de código:

```
<td><input type="submit" value="Enviar" OnServerClick="Enviar_Click" runat="server"></td>
```

Neste caso definimos que, ao ocorrer o evento Click (OnServerClick), deve ser executado o procedimento Enviar\_Click. O procedimento enviar Click, atribui o valor digitado no campo Digitou à propriedade Value do campo Voltou. Isto é equivalente a copiar o conteúdo do campo Digitou para o campo Voltou. O procedimento Enviar\_Click está na seção de código da página, conforme indicado na figura a seguir:

```
<script language="C#" runat="server">
 public void Enviar_Click(Object sender, EventArgs e)
 {
 Voltou.Value = Digitou.Value;
 }
</script>
```

## HTMLInputText Control

É utilizado para a criação de campos, em um formulário, onde o usuário pode digitar texto. Este controle é utilizado para criar controles que rodam no servidor, para os tipos e funções indicados na Tabela 7.5. O que define o controle como sendo de servidor é a propriedade runat="server".

**Tabela 7.5** HTMLInputText Control

Tipo	Descrição
Texto	<input type=text>. Para a digitação de informações como por exemplo: nome, endereço, telefone, e-mail, etc.
Senha	<input type=password>. Para a digitação de senhas. Ao ser digitada uma senha, são exibidos somente asteriscos (*) no Windows 95, 98, NT ou 2000; já no Windows XP são exibidos pontos grandes.

Sintaxe para o controle input:

```
<input
 type="text" | password
 runat="server"
 id="identificação_no_código"
 maxlength="número máximo de caracteres"
 size="tamanho de exibição da caixa para digitação"
 value="valor padrão para o campo"
>
```

- ◆ **runat="server"**: Define que é um controle de servidor, ou seja, que será processado no servidor.
- ◆ **id="identificação\_no\_código"**: Um nome que atribuímos ao controle, nome este que será usado para acessar as propriedades e métodos do controle, através de programação.
- ◆ **maxlength**: tamanho máximo de caracteres para o campo.
- ◆ **size**: tamanho da caixa que é exibida para o campo. Se for menor que o tamanho máximo do campo, ao digitar informação, quando o texto digitado atingir o tamanho máximo de exibição, o texto será deslocado para a esquerda, para que o usuário continue digitando.
- ◆ **value**: Valor padrão associado ao campo. Ao carregarmos a página o controle é exibido em branco, a menos que exista um valor padrão para o mesmo, caso em que será exibido o valor padrão.

No exemplo da página chap7ex3.aspx, utilizamos dois controles do tipo <input type=text>, conforme indicado pelo trecho de código a seguir:

```
...
<td><h3>Digite um texto:</h3> </td>
<td><input id="Digitou" type="text" size=40 runat="server"></td>
...
<td><h3>Você digitou: </h3> </td>
<td><input id="Voltou" type="text" size=40 runat="server"> </td>
...
```

Vamos modificar algumas propriedades destes controles. Vamos definir um valor padrão para o controle Digitou. Vamos definir o valor padrão “Exemplo de ASP.NET”. Também vamos definir um tamanho máximo de 20 para a caixa de exibição do campo Digitou. Após as alterações propostas, o nosso código deve ficar assim:

```
...
<td><h3>Digite um texto:</h3> </td>
<td><input id="Digitou" type="text" size=20 runat="server"
 value="Exemplo de ASP.NET"></td>
...
<td><h3>Você digitou: </h3> </td>
<td><input id="Voltou" type="text" size=40 runat="server"> </td>
...

```

Faça estas alterações no código da Listagem 7.3 e salve a mesma com o nome de chap7ex4.aspx, na subpasta chap7, da pasta wwwroot descrita anteriormente. Para acessar esta nova versão da página, utilize o seguinte endereço: <http://localhost/chap7/chap7ex4.aspx>

Ao carregar esta página você obterá os resultados indicados na Figura 7.5.

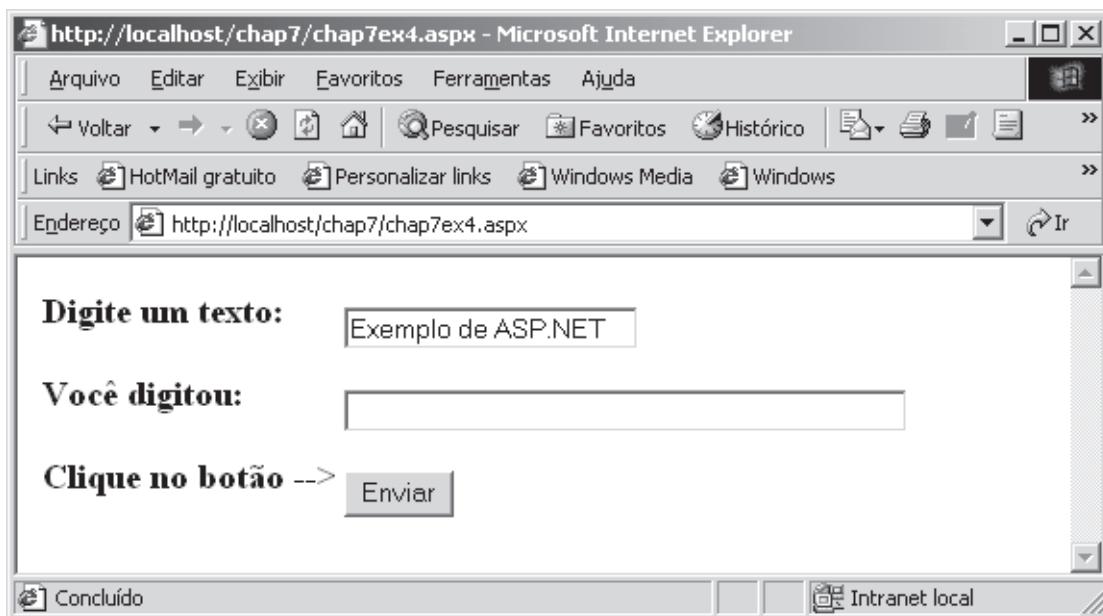


Figura 7.5: Um exemplo `HtmlTextControl` – `chap7ex4.aspx`.

## HTMLInputCheckBox

É utilizado para a criação de controles de seleção em um formulário. Os controles do tipo CheckBox são exibidos como pequenos quadradinhos. São controles do tipo marcado/desmarcado. Se o controle estiver marcado, ao clicarmos ele será desmarcado. Se o controle estiver desmarcado, ao clicarmos ele será marcado. Normalmente utilizado em grupos. Em um grupo de controles do tipo CheckBox podemos selecionar mais do que um controle. Por exemplo, para

criar um formulário de cadastro, onde o cliente pode informar as áreas de interesse em uma livraria. Podemos utilizar controles do tipo CheckBox, onde o cliente pode selecionar uma ou mais áreas de interesse. Apresentaremos este exemplo logo em seguida.

Sintaxe para o controle CheckBox:

```
<input
 type="checkbox"
 runat="server"
 id="identificação_no_código"
 checked
>
```

Onde temos:

- ◆ **runat="server"**: Define que é um controle de servidor, ou seja, que será processado no servidor.
- ◆ **id="identificação\_no\_código"**: Um nome que atribuímos ao controle, nome este que será usado para acessar as propriedades e métodos do controle, através de programação.
- ◆ **checked**: Valor booleano que indica se o controle está marcado (True) ou não (False).

Vamos apresentar um exemplo.

Exemplo: Vamos criar um formulário onde o cliente digita o Nome, o E-mail e seleciona as áreas de preferência, nas quais deseja receber informações por e-mail. Utilizaremos o evento Click do botão Enviar, para exibir os valores digitados pelo usuário. Para exibição dos valores vamos utilizar um controle do tipo caixa de texto HtmlTextArea control; mais adiante falaremos em detalhes sobre este controle.

Na Listagem 7.4 temos a página do exemplo proposto.

#### Listagem 7.4 – O controle CheckBox – chap7ex5.aspx.

```
<html>
<script language="C#" runat="server">
public void Enviar_Click(Object sender, EventArgs e)
{
 Dados.Visible = true;

 //Declaração das variáveis auxiliares

 String Aux;

 // Começo a montar uma string que será atribuída à
 // propriedade Value do controle Dados.
```

```
Aux = Nome.Value;
Aux = Aux + "\n" + Email.Value;

if (Negocios.Checked==true)
{
 Aux= Aux +"\n" +"Negócios";
}

if (Administração.Checked==true)
{
 Aux= Aux +"\n" +"Administração";
}

if (Informática.Checked==true)
{
 Aux= Aux +"\n" +"Informática";
}

if (Ficção.Checked==true)
{
 Aux= Aux +"\n" +"Ficção";
}

Dados.Value = Aux;

}

</script>

<body>
<form method=post runat="server">
 <table>
 <tr>
 <td><h3>Nome:</h3> </td>
 <td><input id="Nome" type=text size=40 runat="server"></td>
 </tr>
 <tr>
 <td><h3>E-mail:</h3> </td>
```

```
<td><input id="Email" type=text size=40 runat="server"> </td>
</tr>
<tr>
 <td colspan="2">Selecione as áres de interesse:</td>
</tr>
<tr>
 <td><h3>Negócios:</h3> </td>
 <td><input id="Negocios" type=checkbox runat="server"> </td>
</tr>

<tr>
 <td><h3>Administração:</h3> </td>
 <td><input id="Administração" type=checkbox runat="server"> </td>
</tr>

<tr>
 <td><h3>Informática:</h3> </td>
 <td><input id="Informática" type=checkbox runat="server"> </td>
</tr>

<tr>
 <td><h3>Ficção:</h3> </td>
 <td><input id="Ficção" type=checkbox runat="server"> </td>
</tr>

<tr>
 <td><h3>Dados do cliente:</h3> </td>
 <td><textarea id="Dados" cols="60" rows="10" visible="false"
runat="server"></textarea> </td>
</tr>

<tr>
 <td><h3>Clique no botão -></h3></td>
 <td><input type=submit value="Enviar" OnServerClick="Enviar_Click"
runat="server"></td>
</tr>

</table>

</form>
</body>
</html>
```

Digite o código da Listagem 7.4 e salve o mesmo em um arquivo chamado chap7ex5.aspx, na pasta chap7, dentro da pasta wwwroot, conforme descrito anteriormente. Para acessar esta página utilize o seguinte endereço: <http://localhost/chap7/chap7ex5.aspx>

Ao carregarmos esta página no Internet Explorer obtemos um formulário com dois campos para digitação de texto. Um para o nome do usuário e outro para o e-mail. Também são exibidos quatro controles do tipo checkbox onde o usuário pode escolher uma ou mais das seguintes opções:

- ◆ Negócios
- ◆ Administração
- ◆ Informática
- ◆ Ficção

Ao carregar a página pela primeira vez, o controle do tipo textarea está oculto, pois definimos sua propriedade visible como false. Para o nome digite José da Silva e para o e-mail digite [josedasilva@abc.com.br](mailto:josedasilva@abc.com.br). Selecione as opções Negócios e Informática, conforme indicado na Figura 7.6

The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost/chap7/chap7ex5.aspx - Microsoft Internet Explorer". The menu bar includes Arquivo, Editar, Exibir, Favoritos, Ferramentas, and Ajuda. The toolbar includes links for Voltar, Avançar, Home, Pesquisar, Favoritos, Histórico, and others. The address bar shows the URL "http://localhost/chap7/chap7ex5.aspx". The main content area contains a form with the following fields:

- Nome:**
- E-mail:**
- Selecionar áreas de interesse:**
- Negócios:**
- Administração:**
- Informática:**
- Ficção:**
- Dados do cliente:**
- Clique no botão -->**

The status bar at the bottom shows "Concluído" and "Intranet local".

Figura 7.6: Um exemplo com controles checkbox – chap7ex5.aspx.

Dê um clique no botão Enviar. Um controle do tipo textarea será exibido na parte final da página, com os dados que você digitou para o Nome e o e-mail do usuário, além das informações selecionadas, conforme indicado na Figura 7.7.

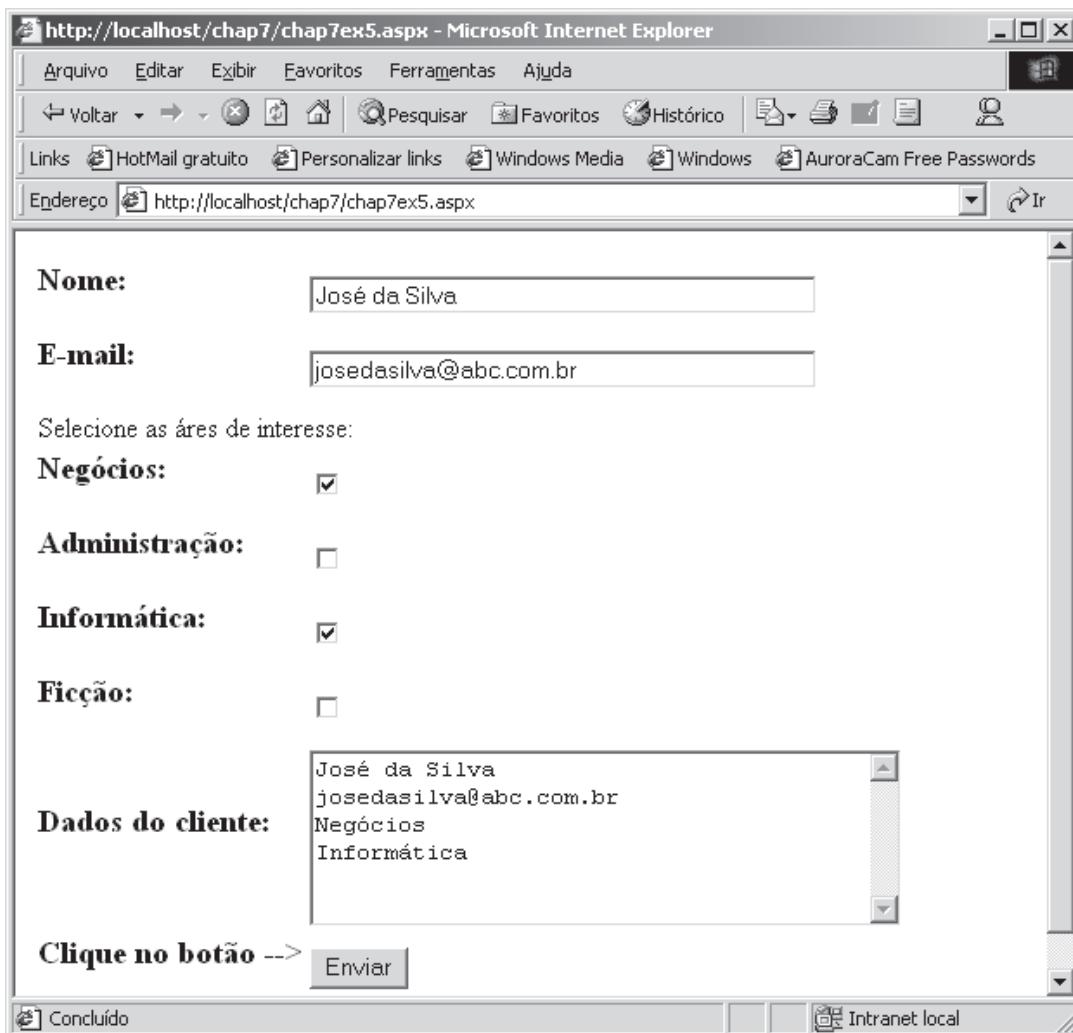


Figura 7.7: Exibição dos dados digitados e das opções selecionadas.

Desmarque a opção Negócios e marque as opções Administração e Ficção. Dê um clique no botão Enviar e observe que as novas opções já são exibidas no controle do tipo textarea.

Alguns comentários sobre o código do nosso exemplo.

- ♦ Na seção de apresentação (entre as tags <body> e </body>) criamos um formulário. Neste formulário colocamos dois controles do tipo text para que o usuário possa digitar o nome e o e-mail. Depois incluímos quatro controles do tipo checkbox para que o usuário possa selecionar uma ou mais preferências. Novamente utilizamos uma tabela para fazer o alinhamento dos controles.
- ♦ Na seção de código utilizamos o evento Click do botão Enviar. Dentro deste evento tornamos o controle Dados visível e depois declaramos uma variável do tipo String – Aux. Aí vamos concatenando os valores digitados para o nome e o e-mail à variável Aux, conforme indicado no trecho de código a seguir:

**NOTA:** Para maiores informações sobre os comandos básicos da linguagem HTML consulte o Anexo I.

```
Dados.Visible = true;

//Declaração das variáveis auxiliares

String Aux;

// Começo a montar uma string que será atribuída à
// propriedade Value do controle Dados.

Aux = Nome.Value;
Aux = Aux + "\n" + Email.Value;
```

Lembrando do Capítulo 3, o + é o operador de concatenação e o \n é o código para troca de linha.

- ◆ Na seqüência utilizamos a instrução if para determinar se o controle Negócios foi selecionado. Em caso afirmativo concatenamos “Negócios” à variável Aux, conforme indicado no trecho de código a seguir:

```
if (Negocios.Checked==true)
{
 Aux= Aux +"\n" +"Negócios";
}
```

O mesmo procedimento é utilizado para determinar se as demais opções foram ou não selecionadas.

- ◆ Na última linha da seção de código definimos a propriedade Value do controle Dados (controle do tipo textarea) como sendo igual à variável Aux, conforme indicado no seguinte comando:

```
Dados.Value = Aux;
```

## HtmTextArea Control

Utilizamos este controle no exemplo da Listagem 7.4. Vamos apresentar mais alguns detalhes sobre este controle. O controle textarea é utilizado para a digitação ou exibição de grandes quantidades de texto. A sua função é semelhante à função dos campos do tipo memo no Microsoft Access.

Sintaxe para o controle CheckBox:

```
<textarea
 runat="server"
 id="identificação_no_código"
 cols="número de colunas"
 name="nome enviado para o navegador"
 rows="número de linhas"
 onserverchange="onserverchangepandler"
```

```
>
Texto para ser exibido no controle.
</textare>
```

A propriedade onserverchange é utilizada para definir um evento que será executado em resposta ao envio da página para processamento no servidor.

Observe que este controle possui uma tag de abertura `<textarea>` e uma de fechamento `</textarea>`. Utilizamos esta sintaxe no exemplo do item anterior:

```
<td><textarea id="Dados" cols="60" rows="10" visible="false" runat="server"></textarea>
```

Ao invés desta sintaxe tradicional podemos utilizar a seguinte sintaxe:

```
<textarea id="TextArea1" cols=40 rows=4 runat=server />
```

Observe que simplesmente colocamos uma barra antes do sinal de fechamento `>`. Podemos utilizar qualquer uma das duas sintaxes.

## HTMLInputRadioButton Control

É utilizado para a criação de controles de seleção em um formulário. Os controles do tipo RadioButton são exibidos como pequenos círculos. São controles do tipo marcado/desmarcado. Se o controle estiver marcado, ao clicarmos ele será desmarcado. Se o controle estiver desmarcado, ao clicarmos ele será marcado. Normalmente utilizado em grupos. Em um grupo de controles do tipo RadioButton podemos selecionar somente um controle. Ao clicarmos em um controle, se outro controle do grupo estava marcado, será desmarcado. Para criar um grupo de controles do tipo RadioButton definimos todos os controles com o mesmo valor para a propriedade name.

Sintaxe para o controle CheckBox:

```
<input
 type=checkbox
 runat="server"
 id="identificação_no_código"
 checked
 name="nome do grupo"
>
```

Onde temos:

- ◆ `runat="server"`: Define que é um controle de servidor, ou seja, que será processado no servidor.
- ◆ `id="identificação_no_código"`: Um nome que atribuímos ao controle, nome este que será usado para acessar as propriedades e métodos do controle, através de programação.
- ◆ `checked`: Valor booleano que indica se o controle está marcado (True) ou não (False).
- ◆ `name`: nome do grupo. Para criar um grupo de controles do tipo RadioButton basta definir o mesmo nome para todos os controles do grupo.

Vamos apresentar um exemplo.

Exemplo: Vamos criar um formulário com três controles do tipo RadioButton. O usuário seleciona uma opção e clica no botão Enviar. Para processar as informações enviadas pela página vamos utilizar a propriedade OnServerChange do controle RadioButton. Esta propriedade permite que seja definido um procedimento que será executado quando a página for processada, em resposta a uma alteração no controle. Este procedimento detecta o tipo de cartão selecionado e exibe o tipo de cartão em um controle do tipo text.

Na Listagem 7.5 temos a página do exemplo proposto.

## Listagem 7.5 – Controles do tipo RadioButton – chap7ex6.aspx.

```
<html>

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

 void Server_Change(object Source, EventArgs e)
 {
 if (Visa.Checked == true)
 Exibe.Value="O SEU CARTÃO É VISA!";
 else if (Master.Checked == true)
 Exibe.Value="O SEU CARTÃO É MASTER CARD!";
 else if (Outros.Checked == true)
<1 Exibe.Value="OUTROS TIPOS DE CARTÃO!";

 }
</script>

<body>
<form runat="server">

 <h3>Selecione o tipo de cartão:</h3>

 <input type="radio"
 id="Visa"
 name="Cartao"
 OnServerChange="Server_Change"
 runat="server"/>

 Visa

 <input type="radio"
 id="Master"
 name="Cartao"

 </form>
</body>
```

```
OnServerChange="Server_Change"
runat="server"/>

Master Card

<input type="radio"
 id="Outros"
 name="Cartao"
 OnServerChange="Server_Change"
 runat="server"/>

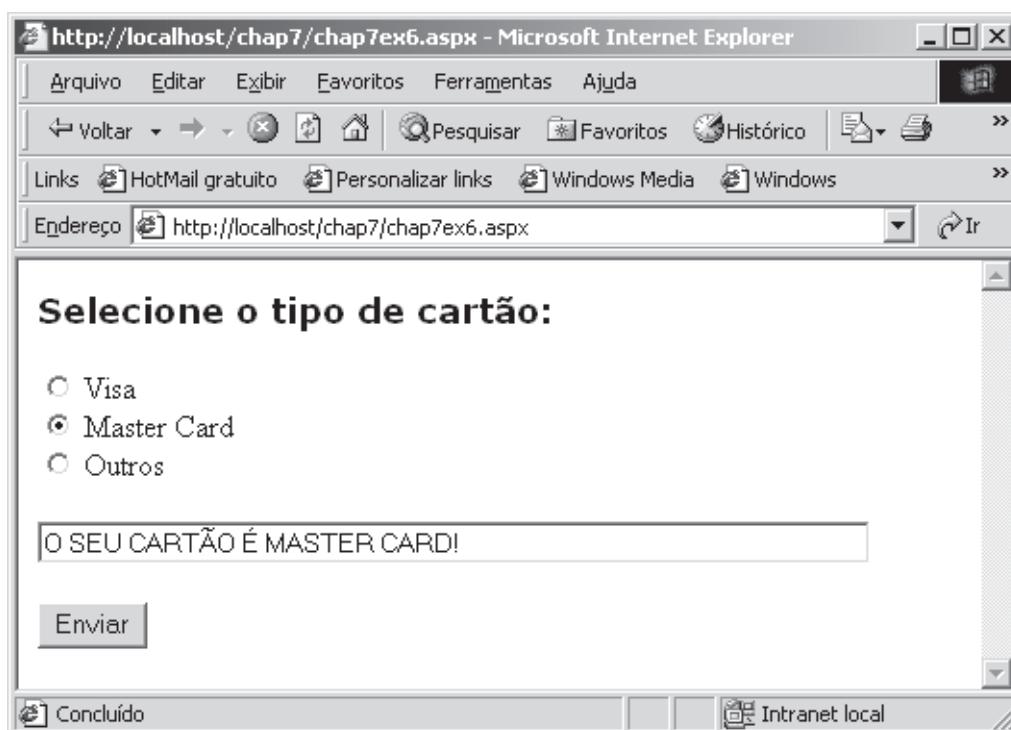
Outros

<p>
<input id="Exibe" type=text size=60 runat="server">
<p>
<input type=submit id="Enviar"
 value="Enviar"
 runat="server">
</form>
</body>
</html>
```

Digite o código da Listagem 7.5 e salve o mesmo em um arquivo chamado chap7ex6aspx, na pasta chap7, dentro da pasta wwwroot, conforme descrito anteriormente. Para acessar esta página utilize o seguinte endereço: <http://localhost/chap7/chap7ex6aspx>

Ao carregarmos esta página no Internet Explorer, obtemos um formulário com três controles do tipo RadioButton. Podemos selecionar um dos controles. Dê um clique na opção Visa. Agora dê um clique na opção Master Card. Observe que a opção Visa foi, automaticamente, desmarcada. Dê um clique no botão Enviar. Você obterá o resultado indicado na Figura 7.8.

A novidade neste exemplo é que, ao invés de utilizar o evento Click do botão de comando, utilizamos um evento definido pela propriedade OnServerChange dos controles do tipo RadioButton.



**Figura 7.8: Um exemplo com controles RadioButton – chap7ex6.aspx.**

## Controles Para a Criação de Tabelas: HtmlTable, HtmlTableRow e HtmlTableCell Controls

Os controles HtmlTable, HtmlTableRow e HtmlTableCell são os controles de servidor, equivalentes às tags HTML para a criação de tabelas: <table> </table>, <tr> </tr> e <td> </td>, respectivamente.

A vantagem de utilizarmos os controles de servidor é que temos acesso a uma série de propriedades e métodos dos controles de servidor. Por exemplo, podemos adicionar ou remover linhas e colunas dinamicamente, com base em eventos ocorridos na página. Através de programação podemos utilizar os métodos das coleções HtmlTableRowCollection e HtmlTableCellCollection para adicionar ou remover elementos da tabela.

Vamos estudar cada um dos controles individualmente, e depois vamos apresentar alguns exemplos de utilização.

### O Controle HtmlTable

Temos a seguinte sintaxe para este controle:

```
<table
 runat="server"
 id="identificação_no_código"
 align=left | center | right
 bgcolor="cor de fundo"
 border="tamanho da borda em pixels"
```

```

bordercolor="cor da borda"
cellpadding="espaço entre a borda e o conteúdo da célula, em pixels."
cellspacing=" espaço entre as células, em pixels "
height="altura da tabela"
rows="um objeto do tipo coleção de linhas - HtmlTableRowCollection "
width="largura da tabela"
>
<tr><td></td></tr>
<tr><td></td></tr>
</table>

```

Observe que, além de definir uma coleção de linhas, no parâmetro rows, também podemos acrescentar linhas e células utilizando as tags tradicionais: <tr> </tr> para adicionar linhas e <td> </td> para adicionar células dentro de uma linha (colunas da tabela).

Quando utilizamos um server control HtmlTable para criar uma tabela, podemos imaginar a tabela como sendo formada por uma coleção de linhas. Podemos adicionar ou remover itens desta coleção, através do código de programação. Também podemos imaginar cada linha, como uma coleção de células. Podemos adicionar ou remover itens desta coleção, utilizando código de programação. No final deste item veremos um exemplo prático de adição de linhas e células utilizando código de programação.

## O Controle HtmlTableRow

Temos a seguinte sintaxe para este controle:

```

<tr
 runat="server"
 id="identificação_no_código"
 align="alinhamento para o conteúdo da linha"
 bgcolor="cor de segundo plano"
 bordercolor="cor da borda"
 height="altura da linha"
 cells="um objeto do tipo HtmlTableCellCollection "
 valign="alinhamento vertical do conteúdo da linha"
>
<td>Conteúdo da célula</td>
<td> Conteúdo da célula </td>
<td> Conteúdo da célula </td>
</tr>

```

## O Controle HtmlTableCell

Temos a seguinte sintaxe para este controle:

```
<td ou th
 runat="server"
 id="identificação_no_código"
 align="alinhamento para o conteúdo da linha"
 bgcolor="cor de segundo plano"
 bordercolor="cor da borda"
 height="altura da célula"
 nowrap="true | false"
 rowspan ou colspan="número de linhas ou células para mesclar"
 valign="alinhamento vertical do conteúdo da linha"
 width="largura da célula"
>
 Conteúdo da célula.
</td or /th>
```

Duas propriedades merecem maiores comentários.

- ◆ **nowrap**: Pode ter o seu valor definido como true; neste caso ocorre o retorno automático do texto ao alcançar o final da célula, caso o conteúdo da célula não caiba no tamanho definido para a célula. Se o valor for definido em false, não ocorrerá o retorno automático.
- ◆ **colspan ou rowspan**: Esta propriedade é utilizada para mesclar células. Por exemplo, se temos uma tabela com duas colunas, porém em uma das linhas queremos mesclar as duas colunas, utilizamos o seguinte código:

```
<tr>
 <td width="100%" colspan="2">Conteúdo da Célula</td>
</tr>
```

## Um Exemplo Prático

Vamos apresentar um exemplo, no qual demonstramos a capacidade de adicionar linhas e células, dinamicamente, a uma tabela, utilizando código de programação. Ao carregar a página, é exibida uma tabela de uma única célula. Vamos criar um formulário onde são apresentados dois campos para o usuário. No primeiro, o usuário digita o número de linhas para a tabela. No segundo campo, o usuário digita o número de colunas. Ao clicar no botão Enviar, utilizaremos o evento Load da página, para criar uma tabela com o número de linhas e colunas especificadas pelo usuário.

Na Listagem 7.6 temos a página do exemplo proposto.

**Listagem 7.6 – Adicionando linhas e células, dinamicamente – chap7ex7.aspx.**

```
<html>
<head>
<script language="C#" runat="server">

void Page_Load(Object sender, EventArgs e) {

int row = 0;

// Vamos gerar, dinamicamente, uma tabela
// com o número de linhas e células, especificado
// pelo usuário.

int numrows = Convert.ToInt32(Linhas.Value);
int numcells = Convert.ToInt32(Colunas.Value);

// O laço de fora define o número de linhas.
// para cada linha, isto é, para cada passagem do laço
// de fora, o laço mais interno é executado tantas vezes
// quantas forem o número de colunas especificadas.

for (int j=0; j<numrows; j++)
{
 //Declaramos uma variável do tipo HtmlTableRow
 HtmlTableRow r = new HtmlTableRow();

 // Define a cor de segundo plano, alternadamente
 // para linhas pares e ímpares.
 if (row%2 == 1)
 r.BgColor="Gainsboro";

 row++;

 for (int i=0; i<numcells; i++)
 {

 // Declaro e crio um novo objeto do tipo HtmlTableCell()
 // e adiciono à linha criada acima
 HtmlTableCell c = new HtmlTableCell();
 r.Cells.Add(c);
 }
}

// Agora que a tabela está pronta, adiciono-a ao formulário
Form1.Controls.Add(r);
}
</script>
</head>
<body>
<form id="Form1" runat="server">
<table border="1">
<tr>
<td>Linha 1, Coluna 1</td>
<td>Linha 1, Coluna 2</td>
</tr>
<tr>
<td>Linha 2, Coluna 1</td>
<td>Linha 2, Coluna 2</td>
</tr>
<tr>
<td>Linha 3, Coluna 1</td>
<td>Linha 3, Coluna 2</td>
</tr>
</table>
</form>
</body>

```

```
 HtmlTableCell c = new HtmlTableCell();

 // Defino o conteúdo da célula que foi recém criada.
 c.Controls.Add(new LiteralControl("Linha " + j.ToString() + ", Coluna " +
i.ToString()));

 // Adiciono a célula à coleção de células da linha.
 r.Cells.Add(c);

 }

 // Adiciono a linha à coleção de linhas da tabela.
 Table1.Rows.Add(r);
}

}

</script>

</head>
<body>

<h3>HtmlTable Example</h3>

<form runat="server">

<p>
<table id="Table1"
 CellPadding=5
 CellSpacing=0
 Border="1"
 BorderColor="black"
 runat="server" />
<p>
 Número de linhas :
 <input id="Linhas" type="text" size=10 value="1" runat="server">

 Número de colunas :
 <input id="Colunas" type="text" size=10 value="1" runat="server">


```

```

<input type="submit" value="Gerar a tabela dinamicamente." runat="server">

</form>

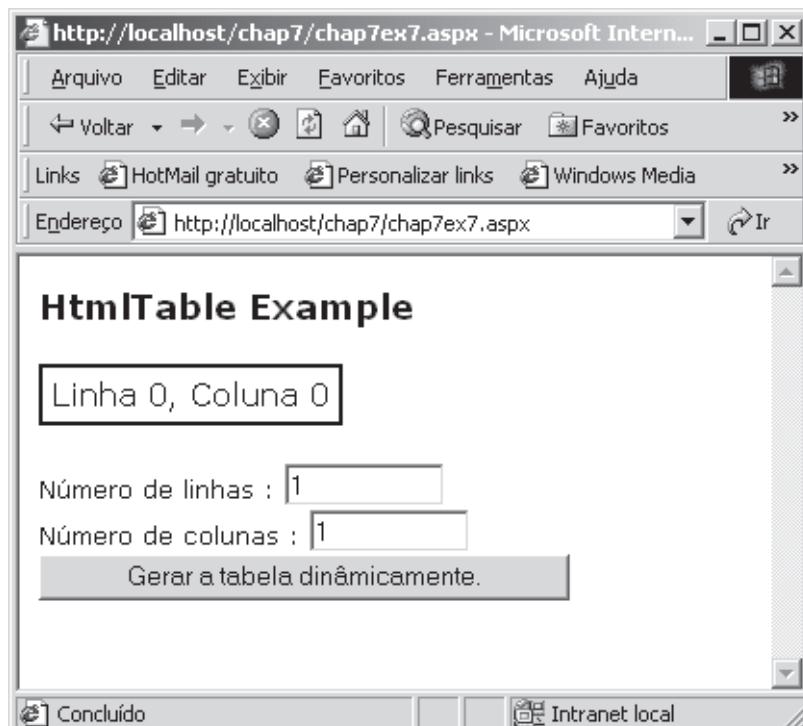
</body>
</html>

```

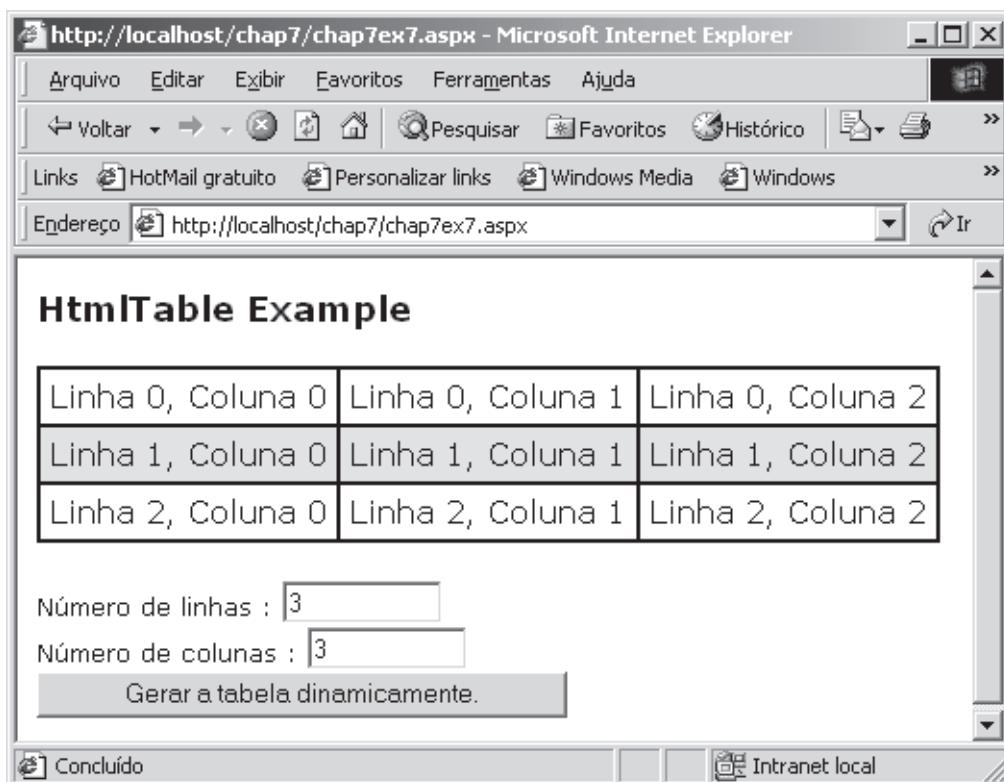
Digite o código da Listagem 7.6 e salve o mesmo em um arquivo chamado chap7ex7.aspx, na pasta chap7, dentro da pasta wwwroot, conforme descrito anteriormente. Para acessar esta página utilize o seguinte endereço: <http://localhost/chap7/chap7ex7.aspx>

Ao carregarmos esta página no Internet Explorer, obtemos uma tabela com uma linha e uma coluna e um formulário com dois campos para digitação. O valor padrão destes campos é definido como 1, conforme indicado na Figura 7.9.

Digite 3 para o número de linhas e 3 para o número de colunas. Dê um clique no botão “Gerar a tabela dinamicamente”. Uma tabela com três linhas e três colunas por linha será automaticamente gerada, conforme indicado na Figura 7.10.



**Figura 7.9: Formulário para geração automática de tabela – chap7ex7.aspx.**



**Figura 7.10: Um exemplo de tabela gerada dinamicamente.**

Alguns comentários sobre o código da página chap7ex7.aspx.

- ◆ Na seção de apresentação da página, criamos um formulário com dois campos para digitação do número de linhas e colunas da tabela a ser gerada dinamicamente. Também colocamos um botão do tipo Submit.
- ◆ Na seção de código é interessante observar que não utilizamos o evento Click do botão Enviar do formulário. Poderíamos ter utilizado este evento, como já fizemos em exemplos anteriores. No exemplo da página chap7ex7.aspx optamos por utilizar o evento Load do objeto Page. Lembrando do que foi apresentado no início do capítulo, toda página APS.NET, ao ser carregada, é compilada e um objeto do tipo Page é gerado. Este objeto é baseado na classe Page do namespace System.Web.UI. Sendo um objeto, temos acesso a seus métodos, propriedades e demais membros. A seguir temos o código para a declaração do evento Load da página:

```
void Page_Load(Object sender, EventArgs e)
```

Iniciamos o procedimento declarando três variáveis do tipo inteiro:

```
int row = 0;
int numrows = Convert.ToInt32(Linhas.Value);
int numcells = Convert.ToInt32(Colunas.Value);
```

A variável row é utilizada para definir se estamos em uma linha ímpar (1<sup>a</sup>, 3<sup>a</sup>, etc.) ou em uma linha par (2<sup>a</sup>, 4<sup>a</sup>, etc). Para as linhas pares definiremos uma cor de fundo diferente, para dar um efeito especial na apresentação da tabela.

**IMPORTANTE:** Uma tabela dinâmica é formada por uma coleção de linhas, onde cada linha é formada por uma coleção de células. Com esta idéia em mente é que, através do código do evento Load, vamos criar uma tabela com o número de linhas e colunas digitado pelo usuário.

A variável numrows contém o número de linhas digitado pelo usuário. Observe que temos que converter o valor digitado pelo usuário para um tipo Int32. Esta operação é necessária porque os valores digitados em um campo de um formulário são considerados valores de texto. A variável numcells contém o número de colunas digitado pelo usuário. Também convertemos este número para Int32.

Em seguida iniciamos dois laços for. O laço for externo varia de 0 até o número de linhas especificado pelo usuário. O laço for interno varia, para cada valor do laço externo, de 0 até o número de colunas especificado pelo usuário, conforme indicado pelo trecho de código a seguir:

```
for (int j=0; j<numrows; j++)
{
 //Declaramos uma variável do tipo HtmlTableRow
 HtmlTableRow r = new HtmlTableRow();

 // Define a cor de segundo plano, alternadamente
 // para linhas pares e ímpares.

 if (row%2 == 1)
 r.BgColor="Gainsboro";

 row++;

 for (int i=0; i<numcells; i++)
 {

 // Declaro e crio um novo objeto do tipo HtmlTableCell()

 HtmlTableCell c = new HtmlTableCell();

 // Defino o conteúdo da célula que foi recém criada.

 c.Controls.Add(new LiteralControl("Linha " + j.ToString() + ", "
 Coluna " + i.ToString()));

 // Adiciono a célula à coleção de células da linha.

 r.Cells.Add(c);
 }

 // Adiciono a linha à coleção de linhas da tabela.

 Table1.Rows.Add(r);
}
```

No início do laço, declaramos e inicializamos uma variável r, do tipo HtmlTableRow. Estamos criando um objeto do tipo HtmlTableRow. Isto é feito com o seguinte código:

```
HtmlTableRow r = new HtmlTableRow();
```

Em seguida um teste para definir se estamos em uma linha par ou ímpar. O teste divide o número da linha por 2 e retorna o resto da divisão. Se este resto for igual a 1 estamos em uma linha ímpar e a cor de segundo plano é alterada. Observe que, para alterar a cor de segundo plano, utilizamos a propriedade BgColor do objeto r, que é um objeto do tipo HtmlTableRow(). Isto é feito pelo seguinte trecho de código:

```
if (row%2 == 1)
 r.BgColor="Gainsboro";

row++;
```

Gainsboro é o nome de uma cor. Ao invés do nome poderíamos utilizar os tradicionais códigos de seis dígitos hexadecimais. Por exemplo: #000000 para preto, #FFFFFF para branco, #FF0000 para vermelho e assim por diante. Após definirmos o nome da cor, incrementamos a variável inteira row. Observe que com ASP.NET e a biblioteca de classes do Framework .NET, a programação para Web ficou muito mais parecida com a programação tradicional para Windows, orientada a eventos.

Na seqüência do código entramos no laço for interno. Neste laço é declarada e inicializada uma variável c, do tipo HtmlTableCell(). Para cada passagem do laço interno vamos adicionar uma célula, a coleção de células do objeto HtmlTableRow atual. Na prática estamos adicionando as colunas para a linha atual. Também é feita a definição do conteúdo para a célula.

A declaração da variável do tipo HtmlTableCell() é feita com o seguinte comando:

```
HtmlTableCell c = new HtmlTableCell();
```

A definição do conteúdo da célula é feita com o seguinte comando:

```
c.Controls.Add(new LiteralControl("Linha " + j.ToString() + ", Coluna " + i.ToString()));
```

A adição da célula c à coleção de células da linha r é feita com o seguinte comando:

```
r.Cells.Add(c);
```

Para cada passada do laço interno uma nova célula é adicionada à linha. Ao final, a linha terá tantas células quantas forem as colunas especificadas pelo usuário.

Uma vez que adicionamos várias células à coleção de células da linha, o passo final é adicionar a linha à coleção de linhas da tabela. Isto é feito no seguinte comando:

```
Table1.Rows.Add(r);
```

Para cada passada do laço externo uma nova linha é adicionada à coleção de linhas da tabela. Ao final teremos uma tabela com tantas linhas quantas especificadas pelo usuário e, em cada linha, tantas células quantas o número de colunas especificado pelo usuário. Em resumo: exatamente o resultado proposto pelo exemplo.

## HTMLSelect Control

É utilizado para a criação de controles do tipo caixa de seleção, onde são apresentados diversos itens em uma lista, onde o usuário pode selecionar um determinado item. A funcionalidade é idêntica à tag <SELECT> </SELECT> do HTML.

Sintaxe para o controle Select:

```
<select
 runat="server"
 id="identificação_no_código"
 OnServerChange="onserverchangehandler"
 DataSource="fonte de dados"
 DataTextField="descrição do campo ao qual é vinculada a caixa de seleção"
 DataValueField="valor do campo ao qual é vinculada a caixa de seleção"
 Multiple
 Items="coleção de elementos do tipo options"
 SelectedIndex="índice do elemento atualmente selecionado"
 Size="número de itens visíveis ao mesmo tempo. Por padrão é 1"
 Value="Valor do item atualmente selecionado"
 >
<option>Texto para a opção 1</option>
<option>Texto para a opção 2</option>
...
<option>Texto para a opção n</option>
</select>
```

Eu não me canso de repetir: “Sendo este um controle de servidor, o mesmo é criado como um objeto no momento da compilação da página ASP.NET. Sendo um objeto, temos acesso a suas propriedades e métodos.

Temos duas maneiras de definir os elementos da lista:

- ◆ Utilizando a maneira tradicional, onde cada elemento é definido através de um conjunto de tags <option> Texto </option>, com uma tag para cada elemento da lista.
- ◆ Utilizando a coleção Items, à qual podemos adicionar dinamicamente, através de programação, elementos do tipo options. Esta coleção é uma instância da classe ListItemCollection, a qual pertence ao namespace System.Web.UI.WebControls. Esta classe possui métodos para adicionar e remover elementos, utilizando programação.

Vamos apresentar um exemplo.

Exemplo: Vamos criar um formulário onde temos três controles:

- ◆ Uma lista com alguns nomes de cores. Ao lado desta lista temos um botão Aplicar. Ao clicar neste botão, a cor selecionada na lista será aplicada como cor de segundo plano de um controle do tipo texto existente no formulário.
- ◆ Um campo onde o usuário pode adicionar uma cor à lista de cores. Ao lado deste campo temos um botão. Ao clicar neste botão, a cor digitada no campo será adicionada à lista de cores. Para isso utilizaremos o evento Click deste botão.
- ◆ Um campo onde exibimos um determinado texto e onde é aplicada a cor de segundo plano selecionada na lista de cores.

Na Listagem 7.7 temos a página do exemplo proposto.

### Listagem 7.7 – Adicionando elementos dinamicamente a uma lista – chap7ex8.aspx.

```
<html>

<head>

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

void Aplicar_Click(object Source, EventArgs e)
{
 Span1.Style["background-color"] = ColorSelect.Value;
}

void AdicionaALista_Click(object Source, EventArgs e)
{
 ColorSelect.Items.Add(Text1.Value);
}

</script>
</head>

<body>

<h3>Exemplo de lista dinâmica!</h3>

<form runat="server">
Selezione uma cor:

```

```

<select id="ColorSelect" runat="server">
 <option>SkyBlue</option>
 <option>LightGreen</option>
 <option>Gainsboro</option>
 <option>LemonChiffon</option>
</select>

<input type="button" runat="server" Value="Aplicar" OnServerClick="Aplicar_Click">

<p> A cor desejada não está na lista? Digite a cor desejada:

<input type="text" id="Text1" runat="server">
<input type="button" runat="server" Value="AdicionaALista"
OnServerClick="AdicionaALista_Click">

<p>

 Clique no botão Aplicar para definir a cor de segundo plano!

</form>

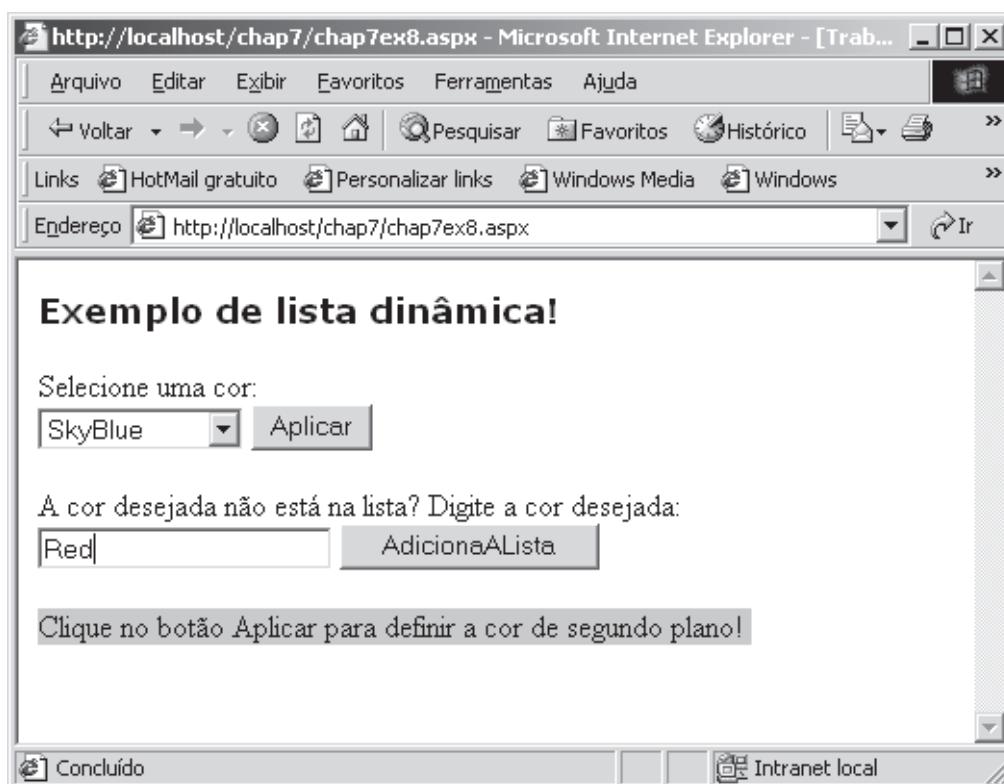
</body>
</html>

```

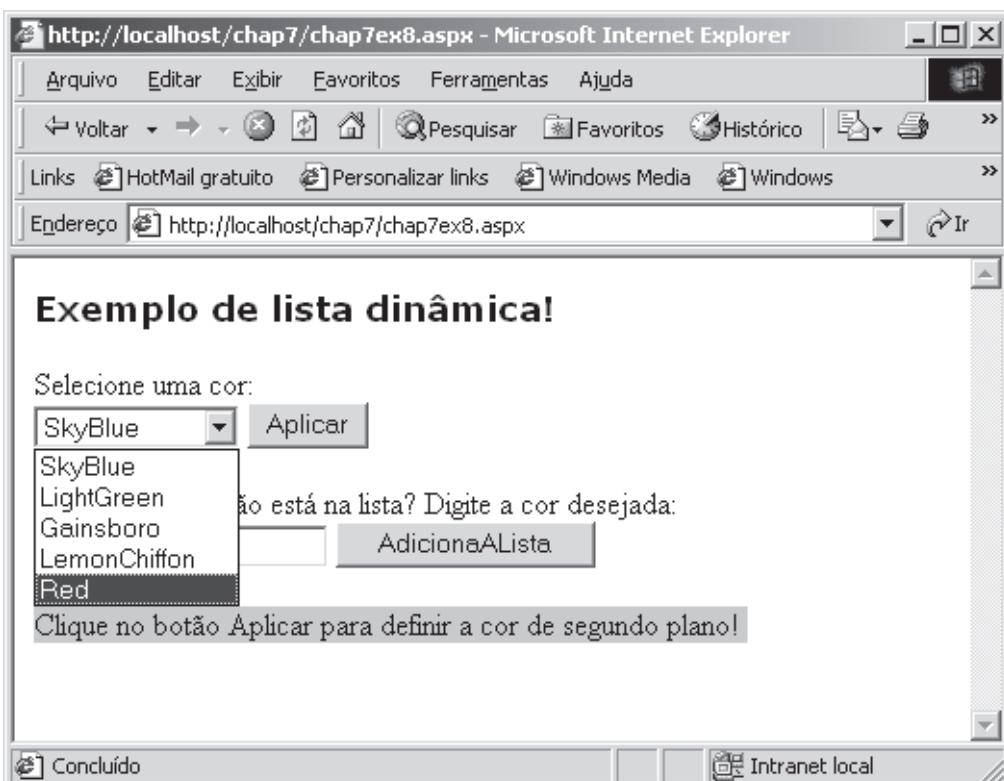
Digite o código da Listagem 7.7 e salve o mesmo em um arquivo chamado chap7ex8.aspx, na pasta chap7, dentro da pasta wwwroot, conforme descrito anteriormente. Para acessar esta página utilize o seguinte endereço: <http://localhost/chap7/chap7ex8.aspx>

Na lista de cores selecione SkyBlue e dê um clique no botão Aplicar. Observe que a cor de segundo plano do texto, no final do formulário, é alterada, conforme indicado na Figura 7.11.

Agora vamos adicionar uma cor à lista. No campo para digitação de texto digite Red e dê um clique no botão AdicionaALista. Abra a lista e observe que a cor Red (vermelho) já faz parte das opções da Lista, conforme indicado na Figura 7.12.



**Figura 7.11: Alterando a cor de fundo usando uma lista – chap7ex8.aspx.**



**Figura 7.12: Elemento adicionado dinamicamente à lista.**

Alguns comentários sobre o código da página chap7ex8.aspx.

- ◆ Quando o usuário seleciona uma cor na lista de cores e clica no botão Aplicar, é disparado o evento Click do botão. Em resposta a este evento é executado o procedimento Aplicar\_Click. Este procedimento utiliza a propriedade Style do controle Span1 para definir a cor de segundo plano como sendo a cor selecionada na lista de cores. O procedimento Aplicar\_Click está indicado no trecho de código a seguir:

```
void Aplicar_Click(object Source, EventArgs e)
{
 Span1.Style["background-color"] = ColorSelect.Value;
}
```

- ◆ Para adicionar um elemento à lista de cores, utilizamos o evento Click do botão AdicionaALista. Neste procedimento utilizamos o método Add da coleção Items da lista, conforme indicado no trecho de código a seguir:

```
void AdicionaALista_Click(object Source, EventArgs e)
{
 ColorSelect.Items.Add(Text1.Value);
}
```

Passamos para o método Add o valor digitado no campo Text1 do formulário: Text1.Value.

Vou repetir mais uma vez: Observe o quanto o modelo de programação para o ASP.NET se parece com a programação tradicional de aplicações Windows orientada a eventos.

## HTMLAnchor Control

É utilizado para a criação de links em páginas ASP.NET. A funcionalidade é idêntica à tag <a> </a> do HTML. A diferença é que, por ser um controle de servidor, o mesmo é processado pelo Framework .NET no servidor IIS, o que nos disponibiliza uma série de métodos e propriedades para o controle HtmlAnchor.

Sintaxe para o controle HtmlAnchor:

```
<a
 runat="server"
 id="identificação_no_código"
 href="enereço absoluto ou relativo"
 name="nome quando adicionado à lista de favoritos do navegador"
 OnServerClick="nome do procedimento a ser executado em resposta a um click no
link"
 target="abre na mesma janela, em uma nova janela, em um frame da janela atual,
etc"
 title="título a ser exibido na janela de título do navegador"
>
 Texto do link.

```

**NOTA:** A propriedade Style é utilizada para definir aspectos visuais dos elementos, como cor da fonte, tamanho, negrito, itálico, etc. Existe uma definição para os vários atributos visuais possíveis de serem alterados. Estes atributos fazem parte das chamadas CSS – Cascading Style Sheets. Para maiores informações sobre CSS e uma referência completa sobre os diversos atributos disponíveis, consulte um dos seguintes endereços: [www.w3.org](http://www.w3.org) ou [www.w3schools.com](http://www.w3schools.com).

Vamos apresentar um exemplo simples.

Exemplo: Vamos criar um formulário com dois links. Um link aponta para a página chap7ex8.aspx e outro para a página chap7ex7.aspx.

Na Listagem 7.8 temos a página do exemplo proposto.

**Listagem 7.8 – O controle HtmlAnchor – chap7ex9.aspx.**

```
<%@ Import Namespace="System" %>
<html>
<script language="C#" runat="server">

</script>
<body>
<1
<H1>Selecione um dos links abaixo !!! </H1>

<TABLE>

<TR>
 <TD>

 Exemplo 7 do Capítulo 7.

 </TD>
</TR>

<TR>
 <TD>

 Exemplo 8 do Capítulo 7.

 </TD>
</TR>

</TABLE>

</body>
</html>
```

Digite o código da Listagem 7.8 e salve o mesmo em um arquivo chamado chap7ex9.aspx, na pasta chap7, dentro da pasta wwwroot, conforme descrito anteriormente. Para acessar esta página utilize o seguinte endereço: <http://localhost/chap7/chap7ex9.aspx>

Você obterá o resultado indicado na Figura 7.13.

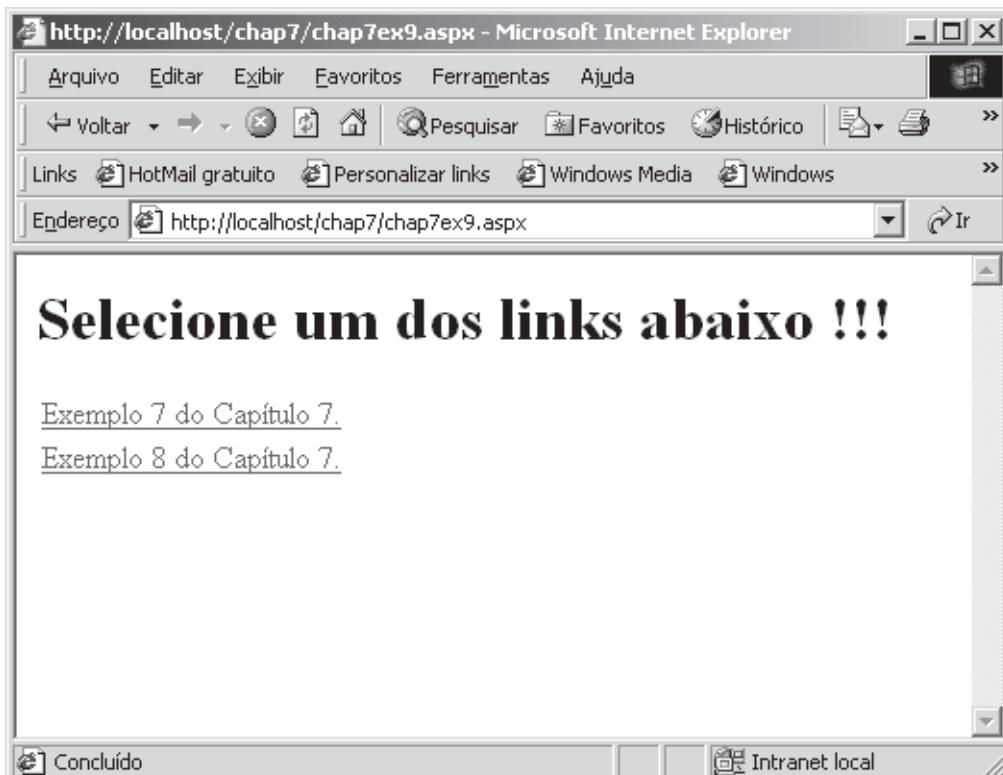


Figura 7.13: O controle HtmlAnchor – chap7ex9.aspx.

## HtmlInputButton Control

É utilizado para a criação de um botão de comando em um formulário. O botão de comando normalmente é utilizado para enviar os dados do formulário para processamento. Já tivemos diversos exemplos de utilização deste controle. Para vê-lo em funcionamento, inclusive com código para responder ao evento Click do botão, consulte os seguintes exemplos deste capítulo:

- ◆ chap7ex3.aspx
- ◆ chap7ex4.aspx
- ◆ chap7ex5.aspx
- ◆ chap7ex8.aspx

Sintaxe para o controle HtmlInputButton:

```
<input
 type=button | submit | reset
```

```
runat="server"
id="identificação_no_código"
OnServerClick="nome do procedimento a ser executado em resposta ao evento Click"
>
```

## HtmButton Control

É utilizado para a criação de botões de comandos mais sofisticados dos que os criados pelo controle HtmlInputButton. Com este controle temos acesso a uma série de métodos e eventos, os quais permitem a criação de alguns efeitos interessantes. Além do texto podemos associar uma determinada figura com o botão de comando.

Sintaxe para o controle HtmButton:

```
<button
 runat="server"
 id="identificação_co_código"
 OnServerClick="procedimento a ser executado ao clicarmos no botão"
>
 texto ou image
</button>
```

Este elemento é bastante rico em funcionalidade. Podemos aplicar uma série de efeitos interessantes ao mesmo. Vamos criar um formulário, baseado em um exemplo da documentação do Framework .NET, onde temos dois controles HtmButton. Vamos utilizar o evento Click de cada controle para informar quando um dos controles foi clicado. Também vamos utilizar a propriedade Style para definir algumas características visuais de cada controle. Em um dos controles além do texto, adicionaremos uma pequena imagem. A imagem está em um arquivo chamado seta.gif, o qual deve estar gravado na pasta Chap7. Para o segundo botão vamos aplicar um efeito onmouseover, ou seja, quando o usuário estiver com o mouse sobre alguma parte do botão, faremos com que a cor de segundo plano do botão seja modificada. Quando o mouse sair da área do botão, volta a cor de segundo plano original.

Na Listagem 7.9 temos a página do exemplo proposto.

### Listagem 7.9 – O controle HtmButton – chap7ex10.aspx.

```
<html>
<head>
 <script language="C#" runat="server">

 void Button1_OnClick(object Source, EventArgs e)
 {
 Span1.InnerHtml="Você clicou no botão 1";
 }
 </script>
</head>
<body>
 <form>
 <input type="button" value="Botão 1" OnServerClick="Button1_Click" />
 <input type="button" value="Botão 2" OnServerClick="Button2_Click" />

 </form>
</body>
</html>
```

```
void Button2_OnClick(object Source, EventArgs e)
{
 Span1.InnerHtml="Você clicou no botão 2";
}

```

</script>

</head>

<body>

<h3><font face="Verdana">Exemplo do controle HtmlButton!</font></h3>

<form runat="server">

<font face="Verdana" size="-1">

<p>

<button id="Button1"
 OnServerClick="Button1\_OnClick"
 style="font: 8pt verdana;
 background-color:lightgreen;
 border-color:black;
 height=30;
 width:110"
 runat="server">
  Clique em mim!
</button>

Além do texto temos uma figura no botão.

<p>

<p>

<button id=Button2
 OnServerClick="Button2\_OnClick"
 style="font: 8pt verdana;
 background-color:lightgreen;
 border-color:black;
 height=30;
 width:110"
 onmouseover="this.style.backgroundColor='yellow'"
 onmouseout="this.style.backgroundColor='lightgreen'"
 runat="server">

```

 Clique em mim!
</button>

Com efeito onmouseover.

<p>
<p>

</form>
</body>
</html>

```

Digite o código da Listagem 7.9 e salve o mesmo em um arquivo chamado chap7ex10aspx, na pasta chap7, dentro da pasta wwwroot, conforme descrito anteriormente. Para acessar esta página utilize o seguinte endereço: http://localhost/chap7/chap7ex10.aspx

Clique no primeiro botão. Observe que o texto “Você clicou no botão 1” é exibido. Agora passe o mouse sobre o segundo botão. Observe que a cor de fundo trocou para amarelo. Afaste o mouse do segundo botão. Observe que a cor de fundo voltou ao normal. Agora clique no segundo botão. O texto “Você clicou no botão 2” é exibido, conforme indicado na Figura 7.14.

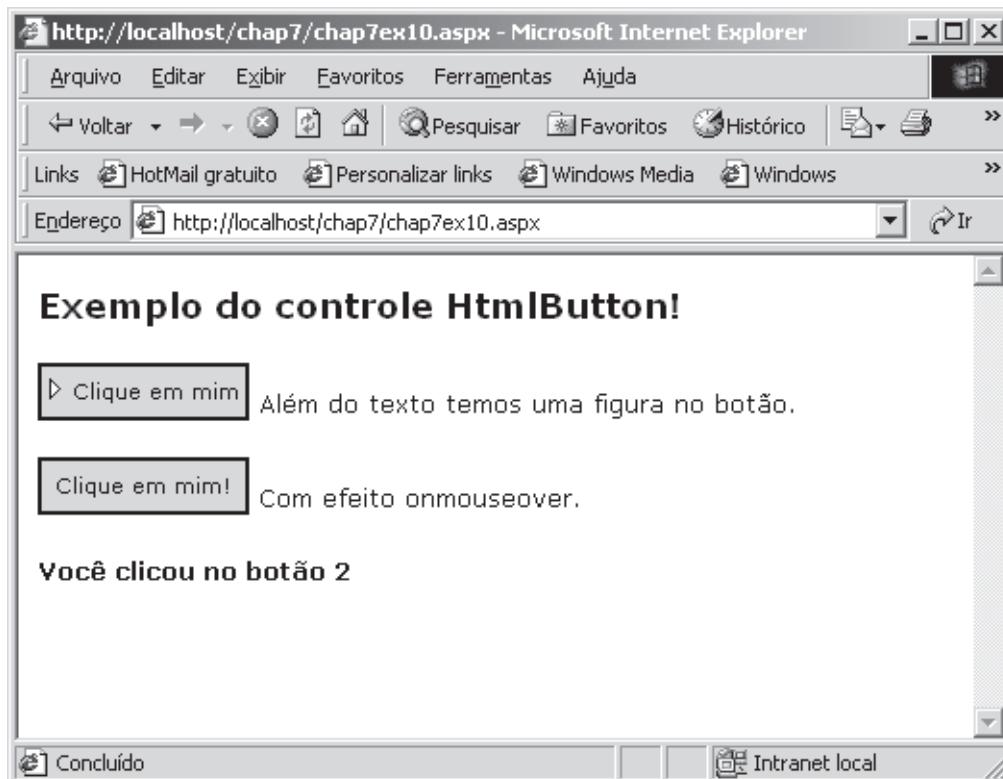


Figura 7.14: O controle **HtmlButton** – chap7ex10.aspx.

Alguns comentários sobre o exemplo.

- ◆ Mais uma vez utilizamos o evento Click de cada botão para enviar uma mensagem informando qual botão foi clicado.
- ◆ A aparência dos botões foi definida através da sua propriedade style. Por exemplo, para o primeiro botão fizemos as seguintes definições:

```
style="font: 8pt verdana;
background-color:lightgreen;
border-color:black;
height=30;
width:110"
```

Observe que através da propriedade style podemos definir uma série de elementos visuais para o botão. Neste exemplo estamos definindo as seguintes propriedades:

- ◆ **font**: tamanho e tipo da fonte.
- ◆ **background-color**: cor de segundo plano.
- ◆ **border-color**: cor das bordas do botão.
- ◆ **height**: altura do botão, em pixels.
- ◆ **width**: largura do botão, em pixels.
- ◆ Para o segundo botão, utilizamos os eventos onmouseover para definir uma cor diferente quando o mouse estiver sobre o botão e onmouseout para retornar a cor original, quando o mouse fosse afastado do botão, conforme indicado no trecho de código a seguir:

```
onmouseover="this.style.backgroundColor='yellow'"
onmouseout="this.style.backgroundColor='lightgreen'"
```

**NOTA:** Para uma referência completa a todas as definições de estilo existentes, consulte [www.w3.org](http://www.w3.org) ou [www.wdvl.com](http://www.wdvl.com).

## HtmlImage Control

É utilizado para Inserir figuras em uma página ASP.NET. A sua funcionalidade é idêntica à tag <img> da linguagem HTML, com a diferença de que, em sendo um controle de servidor, temos acesso a uma série de métodos e eventos do controle.

Sintaxe para o controle HtmlImage:

```

```

Vamos a um exemplo adaptado da documentação do Framework .NET.

Exemplo: Vamos criar um formulário onde é exibida uma imagem. Também exibimos uma lista de seleção com nomes de outras imagens. Esta lista é criada utilizando o controle HtmlSelect explicado anteriormente. O usuário seleciona um nome de imagem na lista e clica no botão Aplicar. O evento Click do botão faz com que a imagem associada à opção selecionada na lista seja exibida.

Na Listagem 7.10 temos a página do exemplo proposto.

### Listagem 7.10 – O controle HtmlImage – chap7ex11.aspx.

```
<html>
<head>

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

 void SubmitBtn_Click(Object Sender, EventArgs e)
 {
 Image1.Src=Select1.Value;
 }

</script>
</head>

<body>
 <h3>Exemplo do controle HtmlImage!!!</h3>

 <form runat="server">

 <p>
 <p>
 Seleccione a Imagem a ser exibida e clique em Aplicar:

 <select id="Select1" runat="server">
 <option Value="Cereal1.gif">Healthy Grains</option>
 <option Value="Cereal2.gif">Corn Flake Cereal</option>
 <option Value="Cereal3.gif">U.F.O.S</option>
 <option Value="Cereal4.gif">Oatey O's</option>
```

```

<option Value="Cereal5.gif">Strike</option>
<option Value="Cereal7.gif">Fruity Pops</option>
</select>

<p>
<p>

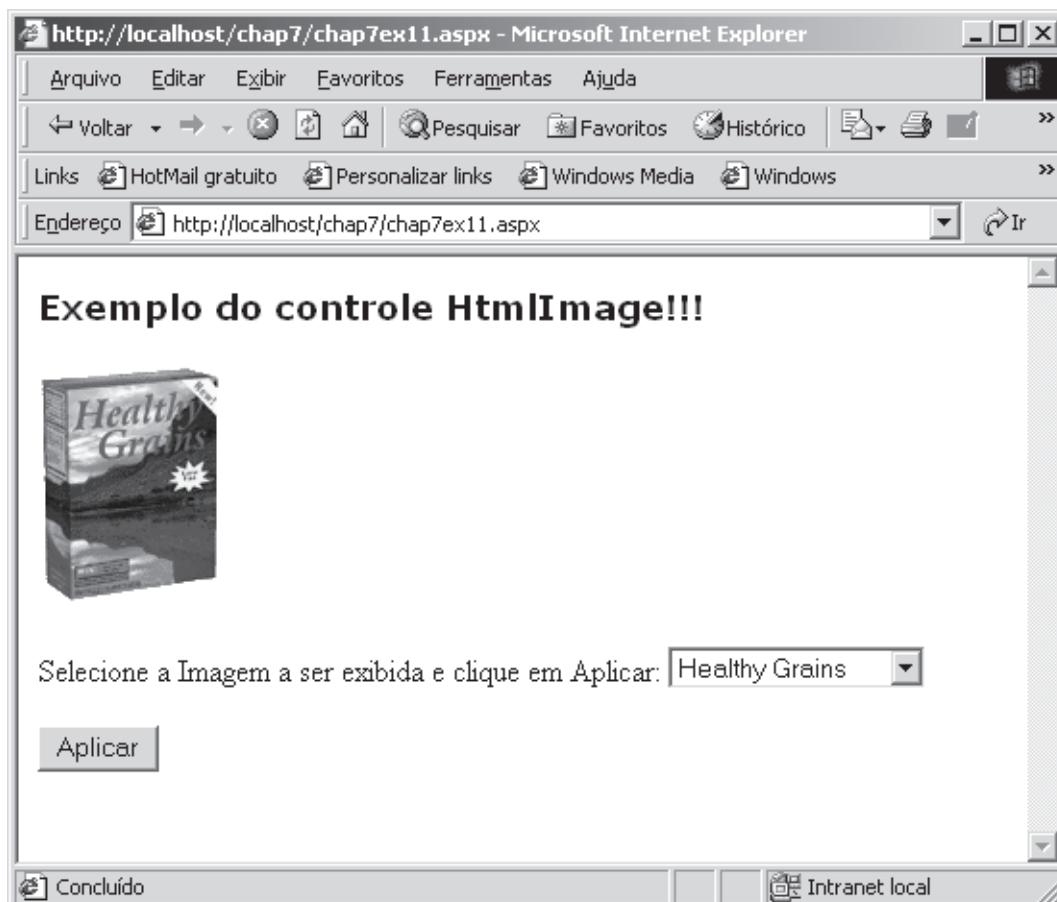
<input type="submit" runat="server" Value="Aplicar" OnServerClick="SubmitBtn_Click">
</form>

</body>
</html>

```

Digite o código da Listagem 7.10 e salve o mesmo em um arquivo chamado chap7ex11.aspx, na pasta chap7, dentro da pasta wwwroot, conforme descrito anteriormente. Para acessar esta página utilize o seguinte endereço: <http://localhost/chap7/chap7ex11.aspx>

Será exibida a figura para o cereal “Healthy Grains”, conforme indicado na Figura 7.15.



**Figura 7.15: O controle HtmlImage – chap7ex11.aspx.**

Selecione uma outra opção na lista; por exemplo, selecione U.F.O.S e dê um clique no botão Aplicar. A figura correspondente será exibida.

Alguns comentários sobre o exemplo.

- ◆ Na seção de apresentação temos um formulário com três controles. Um controle para exibição da imagem, um para a criação da lista com as opções disponíveis e um botão de comando.
- ◆ Utilizamos o evento Click do botão de comando para redefinir a propriedade Src do controle que exibe a imagem. A propriedade Src define o caminho para a imagem a ser exibida. Aqui temos mais um exemplo do poder dos controles de servidor, ou seja, conseguimos alterar suas propriedades através de código de programação. O evento Click para o botão de comando está indicado no trecho de código a seguir:

```
void SubmitBtn_Click(Object Sender, EventArgs e)
{
 Image1.Src=Select1.Value;
}
```

- ◆ Também é interessante observarmos o código que é retornado, como resultado do processamento da página. Por exemplo, quando você seleciona U.F.O.S e clica em aplicar, a figura associada a opção U.F.O.S é exibida. Agora vamos dar uma olhada no código HTML que foi enviado pelo servidor, como resultado do processamento da página.

**NOTA:** Para que este exemplo funcione, as figuras devem estar na pasta Chapt7. Se as figuras estiverem em outra pasta, você precisa fornecer o caminho ao definir o valor da propriedade Src, do controle HtmlImage.

Para visualizar o código HTML correspondente, utilize o seguinte comando no Internet Explorer: Exibir -> Código-fonte. Selecionando este comando será exibido o código a seguir:

```
<html>
<head>

</head>

<body>
 <h3>Exemplo do controle HtmlImage!!!</h3>
 <form name="ctrl10" method="post" action="chap7ex11.aspx" id="ctrl10">

 <input type="hidden" name="__VIEWSTATE" value="dDwtMTkzMzU5NzEzMDt0PDtsPGk8Mj47
 PjtsPHQ8O2w8aTwxPjs+O2w8dDxwPGw8c3JjOz47bDxDZXJ1YWwzLmdpZjs+Pjs7Pjs+Pjs+" />

 <p>
 <p>
 Seleccione a Imagem a ser exibida e clique em Aplicar:

```

```

<select name="Select1" id="Select1">

<option value="Cereal1.gif">Healthy Grains</option>
<option value="Cereal2.gif">Corn Flake Cereal</option>
<option selected="selected" value="Cereal3.gif">U.F.O.S</option>
<option value="Cereal4.gif">Oatey O's</option>
<option value="Cereal5.gif">Strike</option>
<option value="Cereal7.gif">Fruity Pops</option>

</select>
<p>
<p>
<input name="ctrl1" type="submit" value="Aplicar" />
</form>
</body>
</html>

```

Nada de diferente, com exceção do seguinte elemento:

```
<input type="hidden" name="__VIEWSTATE" value="dDwtMTkzMzU5NzEzMDt0PDtsPGk8Mj47PjtsPHQ8O2w8aTwxPjs+O2w8dDxwPGw8c3JjOz47bDxDZXJ1YWwzLmdpZjs+Pjs7Pjs+Pjs+Pjs+" />
```

Este é um campo oculto (type="hidden"), o qual é utilizado para manter o estado dos controles do formulário entre uma chamada e outra da página. Para maiores informações sobre a manutenção de estado consulte os tópicos iniciais deste capítulo.

## Conclusão

Quando uma página .aspx é acessada, o servidor compila a página em um objeto derivado da classe Page. Cada elemento da página é compilado como um objeto. Desta forma temos acesso a métodos e propriedades dos diversos elementos.

Iniciamos o capítulo estudando a classe page. Apresentamos as principais propriedades, eventos e métodos desta classe. Também falamos sobre o processamento de uma página .aspx. Falamos sobre os conceitos de PostBack e round-trip.

Em seguida passamos à definição do que são Html Server controls e quais as vantagens em utilizá-los em substituição às tags HTML tradicionais.

Uma vez entendidos os conceitos teóricos necessários, passamos ao estudo dos principais HTML Server Controls. Estudamos e apresentamos exemplos dos seguintes controles:

- ◆ HTMLForm control.
- ◆ HTMLInputText control.
- ◆ HTMLInputCheckBox control.
- ◆ HtmlTextArea control.
- ◆ HTMLInputRadioButton control.
- ◆ HtmlTable control.
- ◆ HtmlTableRow control.
- ◆ HtmlTableCell control.
- ◆ HTMLSelect control.
- ◆ HTMLAnchor control.
- ◆ HtmlInputButton control.
- ◆ HtmlButton control.
- ◆ HtmlImage control.

No próximo capítulo trataremos de controles que já vêm com uma certa “inteligência” embutida. Veremos os Validation Server Controls.

# Introdução

# CAPÍTULO

8

## Validation Server Controls

Para explicarmos mais esta novidade do ASP.NET: Validation Server Controls, vamos novamente fazer uma analogia com as versões anteriores do ASP. O assunto em pauta é “validação da entrada de dados em formulário”.

Vamos imaginar que você cria um formulário onde o internauta deva preencher os seguintes dados:

- ◆ Nome
- ◆ Endereço
- ◆ e-mail
- ◆ Fone
- ◆ Profissão
- ◆ Data de Nascimento
- ◆ Valor do crédito pretendido
- ◆ Renda mensal
- ◆ Empresa onde trabalha
- ◆ Endereço do site da empresa

Os campos Nome, Endereço e e-mail são obrigatórios. Além disso, o valor do crédito pretendido não pode ser superior a 20% da Renda mensal. Para criarmos um formulário que funcione de acordo com estes critérios, vamos precisar de uma boa quantidade de código. Quando o usuário preenche os dados e clica no botão enviar, os dados são enviados, para processamento, para a página especificada na propriedade action do formulário. Esta página deve, em primeiro lugar, verificar se os campos obrigatórios foram preenchidos e, em segundo lugar, verificar se o valor do crédito não está acima de 20% da renda mensal. Se as informações digitadas pelo usuário atenderem estes critérios, o processamento continua e os dados são gravados em um banco de dados, como o SQL Server ou o ORACLE. Mas basta que um único campo obrigatório não tenha sido preenchido, para que o formulário tenha que ser enviado de volta para o navegador do cliente, mantendo as informações já digitadas e solicitando que sejam feitas as correções necessárias.

Explicar a lógica que está por trás deste processo de validação já não é uma tarefa simples. Implementar toda esta validação é bastante trabalhoso e exige muita codificação. Para facilitar a vida do desenvolvedor, o ASP.NET apresenta uma série de controles que já vêm com funcionalidades, no próprio controle, as quais facilitam a validação dos dados digitados pelo usuário. Estes controles são conhecidos como: Validation Controls ou Server Validation Controls, uma vez que são controles de servidor, a exemplo dos HTML Server Controls que vimos no Capítulo 7.

Neste capítulo vamos entender exatamente o que são os Server Validation Controls. Também aprenderemos a utilizar os diversos controles disponíveis. Iremos estudar os seguintes controles:

- ◆ RequiredFieldValidator Control
- ◆ CompareValidator Control
- ◆ RangeValidator Control
- ◆ CustomValidator Control
- ◆ RegularExpressionValidator Control

Veremos diversos exemplos de utilização destes controles.

## Validation Controls: Definição e Propriedades Gerais

Para facilitar a tarefa de fazer a validação dos dados digitados em um formulário é que foram criados os controles de validação – Validation Controls.

Por que estes controles não são chamados de HTMLServer Validation Controls?

No Capítulo 7 nós vimos que, para cada tag HTML utilizada para a criação de formulários, existe um HTML Server Control correspondente. A vantagem dos HTML Server Controls é que os mesmos são compilados como objetos em uma página ASP.NET. Sendo objeto temos acesso aos métodos e propriedades do controle. Já os controles de validação não possuem correspondentes no HTML. São na verdade controles completamente novos e não uma melhoria em relação a controles já existentes.

Os controles de validação possuem as mesmas características que os chamados Web Server Controls (que serão estudados no Capítulo 9). Estes controles iniciam com a palavra `asp`, conforme exemplo a seguir, onde temos uma pequena amostra da sintaxe do controle `RequiredFieldValidator`:

```
<asp:RequiredFieldValidator
 Atributo1
 Atributo2
 ...
 AtributoN>
</asp:RequiredFieldValidator>
```

**NOTA:** Estudaremos o controle `RequiredFieldValidator` em detalhes ainda neste capítulo.

Outra diferença é quanto à localização das classes bases, ou seja, das classes das quais são derivados os controles. Os HTMLServer controls são derivados de classes do namespace `System.Web.UI.HtmlControls`; já os Web Server Controls, dos quais fazem parte os controles de validação, são derivados de classes contidas no namespace `System.Web.UI.WebControls`.

## Como é que Utilizamos os Controles de Validação?

Na criação do formulário, devemos associar um controle de validação com cada campo onde os dados devam ser validados. Por exemplo, se existe um campo Nome, o qual é de preenchimento obrigatório, devemos associar um campo RequiredFieldValidator com o campo nome que é obrigatório. Quando o usuário envia a página para processamento, cada controle de validação irá checar o valor digitado no controle associado para verificar se está tudo OK, ou seja, para verificar se o formulário passou no teste de validação. Basta que um único campo não atenda as regras de validação definidas, para que o formulário não passe no teste de validação. Neste caso podemos exibir uma mensagem de erro descrevendo o(s) controle(s) que não passou(aram) na validação, em um controle do tipo ValidationSummary.

Por serem controles de servidor, mais especificamente, Web Server Controls, os controles de validação possuem uma série de funcionalidades embutidas que os tornam extremamente interessantes para a criação de aplicações Web. Dizemos que foi embutida “inteligência” nestes controles. Estas funcionalidades, nas versões anteriores do ASP, tinham que ser codificadas manualmente pelo programador. Ao embutir funcionalidades nos próprios controles estamos reduzindo a quantidade de código que precisa ser escrita. Esta frase já foi repetida n vezes neste livro e não me canso de repetir: “Com o Framework .NET o programador precisa se preocupar menos com funções básicas, as quais passaram para o controle do Framework .NET, precisando apenas cuidar da lógica da aplicação”.

Uma das funcionalidades dos controles de validação é que eles são capazes de, automaticamente, detectar qual o navegador ou dispositivo que o cliente está utilizando, e se for o Internet Explorer 5 ou superior, será gerado código de validação no próprio cliente, no momento da criação da página. Em outras palavras, ao carregar uma página, no IE 5 ou superior, página esta que contém controles de validação, será gerado, automaticamente, código para fazer a validação no próprio cliente, evitando que a página seja enviada para o servidor se algum controle não tiver passado no teste de validação. Nesta situação a página somente será enviada para processamento quando todos os controles tiverem passado no teste de validação. Para criar esta mesma funcionalidade nas versões anteriores do ASP, tínhamos que escrever uma boa quantia de código. O código de validação gerado automaticamente, no cliente, é criado utilizando-se da linguagem DHTML – Dynamic HTML. Para maiores informações sobre DHTML consulte o seguinte endereço: [www.wdvl.com](http://www.wdvl.com).

Se quisermos é possível desabilitar a validação no cliente. Para isso temos uma propriedade chamada EnableClientScript, da classe BaseValidator, da qual todos os controles de validação são derivados. A propriedade EnableClientScript é do tipo Boolean – True ou False. Estudaremos a classe BaseValidator logo em seguida. Ao desabilitarmos a validação no cliente (definindo EnableClientScript = False), podemos criar mensagens personalizadas de erro, ao invés de utilizar as mensagens pré-definidas e o controle ValidationSummary.

**IMPORTANTE:** Mesmo que o código para validação no cliente tenha sido gerado, quando a página é enviada para o servidor, a validação é feita pelos Validation Controls, para garantir que resultados incorretos sejam inseridos no banco de dados. Pode acontecer de a página ter sido modificada no caminho entre o navegador do cliente e o servidor IIS. Neste caso os dados que foram digitados e passaram na validação foram modificados por um hacker e não são os mesmos que chegaram no servidor. Por isso, a validação no servidor funciona como uma proteção extra. Claro que isso não dispensa o uso de outras técnicas de segurança como por exemplo a criptografia e Certificados Digitais. Falaremos um pouco mais sobre Segurança, no último capítulo deste livro.

## A Mãe de Todos? Ou Seria o Pai de Todos?

Todos os controles de validação são derivados de uma classe base chamada BaseValidator. Esta classe faz parte do namespace System.Web.UI.WebControls. Por serem baseados na classe BaseValidator, os controles de validação herdam as propriedades e métodos desta classe.

### Principais Propriedades da Classe BaseValidator

Neste tópico vamos apresentar as principais propriedades da classe BaseValidator. Para uma descrição completa de todos os métodos e propriedades da classe BaseValidator, abra a documentação do Framework .NET (Iniciar -> Programas -> Microsoft .NET Framework SDK -> Documentation) e, uma vez dentro da documentação, siga o seguinte caminho:

**.NET Framework SDK -> .NET Framework Reference -> .NET Framework Class Library -> System.Web.UI.WebControls -> BaseValidator Class.**

Vamos ao estudo das principais propriedades.

- ◆ **ControlToValidate:** Define ou retorna o nome do controle que será associado ao controle de validação.
- ◆ **EnableClientScript:** Habilita/desabilita a geração automática de código de validação no cliente, quando o navegador suporta tal funcionalidade. Esta propriedade é do tipo Boleana, podendo assumir os valores True ou False. Pode ser utilizada para retornar ou definir o valor desta propriedade.
- ◆ **Enabled:** Propriedade do tipo Boleana (True ou False) que pode ser utilizada para definir ou retornar um valor que indica se a validação está habilitada (True) ou desabilitada (False), para o controle.
- ◆ **ErrorMessage:** É utilizada para definir o texto da mensagem de erro associada ao controle de validação. Também pode ser utilizada para obter o texto da mensagem de erro associada.
- ◆ **ForeColor:** Utilizada para definir ou retornar a cor da mensagem de erro que será exibida quando a validação falhar.
- ◆ **ID:** Define um identificador associado ao controle, identificador este que é utilizado no código para acessar as propriedades e métodos do controle.
- ◆ **IsValid:** Propriedade do tipo Boleana (True ou False), que indica se a validação ocorreu com sucesso ou não. Por exemplo, se o controle de validação é do tipo que exige uma entrada obrigatória (RequiredFieldValidator) e o usuário não digita nada no campo associado ao controle de validação, o teste de validação irá falhar e esta propriedade conterá o valor False, indicando que o controle não passou no teste de validação.

### Principais Métodos da Classe BaseValidator

Vamos descrever alguns métodos da classe BaseValidator e apresentar um exemplo prático. O objetivo deste primeiro exemplo é ver os controles de validação em funcionamento. Não iremos explicar o código do exemplo neste momento. À medida que formos avançando no capítulo e explicando os diversos controles de validação disponíveis, você poderá voltar a este exemplo e facilmente entender o código do mesmo.

Principais métodos da classe BaseValidator:

- ◆ **Validate:** É responsável por fazer a validação do controle associado com o controle de validação e atualizar a propriedade IsValid para True se a validação ocorrer com sucesso ou para False, caso contrário.
- ◆ **RegisterValidatorCommonScript:** Pode ser utilizado para registrar código na página, para que ocorra a validação no cliente.

## Um Exemplo, só Para Começar

Vamos apresentar um pequeno exemplo, para que possamos ver os controles de validação em funcionamento. Conforme explicado anteriormente, não nos deteremos na explicação do código do exemplo, neste momento.

No exemplo proposto, temos um formulário com um campo para digitação, onde o usuário deve digitar um valor entre 1 e 10. Se o usuário não digitar nada, ao clicar no botão Enviar será exibida a seguinte mensagem: “Você não digitou nenhum número!!!”. Quando o usuário digita um número entre 1 e 10, a propriedade IsValid torna-se verdadeira e o evento Click do botão Enviar será executado. Neste evento é gerado um número aleatório entre 1 e 10. Se o número digitado pelo usuário for igual ao número gerado aleatoriamente, é exibida a seguinte mensagem: “Parabéns, você acertou!!!”; caso contrário será exibida a seguinte mensagem: “Sinto muito, você errou!!!”.

**NOTA:** O exemplo apresentado é baseado em um exemplo da documentação do Framework .NET.

Na Listagem 8.1 temos o código para o exemplo proposto.

### Listagem 8.1 – O primeiro exemplo com controles de validação – chap8ex1.aspx.

```
<html>
<head>
<script language="C#" runat="server">

 void Button_Click(Object sender, EventArgs e)
 {

 Random rand_number = new Random();

 Compare1.ValueToCompare = rand_number.Next(1, 10).ToString();
 Compare1.Validate();

 if (Page.IsValid)
 {
 lblOutput.Text = "Parabéns, você acertou!!!";
 }
 else
 {
```

```
 lblOutput.Text = "Sinto muito, você errou!!!";
 }

 lblOutput.Text += "

" + "O número correto era: " +
Compare1.ValueToCompare;

}
</script>

</head>
<body>

<form runat=server>

<h3>Exemplo de validação.</h3>

<h5>Digite um número entre 1 e 10:</h5>

<asp:RequiredFieldValidator id="Require1"
 ControlToValidate="TextBox1"
 Type="Integer"
 ErrorMessage="Você não digitou nenhum número!"
 Text="*"
 runat="server"/>

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

<asp:CompareValidator id="Compare1"
 ControlToValidate="TextBox1"
 ValueToCompare="0"
 EnableClientScript="False"
 Type="Integer"
 ErrorMessage="Número Incorreto!"
 Text="*"
 runat="server"/>


```

```


<asp:Button id="Button1"
 Text="Enviar"
 OnClick="Button_Click"
 runat="server"/>

<asp:Label id="lblOutput"
 Font-Name="verdana"
 Font-Size="10pt"
 runat="server"/>

<asp:ValidationSummary
 id="Summary1"
 runat="server"/>

</form>

</body>
</html>
```

Digite o código da Listagem 8.1 e salve o mesmo em um arquivo chamado chap8ex1.aspx, na pasta chap8, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap8/chap8ex1.aspx>

Dê um clique no botão Enviar, sem digitar nenhum valor no campo para digitação. Você obterá o resultado indicado na Figura 8.1.

Agora digite um número entre 1 e 10 e clique no botão Enviar. Se você acertar o número gerado pelo código do evento Button\_Click, será exibida a mensagem: “Parabéns, você acertou!!!”; caso contrário será exibida a mensagem indicada na Figura 8.2.

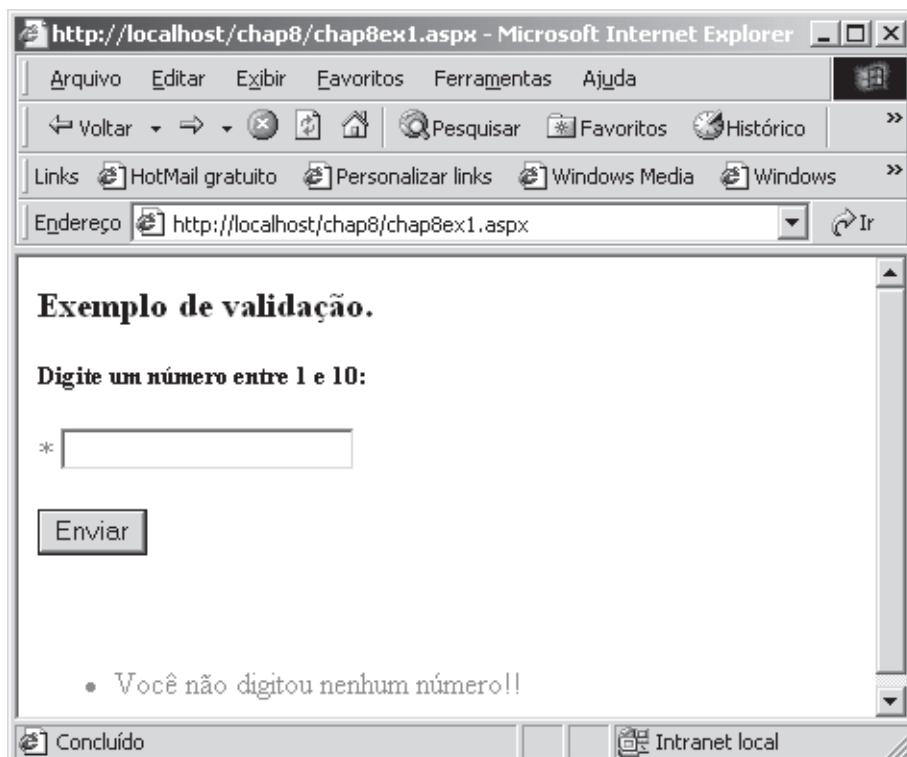


Figura 8.1: Controles de validação – chap8ex1.aspx.

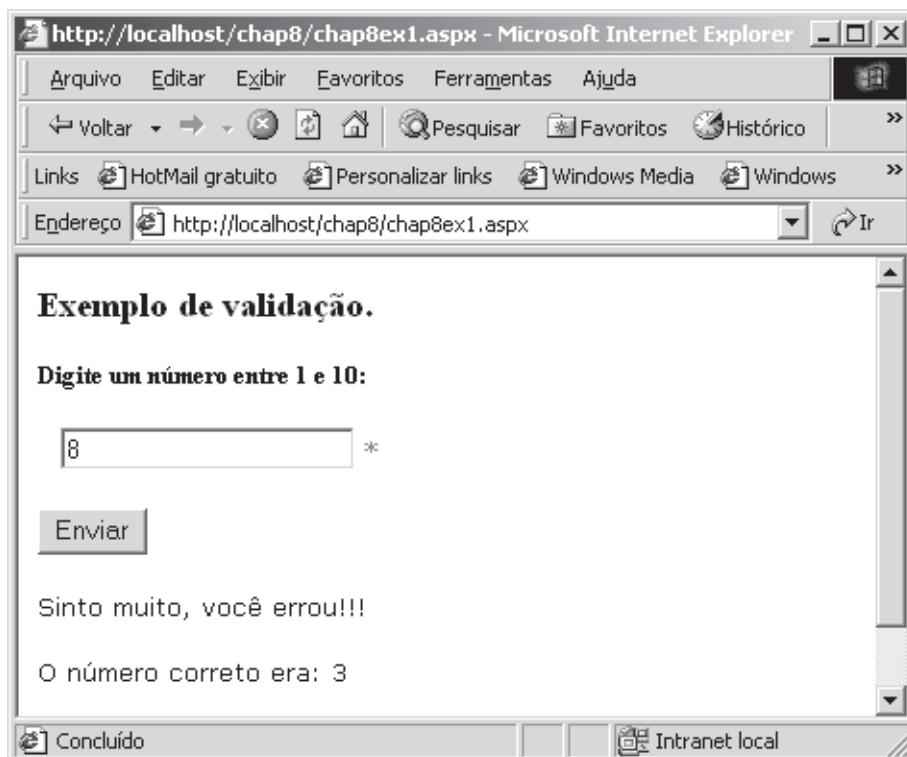


Figura 8.2: O usuário não adivinhou o número gerado aleatoriamente.

Prometi que não iria comentar o código deste primeiro exemplo, mas apenas um comentário rápido, em relação à geração de números aleatórios. Para gerar um número aleatoriamente, entre 1 e 10, declaramos e inicializamos uma variável (talvez seria melhor dizer um objeto) do tipo Random:

```
Random rand_number = new Random();
```

A variável rand\_number é baseada na classe Random, do namespace System. Em seguida utilizamos o método Next da classe Random, para gerar um número aleatório entre 1 e 10.

Agora vamos estudar, em detalhes, os diversos controles de validação do ASP.NET.

## O Controle RequiredFieldValidator

Este controle é utilizado para garantir que um determinado campo seja preenchido, ou seja, não podemos deixar o campo em branco. Conforme descrito anteriormente, os controles de validação são utilizados em conjunto com outros controles do formulário. Por exemplo, se tivermos um campo Nome, em um formulário, podemos associar um controle RequiredFieldValidator com o campo Nome, para garantir que o usuário seja obrigado a preencher o campo Nome.

A sintaxe para este controle é a seguinte:

```
<asp:RequiredFieldValidator
 id="identificação no código"
 runat="server"
 ControlToValidate="nome do controle associado"
 ErrorMessage="Mensagem de erro."
 ForeColor="Red"
 BackColor="Grey"
...
>
</asp:RequiredFieldValidator>
```

Uma sintaxe alternativa é a seguinte:

```
<asp:RequiredFieldValidator
 id="identificação no código"
 runat="server"
 ControlToValidate="nome do controle associado"
 ErrorMessage=" Mensagem de erro."
 ForeColor="Red"
 BackColor="Grey"
...
/>
```

Observe que, ao invés de fechar a tag desta maneira: </asp:RequiredFieldValidator>, simplesmente utilizamos um />. As duas sintaxes são válidas.

Onde podemos utilizar os seguintes atributos:

- ◆ **id:** É um nome que identifica o controle. Este nome é utilizado no código para acessar as propriedades e métodos do controle.
- ◆ **runat="server":** identifica como sendo um controle de servidor. Todos os controles asp: são controles de servidor.
- ◆ **ControlToValidate:** Neste atributo definimos o nome do controle que será validado pelo controle de validação. Por exemplo, se o controle de validação está associado com um campo Nome do formulário, nesta propriedade devemos informar a identificação do controle Nome. A identificação é o valor definido na propriedade id do controle Nome. Esta frase foi repetitiva e enfadonha, de propósito, para que você não esqueça esta associação.
- ◆ **ErrorMessage:** Podemos definir uma mensagem de erro que será exibida, caso a validação do campo falhe. Por exemplo, o usuário não preencheu um campo de preenchimento obrigatório.
- ◆ **ForeColor:** Utilizado para definir a cor do texto da mensagem de erro.
- ◆ **BackColor:** Utilizado para definir a cor de segundo plano do texto da mensagem de erro.

Vamos a um exemplo simples. Vamos criar um formulário onde temos dois campos:

- ◆ e-mail
- ◆ nome

O campo e-mail é de preenchimento obrigatório, já o campo nome é de preenchimento opcional. Para garantir que o campo e-mail seja preenchido vamos associar um controle de validação do tipo RequiredFieldValidator com o campo e-mail. Se o campo email não for preenchido vamos emitir a seguinte mensagem:

**"O email é de preenchimento obrigatório, por favor forneça o seu email e clique no botão Enviar."**

Na Listagem 8.2 temos o código para o exemplo proposto.

### Listagem 8.2 – O controle de validação RequiredFieldValidator.

```
<html>
<head>

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

 void Enviar_Click(Object sender, EventArgs e)
 {
 // Código para gravar os dados recebidos no
 // banco de dados.
 }

</script>

</head>
<body>
```

```
<form runat=server>

 <h3>Exemplo do controle RequiredFieldValidator!!!</h3>

 <asp:RequiredFieldValidator id="Requer_email"
 ControlToValidate="email"
 Type="String"
 ErrorMessage="O email é de preenchimento obrigatório,
 por favor forneça o seu email e clique
 no botão Enviar"
 Text="O e-mail é obrigatório!!"
 ForeColor="Red"
 runat="server"/>

<table>

 <tr>
 <td><h3>Digite o e-mail:</h3> </td>
 <td><input id="email" type=text size=40 runat="server"></td>
 </tr>

 <tr>
 <td><h3>Digite o nome:</h3> </td>
 <td><input id="nome" type=text size=40 runat="server"></td>
 </tr>

 <tr>
 <td><h3>Cadastrar -></h3></td>
 <td><input type=submit value="Enviar"
OnServerClick="Enviar_Click" runat="server">
 </td>
 </tr>

</table>

<asp:ValidationSummary
```

```

 id="Summary1"
 runat="server"/>

</form>

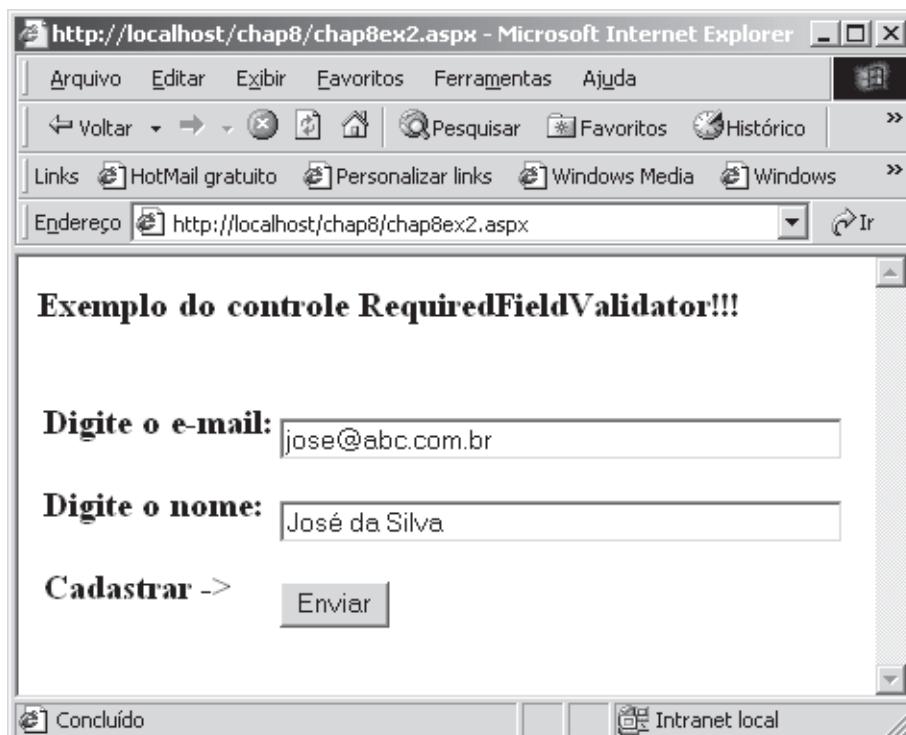
</body>
</html>

```

Digite o código da Listagem 8.2 e salve o mesmo em um arquivo chamado chap8ex2.aspx, na pasta chap8, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap8/chap8ex2.aspx>

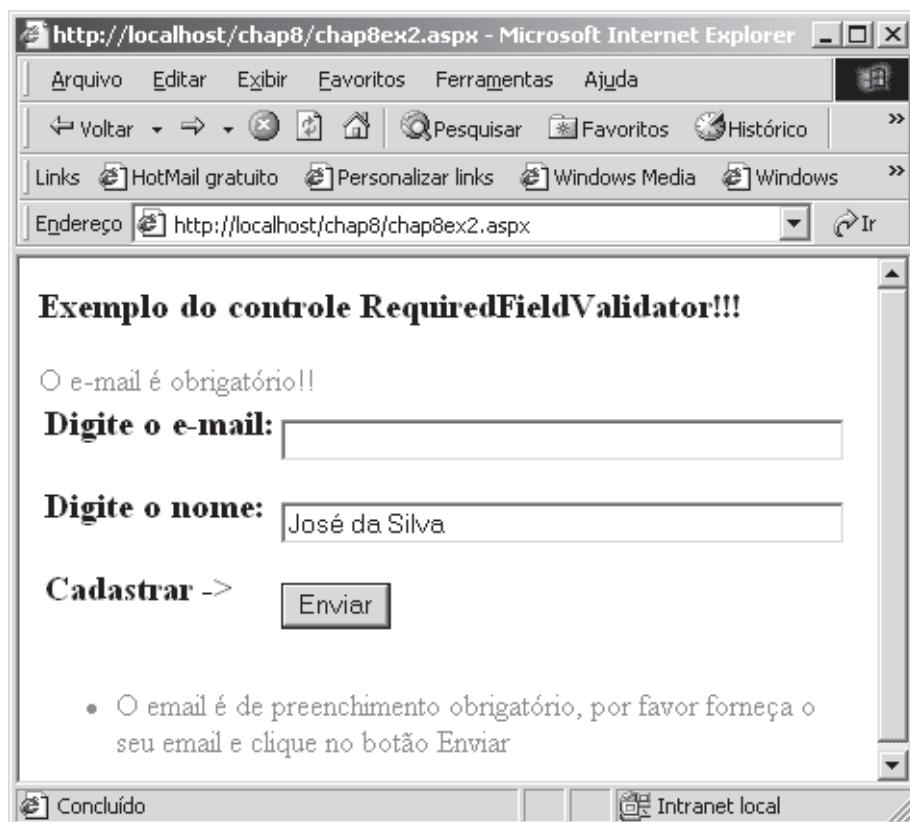
No campo e-mail digite: jose@abc.com.br. No campo Nome digite: José da Silva. Dê um clique no botão Enviar. Você obterá o resultado indicado na Figura 8.3.



**Figura 8.3: Todos os dados obrigatórios foram digitados. – chap8ex2.aspx.**

Deixe o campo e-mail em branco. No campo Nome digite: José da Silva. Dê um clique no botão Enviar. Você obterá o resultado indicado na Figura 8.4.

Observe que neste caso o campo de validação detectou que o campo e-mail não foi preenchido e avisou.



**Figura 8.4:** O controle de validação RequiredFieldValidator entrando em ação.

Vamos entender melhor o código da Listagem 8.2.

- ◆ Na seção de código não fizemos absolutamente nada. Somente colocamos o evento Enviar\_Click e no corpo do procedimento alguns comentários. Por exemplo, no corpo deste procedimento poderíamos detectar se a validação ocorreu com sucesso para todos os controles e, em caso afirmativo, enviar as informações digitadas pelo usuário para um banco de dados. Por enquanto ainda não sabemos fazer a conexão com banco de dados. Aprenderemos a conectar com banco de dados a partir do Capítulo 10.
- ◆ Na seção de apresentação criamos um formulário com dois controles (HtmlServerControls) do tipo texto (type=text). Um para o usuário digitar o e-mail e outro para digitar o nome. Queremos que o campo e-mail seja de preenchimento obrigatório. Para isso utilizamos um controle de validação do tipo RequiredFieldValidator. A inserção do controle de validação é feita com o trecho de código a seguir:

```

<asp:RequiredFieldValidator id="Requer_email"
ControlToValidate="email"
Type="String"
ErrorMessage="O email é de preenchimento obrigatório,
por favor forneça o seu email e clique
no botão Enviar"
Text="O e-mail é obrigatório!@"
ForeColor="Red"
runat="server"/>

```

O atributo mais importante é o seguinte:

```
ControlToValidate="email"
```

Com a definição deste atributo estamos informando que o campo a ser validado, ou seja, o campo que é de preenchimento obrigatório é o campo e-mail do formulário. Neste atributo (ControlToValidate) informamos o ID do campo a ser validado.

O texto do atributo ErrorMessage vai ser exibido em um controle do tipo ValidationSummary. Por isso que incluímos um controle deste tipo, no final do formulário; caso contrário o texto definido no atributo ErrorMessage não seria exibido. No trecho de código a seguir temos a definição do controle ValidationSummary:

```
<asp:ValidationSummary
 id="Summary1"
 runat="server"/>
```

O atributo Text define o texto que será exibido no local onde o controle de validação está definido, caso a validação falhe. Aqui podemos fazer um teste interessante, para comprovar que realmente é gerado código para validação no Cliente, quando o navegador é o Internet Explorer, conforme descrito no início do capítulo. Para provar isto vamos fazer o seguinte teste:

Clique no campo e-mail e apague o que estiver neste campo. Pressione a tecla <TAB> para ir para o campo Nome. Observe que o texto “O e-mail é obrigatório!! (conforme definido no atributo Text do controle de validação) aparece acima do campo e-mail, sem que tenhamos clicado no botão Enviar, conforme indicado na Figura 8.5.

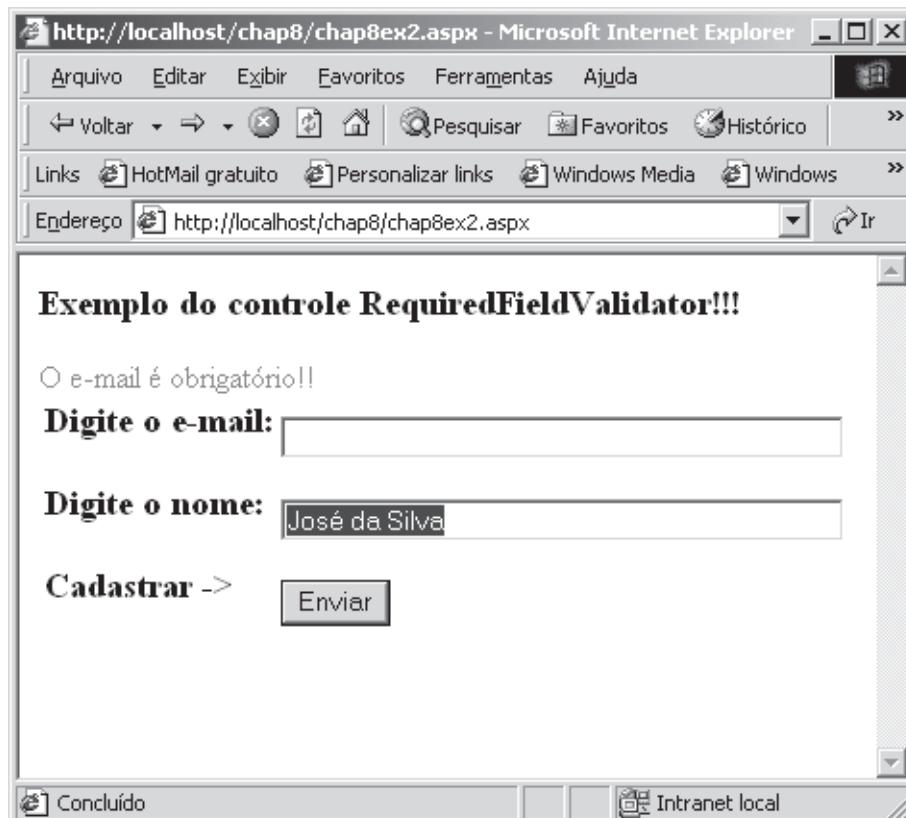


Figura 8.5: A validação no cliente, gerada automaticamente para o IE, em ação.

Se você clicar no botão Enviar, estando o campo e-mail em branco também será exibida a mensagem de erro “O email é de preenchimento obrigatório, por favor forneça o seu e-mail e clique no botão Enviar”. Este texto está definido no atributo ErrorMessage do controle de validação Requer\_email e é exibido no controle Summary1, que é um controle do tipo ValidationSummary, especificamente projetado para exibir mensagens de erro quando algum campo não é validado.

Com este exemplo já podemos ter uma boa idéia do funcionamento dos controles de validação. Se você voltar para o exemplo da Listagem 8.1, já terá condições de entender grande parte do exemplo. Agora vamos continuar estudando os demais tipos de controle de validação disponíveis com o ASP.NET.

## O Controle CompareValidator

Este controle é utilizado para comparar o valor digitado em um campo associado com o controle CompareValidator, com um valor constante, com um valor de outro campo (através da utilização da propriedade value deste outro campo) ou com o valor de um campo de uma tabela em um banco de dados. Se o valor digitado ou selecionado para o campo não for igual ao valor de comparação, o controle não passará no teste de validação.

A sintaxe para este controle é a seguinte:

```
<asp:CompareValidator
 id="identificação no código"
 runat="server"
 ControlToValidate="nome do controle associado"
 ValueToCompare="um valor constante"
 Type="Tipo do valor a ser comparado: string, Int32, Double, etc."
 Operator="Operador de comparação: maior do que, menor do que, etc."
 ErrorMessage="Mensagem de erro."
 ForeColor="Cor da fonte."
 BackColor="Cor de segundo plano"
 >
</asp:CompareValidator>
```

Uma sintaxe alternativa é a seguinte:

```
<asp:CompareValidator
 id="identificação no código"
 runat="server"
 ControlToValidate="nome do controle associado"
 ValueToCompare="um valor constante"
 Type="Tipo do valor a ser comparado: string, Int32, Double, etc."
 Operator="Operador de comparação: maior do que, menor do que, etc."
```

```
ErrorMessage="Mensagem de erro."
ForeColor="Cor da fonte."
BackColor="Cor de segundo plano"
/>>
```

Observe que, ao invés de fechar a tag desta maneira: </asp:CompareValidator>, simplesmente utilizamos um </>. As duas sintaxes são válidas.

Onde podemos utilizar os seguintes atributos:

- ◆ **id:** É um nome que identifica o controle. Este nome é utilizado no código para acessar as propriedades e métodos do controle.
- ◆ **runat="server":** identifica como sendo um controle de servidor. Todos os controles asp: são controles de servidor.
- ◆ **ControlToValidate:** Neste atributo definimos o nome do controle que será validado pelo controle de validação. Por exemplo, se o controle de validação está associado com um campo Nome do formulário, nesta propriedade devemos informar a identificação do controle Nome. A identificação é o valor definido na propriedade id do controle Nome. Esta frase foi repetitiva e enfadonha, novamente, de propósito, para que você não esqueça esta associação.
- ◆ **ValueToCompare:** Define o valor de comparação. Pode ser uma constante, o valor contido em um outro campo ou em um banco de dados.
- ◆ **Type:** Define o tipo de dados do valor de comparação.
- ◆ **Operator:** Define o operador de comparação.
- ◆ **ErrorMessage:** Podemos definir uma mensagem de erro que será exibida, caso a validação do campo falhe. Por exemplo, o usuário não preencheu um campo de preenchimento obrigatório.
- ◆ **ForeColor:** Utilizado para definir a cor do texto da mensagem de erro.
- ◆ **BackColor:** Utilizado para definir a cor de segundo plano do texto da mensagem de erro.

Vamos a um exemplo simples. Vamos criar um formulário onde temos um campo para digitação, onde deve ser digitado um valor menor do que 100. Vamos utilizar um controle CompareValidator para verificar se o número digitado atende o critério de ser menor do que 100.

Na Listagem 8.3 temos o código para o exemplo proposto.

### Listagem 8.3 – O controle de validação CompareValidator.

```
<html>
<head>
 <script language="C#" runat="server">

 void Enviar_Click(Object sender, EventArgs e)
 {
 }
 </script>
</head>
<body>
 <form>
 <input type="text" name="Valor" />

 <asp:CompareValidator ID="CompareValidator1" runat="server" ErrorMessage="O valor deve ser menor que 100.">
 <ControlToValidate>Valor</ControlToValidate>
 <ValueToCompare>100</ValueToCompare>
 <Operator><operator></operator></operator>
 </asp:CompareValidator>
 <input type="button" value="Enviar" onclick="Enviar_Click" />
 </form>
</body>
```

```
// Código para gravar os dados recebidos no
// banco de dados.
}

</script>

</head>
<body>

<form runat=server>

<h3>Exemplo do controle CompareValidator!!!</h3>

<input id="valor" type=text size=40 runat="server">

<asp:CompareValidator
id="valida_valor"
runat="server"
ControlToValidate="valor"
ValueToCompare="100"
Type="Double"
Operator="LessThan"
ErrorMessage="Digite um valor menor do que 100."
ForeColor="Blue"
BackColor="Cyan"
>
</asp:CompareValidator>

<input type=submit value="Enviar" OnServerClick="Enviar_Click" runat="server">

</form>

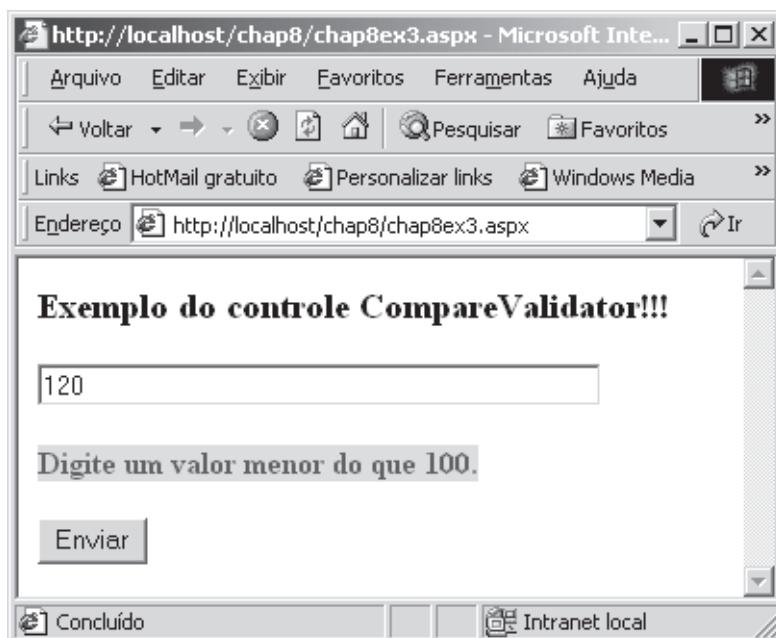
</body>
</html>
```

Digite o código da Listagem 8.3 e salve o mesmo em um arquivo chamado chap8ex3.aspx, na pasta chap8, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap8/chap8ex3.aspx>

No campo valor digite 50. Dê um clique no botão Enviar. Observe que nenhuma mensagem de erro é emitida.

Agora digite 120 no campo valor. Dê um clique no botão Enviar. Você obterá uma mensagem de erro, conforme indicado na Figura 8.6.



**Figura 8.6: O controle de validação CompareValidator entrando em ação.**

Observe que neste caso o campo de validação detectou que o campo valor possui um valor maior do que 100 e a mensagem de erro foi exibida.

Vamos comentar a definição de duas propriedades do controle CompareValidator:

- ◆ **ValueToCompare="100"** : Esta propriedade define um valor constante para comparação com o valor digitado.
- ◆ **Operator="LessThan"**: Nesta propriedade definimos o operador de comparação. No exemplo definimos menor do que (LessThan). Na Tabela 8.1 temos a definição dos operadores permitidos para esta propriedade.

**Tabela 8.1 Valores para a propriedade Operator**

Valor	Descrição
Equal	Igual a.
GreaterThan	Maior do que.
GreaterThanOrEqual	Maior ou igual a.

Valor	Descrição
LessThan	Menor do que.
LessThanEqual	Menor ou igual a.
NotEqual	Diferente.

Ao invés de especificar um valor constante de comparação, utilizando a propriedade ValueToCompare, podemos utilizar a propriedade ControlToCompare, para utilizar, para comparação, o valor contido em outro controle do formulário. Por exemplo, se tivéssemos um campo chamado ValorDeReferência, no formulário do exemplo da Listagem 8.3, poderíamos definir o controle CompareValidator da seguinte maneira:

```
<asp:CompareValidator
 id="valida_valor"
 runat="server"
 ControlToValidate="valor"
 ControlToCompare="ValorDeReferência"
 Type="Double"
 Operator="LessThan"
 ErrorMessage="Digite um valor menor do que 100."
 ForeColor="Blue"
 BackColor="Cyan"
 >
</asp:CompareValidator>
```

**NOTA:** Estes valores estão definidos em uma estrutura do tipo Enumeration, chamada ValidationCompareOperator. Esta estrutura faz parte do namespace System.Web.UI.WebControls.

Com esta modificação, o valor digitado no campo valor será comparado com o valor do controle ValorDeReferência, devendo ser menor do que este valor, conforme definição da propriedade Operator="LessThan".

## O Controle RangeValidator

No item anterior estudamos o controle CompareValidator, o qual permite fazer validações com base em um determinado valor e um operador de comparação. Para comparações que envolvem uma faixa de valores (com limites inferior e superior), precisamos utilizar o controle RangeValidator. Por exemplo, com o controle RangeValidator podemos verificar se os valores digitados no campo Salário estão entre 1000 e 5000.

A sintaxe para este controle é a seguinte:

```
<asp:RangeValidator
 id="identificação no código"
 runat="server"
```

```

 ControlToValidate="nome do controle associado"
 MinimumValue="limite inferior da faixa"
 MaximumValue="limite superior da faixa"
 Type="Tipo do valor a ser comparado: string, Int32, Double, etc."
 ErrorMessage="Mensagem de erro."
 ForeColor="Cor da fonte."
 BackColor="Cor de segundo plano"
/>
</asp:RangeValidator>

```

Uma sintaxe alternativa é a seguinte:

```

<asp:RangeValidator
 id="identificação no código"
 runat="server"
 ControlToValidate="nome do controle associado"
 MinimumValue="limite inferior da faixa"
 MaximumValue="limite superior da faixa"
 Type="Tipo do valor a ser comparado: string, Int32, Double, etc."
 ErrorMessage="Mensagem de erro."
 ForeColor="Cor da fonte."
 BackColor="Cor de segundo plano"
/>

```

Observe que, ao invés de fechar a tag desta maneira: </asp:RangeValidator>, simplesmente utilizamos um />. As duas sintaxes são válidas.

Vamos destacar os seguintes atributos:

- ◆ MinimumValue: Define o limite inferior da faixa.
- ◆ MaximumValue: Define o limite superior da faixa.

Vamos a um exemplo simples. Vamos criar um formulário onde temos um campo para digitação, onde deve ser digitado um valor maior ou igual a 50 e menor ou igual a 100. Vamos utilizar um controle RangeValidator para verificar se o número digitado atende o critério de estar na faixa entre 50 e 100.

Na Listagem 8.4 temos o código para o exemplo proposto.

#### Listagem 8.4 – O controle de validação RangeValidator.

```

<html>
<head>

```

**NOTA:** Os valores definidos em MinimumValue e MaximumValue também fazem parte da faixa de valores permitidos. Por exemplo, se MinimumValue for definido em 20 e MaximumValue for definido em 50; os valores permitidos são: Maior ou igual a 20 e Menor ou igual a 50.

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

void Enviar_Click(Object sender, EventArgs e)
{
 // Código para gravar os dados recebidos no
 // banco de dados.
}

</script>

</head>
<body>

<form runat=server>

 <h3>Exemplo do controle RangeValidator!!!</h3>

 <input id="valor" type=text size=40 runat="server">

 <asp:RangeValidator
 id="valida_valor"
 runat="server"
 ControlToValidate="valor"
 MinimumValue="20"
 MaximumValue="50"
 Type="Double"
 ErrorMessage="Digite um valor entre 20 e 50!!"
 ForeColor="Blue"
 BackColor="Cyan"
 >
 </asp:RangeValidator>

 <input type=submit value="Enviar" OnServerClick="Enviar_Click" runat="server">

</form>
```

```
</form>
```

```
</body>
```

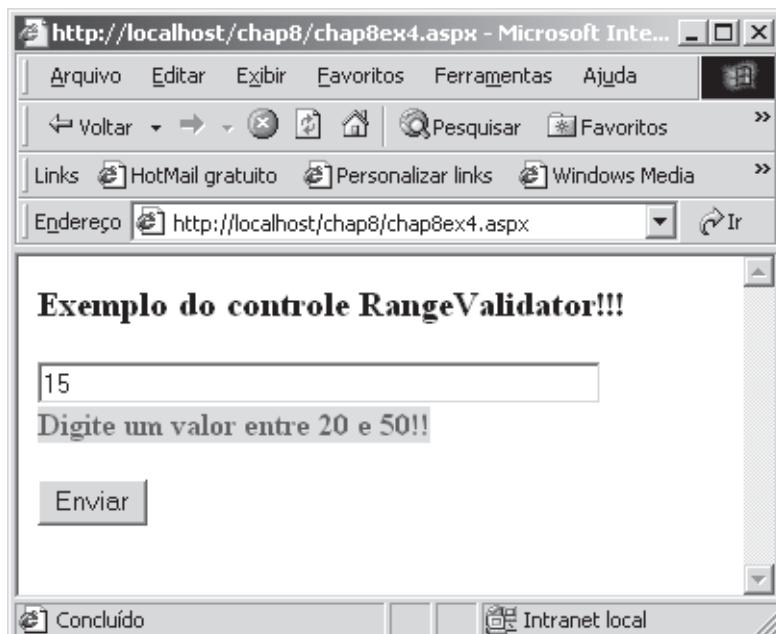
```
</html>
```

Digite o código da Listagem 8.4 e salve o mesmo em um arquivo chamado chap8ex4.aspx, na pasta chap8, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap8/chap8ex4.aspx>

No campo valor digite 35, que é um valor dentro da faixa definida no controle RangeValidator: entre 20 e 50. Dê um clique no botão Enviar. Observe que nenhuma mensagem de erro é emitida.

Agora digite 15 no campo valor. Dê um clique no botão Enviar. Você obterá uma mensagem de erro, conforme indicado na Figura 8.7.



**Figura 8.7:** O controle de validação RangeValidator entrando em ação.

Observe que neste caso o campo de validação detectou que o campo valor possui um valor fora da faixa (entre 20 e 50) e a mensagem de erro foi exibida.

## O Controle CustomValidator

Este controle permite que criemos nossas próprias regras de validação. Por exemplo, podemos precisar de um controle de validação para verificar se o número digitado em um controle é divisível por 5. Outro exemplo poderia ser a criação de um controle para fazer a validação do dígito verificador de um campo CPF ou CNPJ. Para validações complexas,

deste tipo, não temos controles prontos, precisamos criar um controle personalizado, utilizando como base o controle CustomValidator e definindo as funções de validação correspondentes.

A sintaxe para este controle é a seguinte:

```
<asp:CustomValidator
 id="identificação no código"
 runat="server"
 ControlToValidate="controle a ser validado"
 ClientValidationFunction="Nome da função que fará a validação - para validação no
 Cliente.
 OnServerValidate="Nome do procedimento na seção de código da página - para
 validação no servidor"
 ErrorMessage="Mensagem de erro"
 ForeColor="Cor do texto da mensagem de erro."
 BackColor="Cor de segundo plano da mensagem de erro."
 >
</asp:CustomValidator>
```

A validação pode ser definida para acontecer no cliente, sem que a página tenha sido enviada para processamento ou no servidor, quando a página é enviada para processamento.

Para a validação no cliente, especificamos o nome da função de validação, no atributo ClientValidationFunction. A função deve ser escrita em uma linguagem suportada pelo navegador do cliente, como por exemplo VBScript ou JScript.

Para a validação no servidor, especificamos o nome de um procedimento de validação, no atributo OnServerValidate. A função no servidor pode ser escrita em qualquer linguagem suportada pelo Framework .NET, como por exemplo: VB.NET, C#, JScript.NET, C++, etc.

Outro detalhe importante é que podemos utilizar mais do que um controle de validação associado com o mesmo campo. Vamos imaginar um campo quantidade em um formulário de compra. Podemos utilizar um controle RequiredFieldValidator para fazer com que o usuário seja obrigado a digitar um valor neste campo e um controle CustomValidator para garantir que o valor digitado não seja maior do que o valor em estoque. A função de validação do controle CustomValidator pode buscar no banco de dados de estoque a quantidade de itens disponíveis e não aceitar que o usuário digite um valor maior do que o disponível em estoque.

A seguir apresentamos um exemplo da documentação do Framework .NET, onde foi utilizado um controle CustomValidator para fazer com que seja digitado um número par em um campo do formulário. Observe que, embora seja uma função simples de validação, não temos um controle específico para esta função. Por isso precisamos criar uma função de validação. No exemplo apresentado, temos a validação no servidor. O nome do procedimento que fará

a validação no servidor é definido no atributo OnServerValidate, do controle de validação, conforme indicado no comando a seguir:

```
OnServerValidate="ServerValidation"
```

onde ServerValidation é o nome do procedimento, na seção de código da página, que fará a validação para determinar se o número é par ou ímpar.

O procedimento ServerValidation recebe, como primeiro parâmetro, uma referência para o controle de validação e como segundo parâmetro um objeto que contém o valor do campo a ser validado, valor este que é acessível através da propriedade Value deste segundo argumento. Na listagem do exemplo, antes da função de validação, coloquei vários comentários que explicam detalhadamente a utilização dos parâmetros da função de validação.

Na Listagem 8.5 temos o código para o exemplo proposto.

### Listagem 8.5 – O controle de validação CustomValidator.

```
<%@ Page Language="C#" %>

<html>
<head>

 <script runat=server>

 void ValidateBtn_OnClick(object sender, EventArgs e)
 {
 if (Page.IsValid)
 {
 lblOutput.Text = "A página foi validada com sucesso.";
 }
 else
 {
 lblOutput.Text = "Página não validada!";
 }
 }

 // Procedimento que faz a validação do controle.
 // Este procedimento recebe dois argumentos.
 // O argumento source, do tipo object, é uma referência ao próprio controle de
 // validação.
 // O segundo argumento é do tipo ServerValidateEventArgs.
 // a sua propriedade Value contém o valor digitado no campo associado
 // ao controle de validação do tipo CustomValidator.

```

```
// Por isso, dentro da função ServerValidation, para acessar o valor digitado
// no campo a ser validado, utilizamos a seguinte referência:
// args.Value.

// Além disso, utilizamos o método Parse, da estrutura int, para converter
// a referência args.Value, no valor Int32 correspondente.

void ServerValidation (object source, ServerValidateEventArgs args)
{
 try
 {
 int i = int.Parse(args.Value);

 args.IsValid = ((i%2) == 0);
 }
 catch
 {
 args.IsValid = false;
 }
}

</script>

</head>

<body>

<form runat="server">

 <h3>Exemplo do controle CustomValidator!!</h3>

 <asp:Label id=lblOutput runat="server"
 Text="Digite um número par:"
 Font-Name="Verdana"
 Font-Size="10pt" />

 <p>

 <asp:TextBox id="Text1"
 runat="server" />


```

```
<asp:CustomValidator id="CustomValidator1"
 ControlToValidate="Text1"
 OnServerValidate="ServerValidation"
 Display="Static"
 ErrorMessage="Não é um número par!"
 ForeColor="green"
 Font-Name="verdana"
 Font-Size="10pt"
 runat="server"/>

<p>

<asp:Button id="Button1"
 Text="Validar"
 OnClick="ValidateBtn_OnClick"
 runat="server"/>

</form>

</body>
</html>
```

Digite o código da Listagem 8.5 e salve o mesmo em um arquivo chamado chap8ex5.aspx, na pasta chap8, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap8/chap8ex5.aspx>

No campo de digitação digite 2. Dê um clique no botão Enviar. Observe que nenhuma mensagem de erro é emitida, uma vez que dois é um número par.

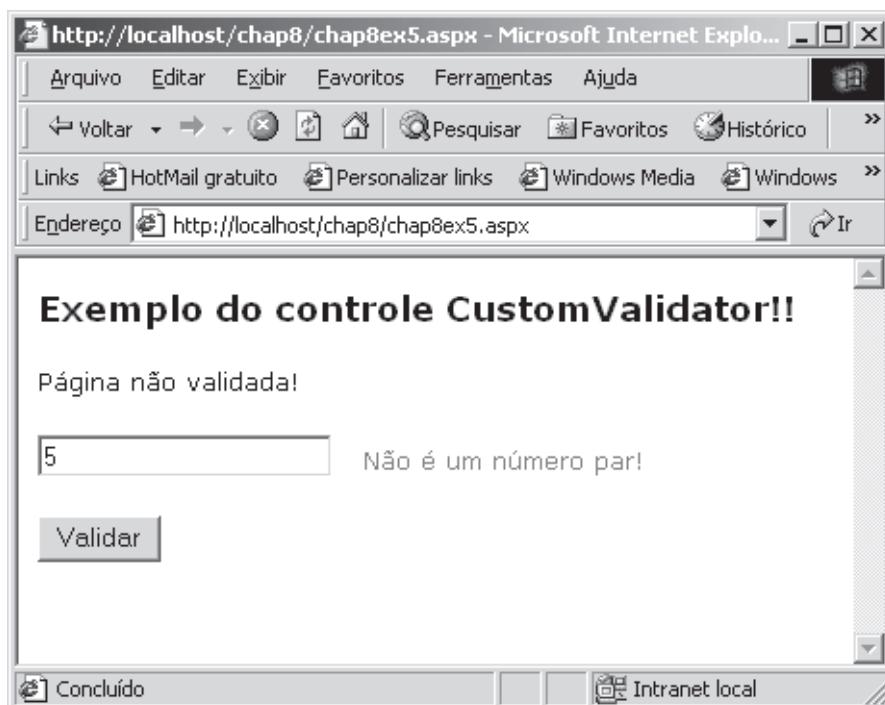
Agora digite 5 no campo de digitação. Dê um clique no botão Enviar. Você obterá uma mensagem de erro, conforme indicado na Figura 8.8.

Algumas observações sobre o exemplo.

Na função de validação ServerValidation, utilizamos a estrutura Try – Catch para fazer o tratamento de exceções. Dentro da função ServerValidation, utilizamos o seguinte comando para determinar se o número é par:

```
args.IsValid = ((i%2) == 0);
```

Neste caso, se o resto da divisão por 2 for zero, significa que o número é par e a propriedade IsValid é definida como True, significando que o valor digitado é par. Se o resto da divisão por dois não for igual a zero, a propriedade IsValid é definida como False, significando que o número digitado é ímpar. Lembre que o operador % retorna o resto da divisão entre dois números inteiros.



**Figura 8.8:** O controle de validação `CustomValidator` entrando em ação.

Também utilizamos dois controles do tipo Web Server Controls, para o campo de digitação e para o botão Validar. No Capítulo 8 aprenderemos a utilizar os Web Server Controls.

Utilizamos o evento Click do botão Validar, para determinar se a página foi validada ou não. Apenas para lembrar, a propriedade `IsValid` do objeto `Page` somente é `True` quando todos os controles da página passarem no teste de validação; basta que um único controle não seja validado, para que a propriedade `Page.IsValid` se torne `False`.

**NOTA:** Para maiores informações sobre o tratamento de exceções e sobre os operadores do C#, consulte os Capítulos 3, 4 e 5.

## O Controle RegularExpressionValidator

Este controle é utilizado para garantirmos que a entrada digitada está em um determinado formato. Por exemplo, podemos utilizar este controle para fazer com que o usuário digite o CPF no seguinte formato: 111.111.111-11. Ou que digite o CEP no seguinte formato: 11111-111.

A sintaxe para este controle é a seguinte:

```
<asp:RegularExpressionValidator
 id="identificação_no_código"
 runat="server"
 ControlToValidate="Controle a ser validado"
 ValidationExpression="Uma expressão do tipo Regular Expression."
 ErrorMessage="Mensagem de erro."
 ForeColor="Cor da fonte da mensagem de erro.">
```

```
BackColor="Cor de segundo plano da fonte da mensagem de erro."
>
</asp: RegularExpressionValidator>
```

A maior dificuldade de utilizarmos este controle é a definição correta do atributo ValidationExpression. Para você ter uma idéia da complexidade da criação de expressões regulares existe um livro somente sobre este assunto: Mastering Regular Expressions: Powerful Techniques for Perl and Other Tools. Editora: (O'Reilly Nutshell). Páginas: 368

Para uma lista de artigos sobre expressões regulares, consulte o seguinte endereço: <http://www.4guysfromrolla.com/webtech/RegularExpressions.shtml>

Vamos apresentar um exemplo, onde vamos utilizar o controle RegularExpressionValidator para verificar o formato do CPF digitado pelo usuário. O CPF tem o seguinte formato padrão:

**nnn.nnn.nnn-nn**

Onde n significa número.

Na Listagem 8.6 temos o código para o exemplo proposto.

### Listagem 8.6 – O controle de validação CustomValidator.

```
<html>
<head>
 <script language="C#" runat="server">

 void Enviar_Click(Object sender, EventArgs e)
 {
 // Código para gravar os dados recebidos no
 // banco de dados.
 }

 </script>

</head>
<body>
 <form runat=server>
 <h3>Exemplo do controle RegularExpressionValidator!!!</h3>
```

```

<input id="cpf" type="text" size=40 runat="server">

<asp:RegularExpressionValidator
id="valida_valor"
runat="server"
ControlToValidate="cpf"
ValidationExpression="[0-9]{3}\.[0-9]{3}\.[0-9]{3}\-[0-9]{2}"
ErrorMessage="Digite um CPF no formato 111.111.111-11"
ForeColor="Blue"
BackColor="Cyan"
/>

<input type=submit value="Enviar" OnServerClick="Enviar_Click" runat="server">
</form>

</body>
</html>

```

Digite o código da Listagem 8.6 e salve o mesmo em um arquivo chamado chap8ex6.aspx, na pasta chap8, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap8/chap8ex6.aspx>

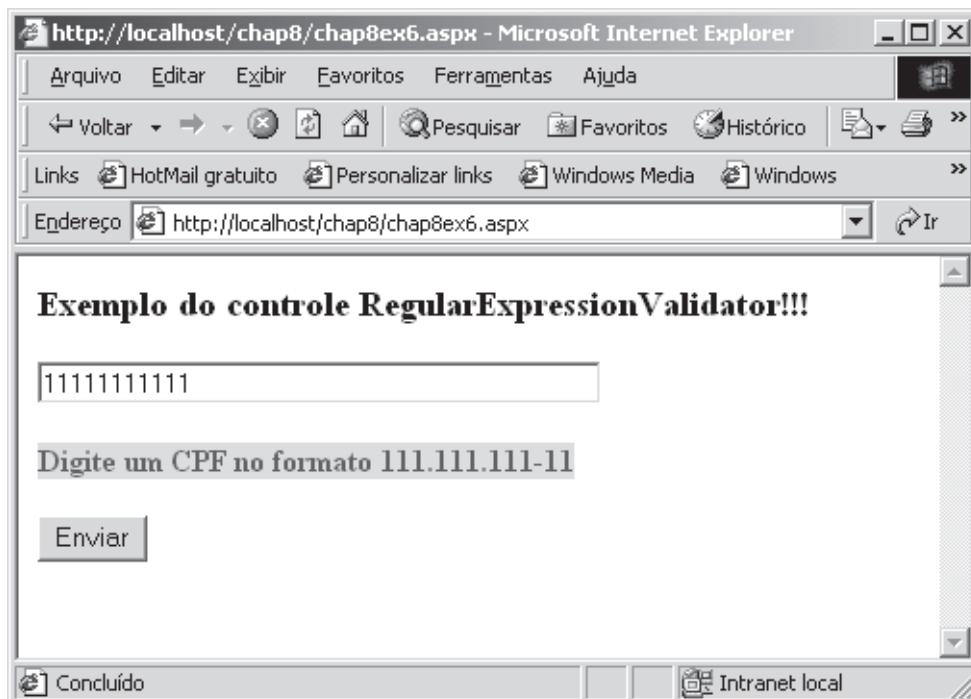
No campo de digitação digite 111.111.111-11, ou seja, um CPF no formato correto. Dê um clique no botão Enviar. Observe que nenhuma mensagem de erro é emitida, uma vez que o CPF está no formato definido pelo controle RegularExpressionValidator.

Agora digite 1111111111 no campo de digitação, ou seja, sem os pontos e o traço. Dê um clique no botão Enviar. Você obterá uma mensagem de erro, conforme indicado na Figura 8.9.

A definição do formato aceito para o campo CPF é feita através de uma expressão regular, no atributo ValidationExpression, conforme indicado a seguir:

```
ValidationExpression="[0-9]{3}\.[0-9]{3}\.[0-9]{3}\-[0-9]{2}"
```

O [0-9] indica que o caractere digitado deve ser um valor entre 0 e 9. O {3} entre chaves indica que os três primeiros caracteres devem estar entre 0 e 9. Todo caractere que deve ser digitado diretamente deve ser precedido de uma barra invertida. É o caso do ponto que deve ser digitado após os três primeiros números. Utilizando este mesmo raciocínio, definimos o restante da expressão.



**Figura 8.9:** O controle de validação RegularExpressionValidator entrando em ação.

Apenas para exemplificar, coloco como ficaria uma expressão para o CEP:

```
ValidationExpression="[0-9]{5}\-[0-9]{3}"
```

## Conclusão

Neste capítulo aprendemos a utilizar os controles de validação do ASP.NET. Aprendemos a utilizar os seguintes controles:

- ◆ RequiredFieldValidator Control
- ◆ CompareValidator Control
- ◆ RangeValidator Control
- ◆ CustomValidator Control
- ◆ RegularExpressionValidator Control

Também apresentamos diversos exemplos para ilustrar o funcionamento destes controles.

No próximo capítulo vamos aprender a utilizar uma série de Web Server Controls.

## Introdução

No Capítulo 7 estudamos os HTML Server Controls, os quais são, basicamente, os controles HTML para a criação de formulários, porém processados no servidor. Ao ser processado no servidor, cada controle é compilado como um objeto incorporado à página ASP.NET. Por ser um objeto temos acesso, via código de programação, a uma série de propriedades e métodos de cada controle. Com este novo modelo, o Framework .NET traz para o desenvolvimento Web um modelo baseado em eventos, muito parecido com o modelo de desenvolvimento baseado em eventos de aplicações Windows tradicionais.

Em seguida, no Capítulo 8 tratamos dos controles de validação – Validation Server Controls. Estes controles também são processados no servidor, porém não existem controles HTML correspondentes. Os controles de validação fazem parte de um conjunto de controles maior, conhecido como Web Server Controls. Salientamos que os Web Server Controls são definidos pela tag <asp: ...

No presente capítulo vamos continuar o nosso estudo sobre os controles de servidor. Vamos aprender a utilizar um conjunto de controles que nos permitem criar uma interface visual mais aprimorada, com maior riqueza de detalhes, dando um aspecto mais profissional a nossas páginas ASP.NET. Estes controles são chamados de Web Form Controls.

Os controles que estudaremos neste capítulo são, a exemplo dos controles de validação, Web Server Controls. Vamos aprender a utilizar controles para a criação de interface, onde o usuário pode digitar informações, e interagir com a página. Na prática estudaremos controles que podem ser utilizados no lugar dos HTML Server Controls, para a criação de páginas com uma interface mais “profissional”.

Todos os Web Form Controls são derivados de uma classe básica: WebControl. Esta classe pertence ao namespace System.Web.UI.WebControls. Iniciaremos o capítulo estudando as propriedades e métodos desta classe. Por serem derivados desta classe básica, os Web Form Controls herdam as propriedades e métodos da classe básica.

Em seguida passaremos ao estudo dos seguintes Web Form Controls:

- ◆ TextBox
- ◆ Label
- ◆ CheckBox
- ◆ RadioButton
- ◆ Button

- ◆ ListBox
- ◆ Table, TableCell e TableRow
- ◆ Panel
- ◆ Image
- ◆ HyperLink
- ◆ LinkButton
- ◆ ImageButton

Estes controles nos dão acesso a um grande número de propriedades e eventos, o que permite um controle bastante preciso sobre o layout e a funcionalidade de uma página ASP.NET, o que justifica a existência de mais um conjunto de controles, além do conjunto dos HTML Server Controls. À medida que formos estudando os diversos controles, iremos apresentar exemplos de utilização dos mesmos.

**NOTA:** Para acompanhar os exemplos deste capítulo é importante que você tenha estudado os Capítulos 3, 4 e 5, os quais tratam dos aspectos básicos da Linguagem C#.

## A Classe Básica – WebControl

A classe WebControl, do namespace System.Web.UI.WebControls, é a classe da qual são derivados os controles que iremos estudar neste capítulo. Esta classe define os métodos, eventos e propriedades comuns a todos os controles pertencentes ao namespace System.Web.UI.WebControls. Por serem derivados da classe WebControl, estes controles irão herdar suas propriedades e métodos públicos. Estudaremos as principais propriedades e métodos da classe WebControl.

### Principais Propriedades da Classe WebControl

Na Tabela 9.1 temos a descrição das principais propriedades da classe WebControl.

**Tabela 9.1** Propriedades da classe WebControl.

Propriedade	Descrição
AccessKey	É utilizada para definir ou retornar uma combinação de teclas (tecla de atalho) utilizada para colocar o foco no controle.
BackColor	É utilizada para definir ou retornar a cor de segundo plano, do controle.
BorderColor	É utilizada para definir ou retornar a cor das bordas do controle.
BorderStyle	É utilizada para definir ou retornar o estilo das bordas do controle.
BorderWidth	É utilizada para definir ou retornar a largura das bordas do controle.
ControlStyle	É utilizada para definir ou retornar o estilo (aspectos visuais) do controle, como por exemplo: com ou sem preenchimento, com sombra, etc.

Propriedade	Descrição
Enabled	Propriedade do tipo Boleana (True ou False), utilizada para definir ou retornar um valor que indica se o controle está habilitado: True = habilitado e False=desabilitado.
EnableViewState	Propriedade do tipo Boleana (True ou False), utilizada para definir ou retornar um valor que indica se o controle de estado está habilitado (True) ou desabilitado (False).
Font	Retorna informações sobre a fonte utilizada para o controle.
ForeColor	É utilizada para definir ou retornar informações sobre a cor do texto do controle.
Height	É utilizada para definir ou retornar a altura do controle.
Style	Faz referência a uma coleção de propriedades do tipo CSS – Cascading Style Sheets, associadas com o controle.
TabIndex	É utilizada para definir ou retornar informações sobre o índice de tabulação do controle. O índice de tabulação define a ordem de deslocamento do cursor, dentro de um formulário, quando pressionamos a tecla TAB. Por exemplo, se estamos em um campo com índice de tabulação 2 e pressionamos a tecla TAB, o cursor irá para o controle com índice de tabulação 3.
ToolTip	É utilizada para definir ou retornar um pequeno texto que será exibido quando o usuário apontar o mouse para o controle.
Visible	Propriedade boleana, utilizada para retornar ou definir um valor que indica se o controle deve ser exibido (True) ou não (False).
Width	É utilizada para definir ou retornar a largura do controle.

Estas propriedades estão disponíveis para todos os controles derivados da classe WebControl; em outras palavras, os controles que têm a classe WebControl como base irão herdar estas propriedades.

## Principais Métodos da Classe WebControl

Na Tabela 9.2 temos a descrição dos principais métodos da classe WebControl.

**Tabela 9.2** Métodos da classe WebControl.

Propriedade	Descrição
CopyBaseAttributes	Copia o valor das propriedades AccessKey, Enabled, TabIndex e Attributes para o controle atual, a partir de um controle especificado como parâmetro para o método. É semelhante ao que faz a ferramenta pincel nos aplicativos do Microsoft Office, quando copia uma série de características de formatação de um parágrafo para outro.

Propriedade	Descrição
.DataBind	Faz a associação de uma fonte de dados com o controle e com todos os seus controles filhos.
Equals	Determina se duas instâncias de um controle são iguais.
FindControl	Procura um determinado controle no Conteiner atual. Normalmente o Conteiner atual é a página ASP.NET onde está o controle.
HasControls	Retorna um valor Booleano indicando se o controle possui controles filhos (True) ou não (False).
ToString	Retorna uma representação do controle, no formato String.

## Principais Eventos da Classe WebControl

Na Tabela 9.3 temos a descrição dos principais eventos da classe WebControl.

**Tabela 9.3** Eventos da classe WebControl.

Evento	Descrição
DataBinding	Ocorre quando o controle é ligado (associado) a uma fonte de dados.
Disposed	Ocorre quando o controle é descarregado da memória, o qual é o último estágio no ciclo de vida do controle quando a página ASP.NET é carregada.
Init	Ocorre quando o controle é inicializado, o que é o primeiro estágio no seu ciclo de vida.
Load	Ocorre quando o objeto, que representa uma instância do controle, é carregado no objeto Page, que representa a página sendo carregada. Lembre que toda página ASP.NET ao ser carregada é compilada em um objeto que é uma instância da classe Page.

Agora vamos passar ao estudo de cada controle, individualmente.

## TextBox Web Server Control

Este controle é utilizado para criar uma caixa de texto para digitação de texto. Podemos criar uma caixa de texto de uma única linha ou de múltiplas linhas (o que criávamos utilizando um HTMLTextArea Server Control).

Por padrão, a propriedade TextMode é definida como SingleLine, o que significa que o controle tem uma única linha. Para definir este controle com múltiplas linhas, devemos definir a propriedade TextMode para MultiLine. Também podemos definir esta propriedade com o valor Password, para criar um texto para digitação de senhas, onde são exibidos apenas asteriscos enquanto o usuário digita a senha.

A largura da caixa de texto é definida pela propriedade Columns. Se a caixa de texto for de múltiplas linhas, a sua altura é definida pela propriedade Rows. Este controle é derivado da classe TextBox, do namespace System.Web.UI.WebControls.

A sintaxe para este controle é a seguinte:

```
<asp:TextBox id="value"
 AutoPostBack="True | False"
 Columns="Número de colunas"
 MaxLength="Tamanho máximo do texto a ser digitado"
 Rows="Número de linhas."
 Text="Texto para a caixa de digitação."
 TextMode="Single | Multiline | Password"
 Wrap="True | False"
 OnTextChanged="OnTextChangedMethod"
 runat="server"
/>
```

A propriedade AutoPostBack é do tipo Boleana, podendo assumir os valores True ou False. Se o valor desta propriedade estiver definido em True, um PostBack será gerado para o servidor, toda vez que o valor do controle for alterado. Por padrão, o valor desta propriedade é definido como False. Se for definida para True, toda vez que o valor do controle for alterado, o controle reenvia para o servidor (postback) o seu valor e o valor de todos os controles do formulário. Algo parecido com disparar um evento “Após alterar”, para um campo de um formulário criado com o Microsoft Access ou com o VB.

A propriedade Wrap define se deve ser feita uma quebra automática de linha, cada vez que o texto atinge o limite da caixa de texto.

OnTextChanged é um método que dispara o evento TextChanged, para o qual podemos definir um procedimento que será executado em resposta ao evento.

Vamos a um exemplo simples. Vamos criar um formulário onde temos três campos:

- ◆ Nome
- ◆ Senha
- ◆ Comentários

O usuário preenche os dados e clica no botão Enviar. Vamos criar um procedimento para o evento Click do botão Enviar. Este procedimento irá exibir, em um controle HtmlTextArea, os valores digitados pelo usuário, com exceção da senha.

Na Listagem 9.1 temos o código para o exemplo proposto.

**Listagem 9.1 – O controle TextBox – chap9ex1.aspx.**

```
<html>

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

 public void Enviar_Click(Object sender, EventArgs e)
 {
 Exibe.Value = Nome.Text + "\n" + "Seus comentários:"
 + "\n" + Comentários.Text;
 }
</script>

<body>

<form method=post runat="server">
 <H2> Exemplo do controle TextBox!!</H2>

 <table>
 <tr>
 <td><h3>Nome:</h3> </td>
 <td>
 <asp:TextBox runat=server
 id="Nome"
 Text=""
 Font_Face="Arial"
 Font_Size="3"
 BackColor="Cyan"
 ForeColor="Blue"
 BorderColor="Red"
 TextMode="SingleLine"
 Columns="40"
 />
 </td>
 </tr>
 <tr>

```

```
<td><h3>Senha:</h3> </td>

<td>

<asp:TextBox runat=server

 id="Senha"

 Text=""

 Font_Face="Arial"

 Font_Size="3"

 BackColor="Cyan"

 ForeColor="Blue"

 BorderColor="Green"

 TextMode="Password"

 Columns="40"

/>

</td>

</tr>

<tr>

<td><h3>Comentários:</h3> </td>

<td>

<asp:TextBox runat=server

 id="Comentários"

 Text=""

 Font_Face="Arial"

 Font_Size="2"

 BackColor="Silver"

 ForeColor="Blue"

 BorderColor="Green"

 TextMode="MultiLine"

 Rows="10"

 Columns="30"

/>

</td>

</tr>

<tr>

<td><h3>Você digitou:</h3></td>
```

```
<td><textarea id="Exibe" cols="40" rows="5" runat="server" /></td>
</tr>

<tr>
 <td>Clique no botão--></td>
 <td>
 <input type=submit value="Enviar" OnServerClick="Enviar_Click"
 runat="server">
 </td>
</tr>

</table>

</form>

</body>
</html>
```

Digite o código da Listagem 9.1 e salve o mesmo em um arquivo chamado chap9ex1.aspx, na pasta chap9, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap9/chap9ex1.aspx>

No campo Nome digite: José abc da silva.

No campo Senha digite: 123456.

No campo de comentários digite o seguinte texto:

“Estudo dos Web Form Controls do ASP.NET.

Controles avançados e com melhor interface visual, com acesso a uma variedade de métodos, propriedades e eventos.”

Dê um clique no botão Enviar. Você obterá o resultado indicado na Figura 9.1.

Observe que o controle que exibe os valores digitados pelo usuário não consegue exibir todo o conteúdo digitado; por isso uma barra de rolagem vertical é disponibilizada.

No evento Click do botão Enviar, definimos a propriedade Value do controle Exibe como sendo a concatenação dos valores digitados nos campos Nome e Comentários. Para concatenar strings, apenas para lembrar, utilizamos o operador +. O “\n” define uma quebra de linha. Também é importante salientar que, para obter o valor digitado nos controles, utilizamos a propriedade Text. Nos controles do tipo HtmlServer, utilizávamos a propriedade Value, ao invés da

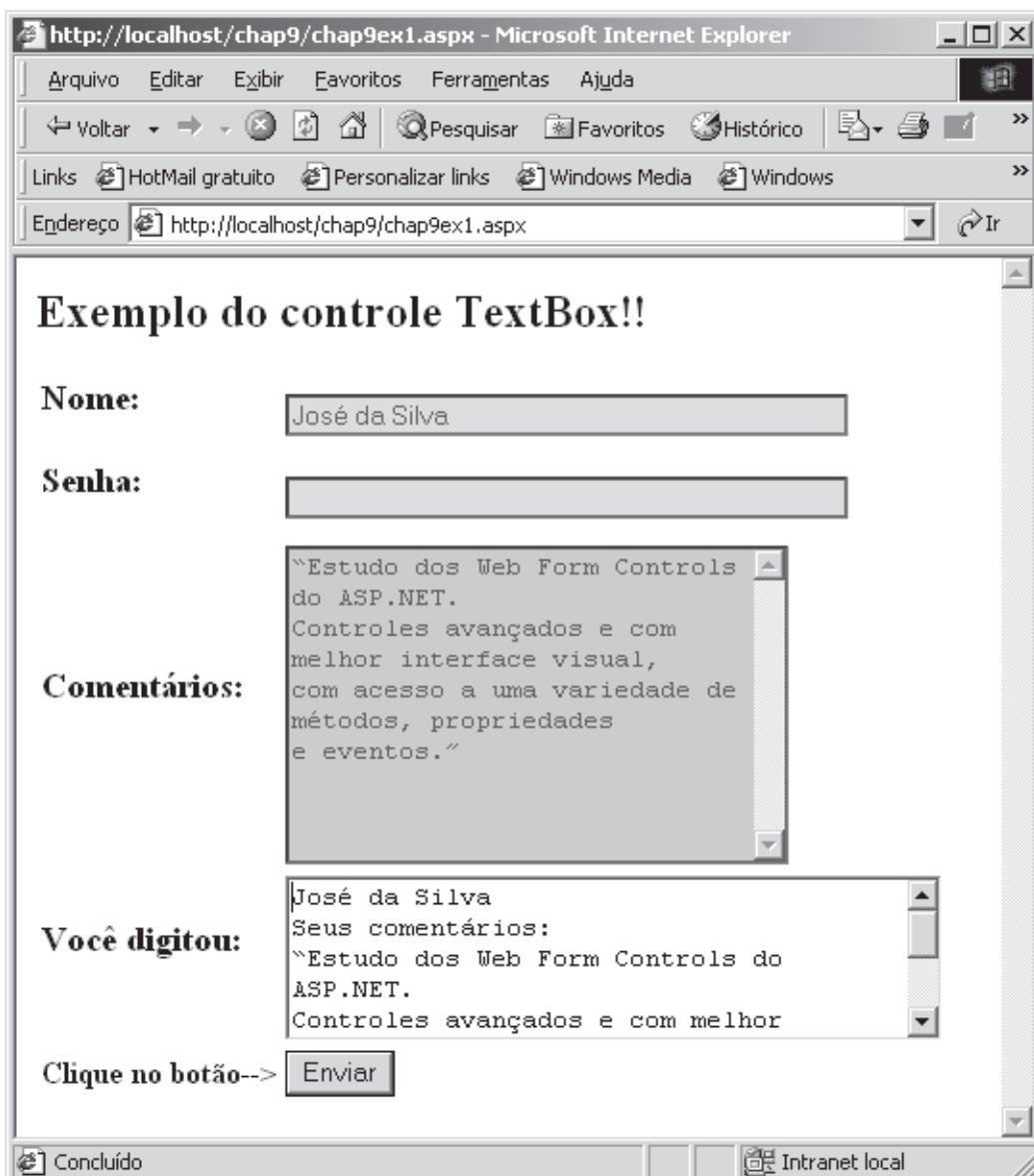


Figura 9.1: Utilizando o controle TextBox com as opções SingleLine, Password e MultiLine.

propriedade Text. Aliás estamos fazendo isto neste exemplo, pois o controle que exibe os valores digitados pelo usuário é um controle do tipo HtmlServer. Para definir o que o controle irá exibir, estamos definindo a sua propriedade Value, conforme indicado no trecho de código a seguir:

```
public void Enviar_Click(Object sender, EventArgs e)
{
 Exibe.Value = Nome.Text + "\n" + "Seus comentários:"
 + "\n" + Comentários.Text;
}
```

## Label Web Server Control

Este controle é utilizado para exibir texto em um formulário. Podemos acessar suas propriedades e métodos, inclusive alterar o texto do controle, através de programação.

A sintaxe para este controle é a seguinte:

```
<asp:Label
id="identificação_no_código"
Text="Texto a ser exibido
runat="server"/>
```

ou

```
<asp:Label
id="Identificação_no_código"
runat="server">
 Text a ser exibido
</asp:Label>
```

**IMPORTANTE:** Você deve estar lembrado de que, nas versões anteriores do ASP, não podíamos “quebrar” um comando em várias linhas, sem colocar um caractere especial, no final de cada linha. Com ASP.NET, não temos mais esta limitação; observe que podemos dividir o comando em diversas linhas, sem maiores problemas.

Para definir o texto a ser exibido em um controle Label, utilizamos a propriedade Text do controle.

A seguir um exemplo que adiciona um controle do tipo Label, com o texto inicial “Este é o texto original do controle”. Ao clicarmos no botão Enviar, o evento Click deste botão altera o texto do Label para “Este é o texto modificado”.

Na Listagem 9.2 temos o código para o exemplo proposto.

### Listagem 9.2 – O controle Label – chap9ex2.aspx.

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

 public void Enviar_Click(Object sender, EventArgs e)
 {
 Rotulo1.Text="Este é o texto modificado";
 }

</script>

<body>

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

```
<H2> Exemplo do controle Label!!</H2>
```

```


 <asp:Label
 id="Rotulo1"
 Text="Este é o texto original do controle"
 BackColor="Black"
 ForeColor="White"
 runat="server"
 />

<table>
 <tr>
 <td>Clique no botão--></td>
 <td>
 <input type=submit value="Enviar" OnServerClick="Enviar_Click"
runat="server">
 </td>
 </tr>
</table>

</form>

</body>
</html>

```

Digite o código da Listagem 9.2 e salve o mesmo em um arquivo chamado chap9ex2.aspx, na pasta chap9, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap9/chap9ex2.aspx>

Ao carregar a página será exibido o texto “Este é o texto original do controle”. Observe que o texto está com cor de fundo preta e cor de fonte branca, conforme definimos nas propriedades do controle Label. Dê um clique no botão Enviar. O texto do controle do tipo Label é alterado para “Este é o texto modificado”, conforme indicado na Figura 9.2.

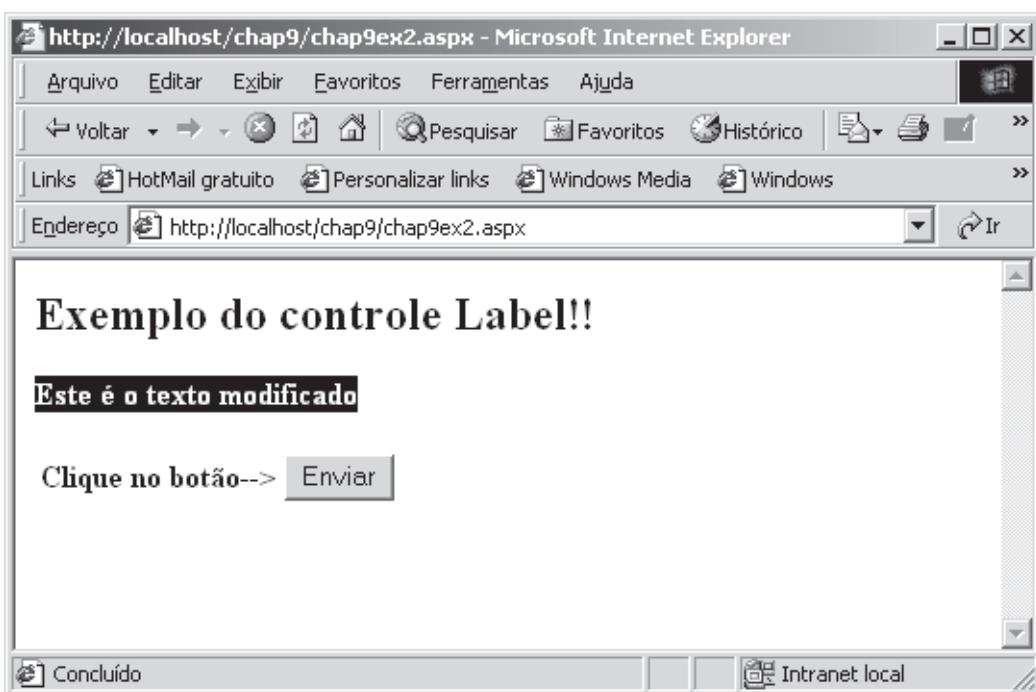


Figura 9.2: Utilizando o controle Label – chap9ex2.aspx.

## CheckBox Web Server Control

Este controle é utilizado para a criação de um controle do tipo caixa de seleção. Este tipo de controle pode assumir dois estados: marcado ou não marcado. Ao clicar no controle, o seu estado é alterado, isto é, se estiver marcado, ele será desmarcado; se estiver desmarcado, ele será marcado.

O evento CheckedChanged é disparado quando o formulário é enviado para o servidor e o status (marcado/desmarcado) do controle foi alterado em relação ao último envio do formulário.

A sintaxe para este controle é a seguinte:

```
<asp:CheckBox
id="identificação_no_código"
AutoPostBack="True | False"
Text="Rótulo de texto associado ao controle."
TextAlign="Right | Left"
Checked="True | False"
OnCheckedChanged="Método que será executado em resposta ao evento
OnCheckedChanged"
runat="server"
/>
```

**NOTA:** Para criar múltiplos controles do tipo CheckBox, de uma só vez, ligados a uma fonte de dados, devemos utilizar o controle CheckBoxList. Aprenderemos a utilizar este controle nos Capítulos 10 e 11.

Vamos apresentar um exemplo de utilização do controle CheckBox. O nosso exemplo é constituído de um formulário no qual temos três controles do tipo CheckBox, onde o usuário pode selecionar um ou mais controles. Ao clicar no botão Enviar, o evento Click deste botão informa, em um controle do tipo Label, quais as opções selecionadas.

Na Listagem 9.3 temos o código para o exemplo proposto.

**Listagem 9.3 – O controle CheckBox – chap9ex3.aspx.**

```
<html>

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

 public void Enviar_Click(Object sender, EventArgs e)
 {

 //Declaração das variáveis auxiliares

 String Aux="Opções selecionadas: " ;

 // Começo a montar uma string que será atribuída à
 // propriedade Value do controle Dados.

 if (Negócios.Checked==true)
 {
 Aux= Aux +"Negócios ";
 }

 if (Direito.Checked==true)
 {
 Aux= Aux +"Direito ";
 }

 if (Economia.Checked==true)
 {
 Aux= Aux +"Economia ";
 }

 Exibe.Text = Aux;
 }
</script>
```

```
}

</script>

<body>

<form method=post runat="server">
 <table>
 <tr>
 <td colspan="2"><H2>Selecione as áreas de interesse:</H2></td>
 </tr>

 <tr>
 <td>Opção 1: </td>
 <td>
 <asp:CheckBox
 id="Negócios" runat="server"
 Text="Negócios"
 AutoPostBack="True"
 />
 </td>
 </tr>

 <tr>
 <td>Opção 2: </td>
 <td>
 <asp:CheckBox
 id="Direito"
 runat="server"
 Text="Direito"
 AutoPostBack="True"
 />
 </td>
 </tr>
 </table>
</form>
```

```
<tr>
 <td>Opção 3: </td>
 <td>
 <asp:CheckBox
 id="Economia"
 runat="server"
 Text="Economia"
 AutoPostBack="True"
 />
 </td>
</tr>

<tr>
 <td>Clique no botão -></td>
 <td>
 <input type="submit" value="Enviar"
 OnServerClick="Enviar_Click"
 runat="server">
 </td>
</tr>

<tr>
 <td>Dados do cliente: </td>
 <td>

 <asp:Label
 id="Exibe"
 Text=""
 BackColor="Black"
 ForeColor="White"
 runat="server"
 />

 </td>
</tr>

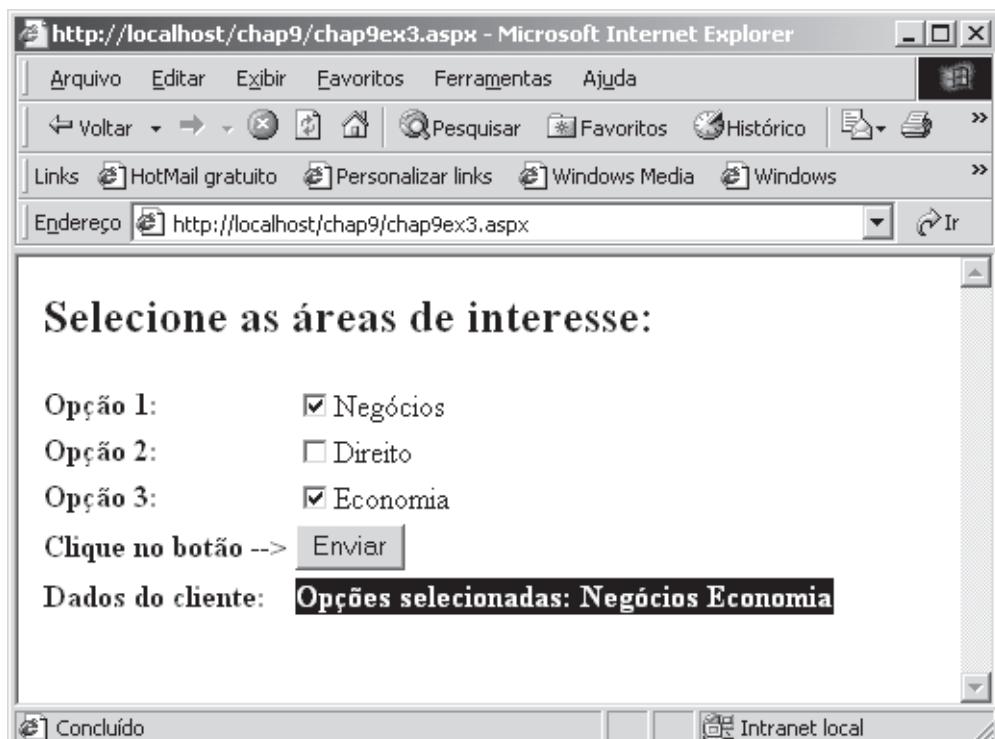
</table>

</form>
</body>
</html>
```

Digite o código da Listagem 9.3 e salve o mesmo em um arquivo chamado chap9ex3.aspx, na pasta chap9, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap9/chap9ex3.aspx>

Ao carregar a página será exibido um formulário com três controles do tipo CheckBox, o botão Enviar e um controle do tipo Label. Selecione as opções Negócios e Economia. Dê um clique no botão Enviar. As opções selecionadas serão informadas em um controle do tipo Label, conforme indicado na Figura 9.3.



**Figura 9.3: Utilizando o controle CheckBox – chap9ex3.aspx.**

Neste exemplo estamos utilizando a propriedade Checked para verificar se o controle foi ou não selecionado. No final do procedimento Enviar\_Click, definimos o texto do controle do tipo Label, como sendo igual ao conteúdo da variável Aux. Observe que para fazer esta definição utilizamos a propriedade Text, conforme indicado no comando a seguir:

```
Exibe.Text = Aux;
```

## RadioButton Web Server Control

Este controle é utilizado para a criação de controles do tipo “Botão de rádio”. Estes controles são criados em grupos, sendo que somente um dos controles do grupo pode ser selecionado. Para criar um grupo de controles do tipo RadioButton, devemos definir o mesmo valor para a propriedade GroupName, de todos os controles que fazem parte do grupo.

**NOTA:** Para criar múltiplos controles do tipo RadioButton, de uma só vez, ligados a uma fonte de dados, devemos utilizar o controle RadioButtonList. Aprenderemos a utilizar este controle nos Capítulos 10 e 11.

O Controle RadioButton Web Server Control é derivado da classe base RadioButton, do namespace System.Web.UI.WebControls.

A sintaxe para este controle é a seguinte:

```
<asp:RadioButton
 id="Identificação_no_código"
 AutoPostBack="True|False"
 Checked="True|False"
 GroupName="Nome do Grupo."
 Text="Rótulo de texto que identifica o controle"
 TextAlign="Right|Left"
 OnCheckedChanged="Método que será executado em resposta ao evento
 OnCheckedChanged"
 runat="server"
/>

Considere o trecho de código a seguir:
<asp:RadioButton
 id="Cartão"
 Text="Cartão de Crédito:"
 Checked="True"
 GroupName="GrupoPagamento"
 runat="server" />

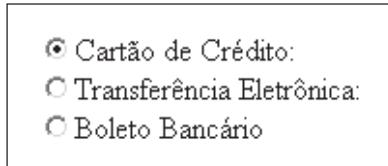
<asp:RadioButton
 id="Transferência"
 Text="Transferência Eletrônica:"
 GroupName="GrupoPagamento"
 runat="server"/>

<asp:RadioButton id="Boleto"
 Text="Boleto Bancário"
 GroupName="GrupoPagamento"
 runat="server"/>
```

Neste trecho criamos um grupo chamado GrupoPagamento, o qual possui três controles do tipo RadioButton:

- ◆ Cartão
- ◆ Transferência
- ◆ Boleto

Ao carregarmos a página, o controle Cartão já vem selecionado, pois definimos a sua propriedade Checked="True". Este trecho de código gera a saída indicada na Figura 9.4.



**Figura 9.4: Grupo de Controles do tipo RadioButton – GrupoPagamento.**

Para verificar qual opção foi selecionada, podemos utilizar a propriedade Checked de cada controle, conforme indicado no seguinte trecho de código:

```
if (Cartão.Checked)
{
 Label1.Text = "Você selecionou: " + Cartão.Text;
}

else if (Transferência.Checked)
{
 Label1.Text = " Você selecionou:" + Transferência.Text;
}

else if (Boleto.Checked)
{
 Label1.Text = " Você selecionou:" + Boleto.Text;
}
```

Neste exemplo estamos definindo o texto a ser exibido em um controle do tipo Label (Label1), com base na opção selecionada. A propriedade Text do controle RadioButton (Cartão.Text, Transferência.Text e Boleto.Text) retorna o valor definido na propriedade Text do controle. Por exemplo, para o controle Cartão, Cartão.Text="Cartão de Crédito".

## Button Web Server Control

Este controle é utilizado para criar um botão de comando em um formulário. Podemos criar dois tipos diferentes de botões, com este controle:

- ◆ Botão Enviar (Submit)
- ◆ Botão de Comando (Command)

Um botão do tipo Enviar não tem um nome de comando (definido pela propriedade CommandName) associado com o botão. Este botão simplesmente envia a página para processamento no servidor. Por padrão, um controle Button é do tipo Enviar. Podemos definir um procedimento que executa em resposta ao evento Click do botão, como já fizemos em diversos exemplos deste livro.

**NOTA:** Para uma referência completa sobre todas as propriedades do controle RadioButton, consulte a documentação do Framework .NET. Mais especificamente, consulte a classe RadioButton do namespace **System.Web.UI.WebControls**.

Um botão do tipo Command possui um nome de comando associado com o botão, nome este que é definido na propriedade CommandName do controle. Isso permite que sejam criados múltiplos controles do tipo Button, em um formulário (Web Form) de uma página ASP.NET. Através de código podemos determinar qual o botão que foi “clicado” e, para cada botão, podemos definir o código que será executado em resposta ao evento Click do botão.

A sintaxe para este controle é a seguinte:

```
<asp:Button
 id="Identificação_no_código"
 Text="Texto exibido no botão"
 CommandName="Nome de comando. Define o controle como sendo
 do tipo Command"
 CommandArgument="Define ou retorna os argumentos passados
 para o método Command."
 OnClick="Método que é executado em resposta ao evento
 Click do botão."
 runat="server"
/>
```

**NOTA: O Controle Button**  
 Web Server Control é derivado da classe base Button, do namespace System.Web.UI.WebControls. Para uma referência completa a todas as propriedades e métodos deste controle, consulte a documentação do Framework .NET.

No trecho de código a seguir, temos o exemplo da criação de um controle Button, do tipo Submit, em uma página .aspx:

```
<asp:Button
 id="Enviar"
 Text="Enviar"
 OnClick="Enviar_Click"
 runat="server"
/>
```

No trecho de código a seguir, temos o exemplo da criação de um controle Button, do tipo Command, em uma página .aspx:

```
<asp:Button
 id="SortAscendingButton"
 Text="Sort Ascending"
 CommandName="Sort"
 CommandArgument="Ascending"
 OnClick="CommandBtn_Click"
 runat="server"
/>
```

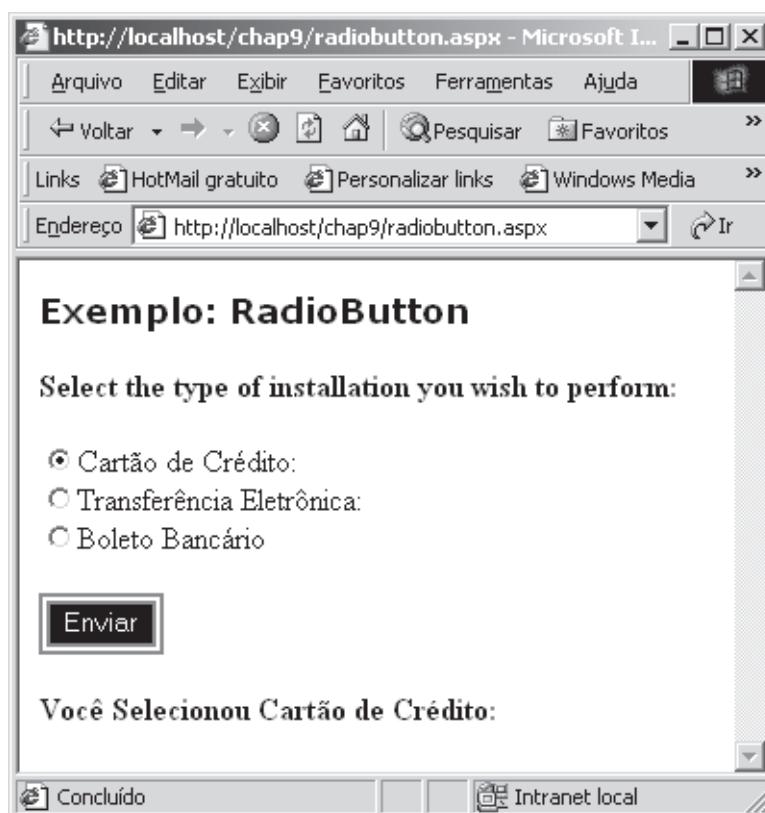
Podemos alterar as aparências de qualquer controle, utilizando a sua propriedade Style. Através desta propriedade, podemos controlar os diversos aspectos visuais de um controle. A seguir temos um exemplo de utilização desta propriedade:

```
<asp:Button
id="Enviar"
Text="Enviar"
style="color:White;
background-color:Black;
border-color:Red;
border-width:6px;
border-style:Double;"
onClick="Enviar_Click"
runat="server"
/>
```

Observe que os aspectos visuais são definidos aos pares: aspecto:valor;. Por exemplo, a definição da cor das bordas do botão é feita da seguinte maneira:

```
border-color:Red;
```

Este controle produz o resultado indicado na Figura 9.5.



**Figura 9.5: Usando a propriedade Style.**

## ListBox Web Server Control

Este controle é utilizado para criar uma lista de seleção, onde podemos selecionar um ou mais elementos. Para definir o número de elementos que são exibidos, simultaneamente, devemos definir a propriedade Rows. Por padrão, é exibida uma lista de uma única linha. Para habilitar a seleção de múltiplos elementos da lista, devemos definir a propriedade SelectionMode="Multiple".

Para adicionar elementos à lista de elementos do controle ListBox, utilizamos o controle ListItem, conforme veremos no exemplo mais adiante.

A sintaxe para este controle é a seguinte:

```
<asp:ListBox
 id="Identificação_no_código"
 DataSource="<% Expressão para conexão com uma fonte de dados %>"
 DataTextField="Nome do campo"
 DataValueField="Valor do campo"
 AutoPostBack="True | False"
 Rows="Número de linhas do controle"
 SelectionMode="Single | Multiple"
 OnSelectedIndexChanged="Método que será executado em
 resposta ao evento."
 runat="server">

 <asp:ListItem value="value" selected="True | False">
 Texto do item.
 </asp:ListItem>

</asp:ListBox>
```

Vamos apresentar um exemplo de utilização do controle ListBox. O nosso exemplo é constituído de um formulário no qual colocamos um controle ListBox, com seis elementos. O controle permite seleções múltiplas e seu tamanho é definido em 4 linhas. Ao clicar no botão Enviar, o evento Click deste botão informa qual foi o primeiro elemento a ser selecionado.

Na Listagem 9.4 temos o código para o exemplo proposto.

### Listagem 9.4 – O controle ListBox – chap9ex4.aspx.

```
<html>
<head>
 <script language="C#" runat="server">
```

**NOTA:** As definições dos elementos visuais fazem parte de uma especificação conhecida como CSS – Cascading Style Sheets. O estudo da definição CSS está fora do escopo deste livro. Para maiores informações sobre CSS, consulte os seguintes endereços:  
[www.w3.org](http://www.w3.org) –  
[www.wdvl.com](http://www.wdvl.com) –  
[www.internet.com](http://www.internet.com) e  
[www.developer.com](http://www.developer.com)

**NOTA:** O Controle ListBox Web Server Control é derivado da classe base ListBox, do namespace System.Web.UI.WebControls. Para uma referência completa a todas as propriedades e métodos deste controle, consulte a documentação do Framework .NET.

```
void Enviar_Click(Object sender, EventArgs e)
{
 if (ListBox1.SelectedIndex > -1)
 Label1.Text = "O primeiro item que você selecionou foi: " +
 ListBox1.SelectedItem.Text;
}

</script>
</head>
<body>

<form runat=server>
 <h3>Exemplo do controle ListBox!!</h3>

 <asp:ListBox id="ListBox1"
 Rows="4"
 SelectionMode="Multiple"
 Width="100px"
 runat="server"
 style="color:White;
 background-color:Black;">
 <asp:ListItem>Item 1</asp:ListItem>
 <asp:ListItem>Item 2</asp:ListItem>
 <asp:ListItem>Item 3</asp:ListItem>
 <asp:ListItem>Item 4</asp:ListItem>
 <asp:ListItem>Item 5</asp:ListItem>
 <asp:ListItem>Item 6</asp:ListItem>
 </asp:ListBox>
 <asp:button id="Enviar"
 Text="Enviar"
 OnClick="Enviar_Click"
 runat="server" />
 <p>

 <asp:Label id="Label1"
 Font-Name="Verdana"
 Font-Size="10pt"

```

```

 runat="server" />

</form>

</body>
</html>

```

Digite o código da Listagem 9.4 e salve o mesmo em um arquivo chamado chap9ex4.aspx, na pasta chap9, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço:

<http://localhost/chap9/chap9ex4.aspx>

Para selecionar múltiplas opções na lista de opções, mantenha a tecla Ctrl pressionada e vá clicando nas opções a serem selecionadas. Selecione as opções 1, 2 e 3 e dê um clique no botão Enviar, conforme indicado na Figura 9.6.

**Figura 9.6: Utilizando o controle ListBox – chap9ex4.aspx.**

No evento Click, do botão Enviar, utilizamos o seguinte código, para informar o primeiro item selecionado:

```

void Enviar_Click(Object sender, EventArgs e)
{
 if (ListBox1.SelectedIndex > -1)
 Label1.Text = " O primeiro item que você selecionou foi: " +
 ListBox1.SelectedItem.Text;
}

```

O comando ListBox1.SelectedItem.Text é que retorna o texto do primeiro item selecionado.

## Table, TableCell e TableRow Web Server Controls

Este controles são utilizados para a criação de tabelas em uma página .aspx. São semelhantes aos Html Server Controls: HtmlTable, HtmlTableRow e HtmlTableCell, porém oferecem mais opções de propriedades, métodos e eventos.

Podemos construir uma tabela de tamanho fixo (número de linhas e colunas), ou podemos construir a tabela dinamicamente, através de código de programação. Porém as modificações nas linhas e colunas da tabela serão perdidas quando a página for recarregada. Se o número de modificações for grande, devemos utilizar os controles DataList e DataGrid, ao invés do controle Table. Aprenderemos a utilizar os controles DataList e DataGrid nos Capítulos 10 e 11.

**NOTA:** Nos Capítulos 10 e 11 veremos exemplos de utilização das propriedades DataSource, DataTextField e DataValueField, onde os items do controle ListBox serão obtidos a partir de uma fonte de dados.

A sintaxe para este controle é a seguinte:

```
<asp:Table
 id="Identificação_no_código"
 BackImageUrl="Endereço da imagem de segundo plano."
 CellSpacing="Distância, em pixels, entre as bordas
 das células e o conteúdo."
 CellPadding=" Distância, em pixels, entre as
 células da tabela."
 GridLines="None|Horizontal|Vertical|Both"
 HorizontalAlign="Center|Justify|Left|NotSet|Right"
 runat="server"
>

<asp:TableRow>

 <asp:TableCell>
 Texto da Célula.
 </asp:TableCell>

</asp:TableRow>

</asp:Table>
```

**NOTA: O Controle Table Web Server Control é derivado da classe base Table, do namespace System.Web.UI.WebControls. Para uma referência completa a todas as propriedades desta classe, consulte a documentação do Framework .NET.**

Para criar uma nova linha da tabela, utilizamos o controle TableRow. Uma tabela é formada por uma coleção de linhas, onde cada linha é definida por uma coleção de células – Cells Collection. Podemos definir o conjunto de linhas da tabela estaticamente, através da utilização de uma série de controles TableRow, ou dinamicamente, através de código de programação.

Sintaxe para o controle TableRow:

```
<asp:TableRow
 id="Identificação_co_código"
 HorizontalAlign="Center|Justify|Left|NotSet|Right"
 VerticalAlign="Bottom|Middle|NotSet|Top"
 runat="server"
>

 <asp:TableCell>
 Texto da célula.
 </asp:TableCell>

</asp:TableRow>.
```

A propriedade HorizontalAlign define o alinhamento horizontal do conteúdo da célula, em relação à célula; a propriedade VerticalAlign define o alinhamento vertical do conteúdo da célula, em relação à célula.

Para adicionarmos células (colunas) a uma linha da tabela, utilizamos o controle TableCell. Uma linha é formada por uma coleção de células (Cells).

Sintaxe para o controle TableCell:

```
<asp:TableCell
 id="Identificação_no_código"
 ColumnSpan="Número de colunas a ser mescladas."
 RowSpan="Número de linhas a ser mescladas."
 HorizontalAlign="Center|Justify|Left|NotSet|Right"
 VerticalAlign="Bottom|Middle|NotSet|Top"
 Wrap="True|False"
 runat="server">
 Texto da Célula.
</asp:TableCell>
```

A propriedade Wrap define se deve haver o retorno automático de texto dentro da célula (Wrap="True") ou não (Wrap="False").

Vamos apresentar um exemplo, no qual uma tabela é criada dinamicamente, através de código executado no evento Load da página (Page\_Load). O código do evento Load adiciona três linhas, com três células em cada linha. A tabela é criada na seção de apresentação da página, porém sem nenhuma linha ou coluna, elementos estes que são adicionados pelo código do evento Load.

Na Listagem 9.5 temos o código para o exemplo proposto.

#### Listagem 9.5 – Os controles Table, TableRow e TableCell – chap9ex5.aspx.

```
<%@ Page Language="C#" %>
<html>
<head>

<script runat="server">
 void Page_Load(Object sender, EventArgs e)
 {
 // Declaro duas variáveis.
```

**NOTA: O Controle TableRow Web Server Control é derivado da classe base TableRow, do namespace System.Web.UI.WebControls. Para uma referência completa a todas as propriedades desta classe, consulte a documentação do Framework .NET.**

**NOTA: O Controle TableCell Web Server Control é derivado da classe base TableCell, do namespace System.Web.UI.WebControls. Para uma referência completa a todas as propriedades desta classe, consulte a documentação do Framework .NET.**

```
// numrows contém o número de linhas.
// numcells contém o número de colunas.

int numrows = 3;
int numcells = 2;

// O laço externo cria uma nova linha a cada passagem do laço.
// O laço interno vai adicionando células a linha criada pela
// passada do laço externo.
for (int j=0; j<numrows; j++)
{

 // Para cada passagem do laço externo, adiciono uma nova linha.

 TableRow r = new TableRow();

 for (int i=0; i<numcells; i++)
 {

 // Para cada passagem do laço interno, adiciono uma nova célula.

 TableCell c = new TableCell();

 c.Controls.Add(new LiteralControl("linha " + j.ToString() + ", coluna "
+ i.ToString()));
 r.Cells.Add(c);
 }
 Table1.Rows.Add(r);
}
}
</script>

</head>
<body>

<form runat=server>

<h3>Exemplo de tabela dinâmica!!</h3>
```

```

<asp:Table
 id="Table1"
 GridLines="Both"
 HorizontalAlign="Center"
 style="color:White;
 background-color:Black;
 border-color:Red;
 border-width:6px;
 border-style:Double;">

 Font-Name="Verdana"
 Font-Size="8pt"
 CellPadding="15"
 CellSpacing="0"
 runat="server"
/>

</form>

</body>
</html>

```

Digite o código da Listagem 9.5 e salve o mesmo em um arquivo chamado chap9ex5.aspx, na pasta chap9, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap9/chap9ex5.aspx>

Você vai obter o resultado indicado na Figura 9.7.

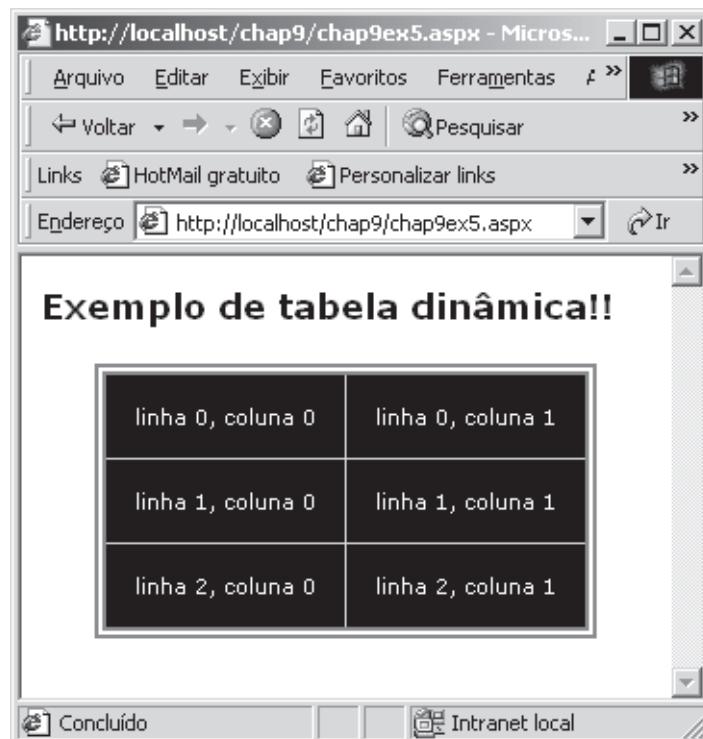
Alguns comentários sobre o exemplo:

No evento load utilizamos dois laços for. O laço externo adiciona uma nova linha à tabela, a cada passada do laço; já o laço interno adiciona uma nova célula a cada passada do laço. Para definir o conteúdo da célula, utilizamos o seguinte comando:

```
c.Controls.Add(new LiteralControl("linha " + j.ToString() + ", coluna " +
i.ToString()));
```

Esta técnica é diferente da utilizada com os HtmlServer Controls para a criação de tabelas, quando utilizamos a propriedade InnerHtml do controle HtmlTableCell. Não podemos utilizar esta técnica com o controle TableCell, pois ele não possui uma propriedade InnerHtml.

Para resolver este problema, utilizamos um controle do tipo LiteralControl. Quando esta página é compilada, o controle LiteralControl retorna exatamente o que foi passado como parâmetro. No nosso exemplo, ao carregar a página, tudo o



**Figura 9.7: Utilizando os controles Table, TableRow e TableCell – chap9ex5.aspx.**

que foi passado como parâmetro para o controle é executado e o resultado da execução é inserido no local do controle. Vamos acompanhar o que acontece para a primeira passada do laço externo e a primeira passada do laço interno, em outras palavras:

```
j=0
```

```
i=0
```

quando a página é processada, o parâmetro passado para o controle LiteralControl fica da seguinte maneira:

```
linha 0, coluna 0
```

- ◆ j.ToString é substituído pela string correspondente ao valor de j – 0.
- ◆ i.ToString é substituído pela string correspondente ao valor de i – 0.

Com isso, o nosso comando fica:

```
c.Controls.Add(new LiteralControl("linha 0, coluna 0));
```

Ao processar todo este comando, é retornado apenas o argumento entre aspas:

```
linha 0, coluna 0
```

que é exatamente o conteúdo que será exibido na célula.

Embora pareça um pouco complicado, à primeira vista, este exemplo demonstra o poder e a flexibilidade que temos à disposição com o uso dos Web Server Controls e da biblioteca de classes do Framework .NET. Qualquer elemento da página é tratado como um objeto. Através de programação temos um controle total sobre os métodos, propriedades e eventos destes objetos.

Para finalizar, gostaria de lembrar que mais uma vez utilizamos a propriedade style, para definir alguns aspectos visuais do controle Table, conforme indicado no fragmento a seguir:

```
<asp:Table id="Table1"
 GridLines="Both"
 HorizontalAlign="Center"
 style="color:White;
 background-color:Black;
 border-color:Red;
 border-width:6px;
 border-style:Double;"
 Font-Name="Verdana"
 Font-Size="8pt"
 CellPadding="15"
 CellSpacing="0"
 runat="server"
/>
```

## Panel Web Server Control

Este controle funciona como um Container para outros controles; ao ser processado é gerada uma cláusula HTML - <DIV>. Utilizamos este controle quando precisamos adicionar controles à pagina, dinamicamente através de programação, ou quando precisamos ocultar e exibir um grupo de controles através do código de programação. Por exemplo, vamos supor que você queira que um conjunto de controles somente seja visível quando o valor de um determinado valor de uma caixa de combinação seja selecionado. Neste caso você pode colocar os diversos controles em um controle Panel e tornar o controle Panel não visível. Quando um valor é selecionado na caixa de combinação, utilizamos um evento associado a esta seleção para detectar qual foi o valor selecionado. Dependendo do valor selecionado, podemos tornar True a propriedade Visible do controle Panel; com isso todos os controles que foram colocados no controle Panel também serão exibidos.

A sintaxe para este controle é a seguinte:

```
<asp:Panel
 id="Identificação_no_código"
 BackImageUrl="endereço da imagem de segundo plano."
 HorizontalAlign="Center|Justify|Left|NotSet|Right"
 Wrap="True|False"
 runat="server">
 Outros controles que terão o controle Panel como Container.
</asp:Panel>
```

**IMPORTANTE: 0**  
**controle Panel Web**  
**Server Control é derivado**  
**da classe base Panel, do**  
**namespace**  
**System.Web.UI.WebControls.**  
**Para uma referência**  
**completa a todas as**  
**propriedades e métodos**  
**deste controle, consulte a**  
**documentação do Frame-**  
**work .NET.**

Vamos analisar um exemplo retirado da documentação do Framework .NET. Neste exemplo é apresentado um formulário com os seguintes controles:

- ◆ Um controle Panel, com uma linha de texto de conteúdo estático. Durante o processamento da página, mais especificamente, durante o evento Load da página, vamos adicionar controles a este Panel, utilizando programação.
- ◆ Dois controles do tipo DropDownList (Caixa de Combinação). Em um controle, o usuário seleciona o número de rótulos a serem inseridos no Panel (de 0 a 4 rótulos). Em outro controle o usuário seleciona o número de Caixas de texto a serem inseridas no Panel.
- ◆ Um controle do tipo CheckBox que estiver marcado faz com que o Panel seja ocultado, o que também oculta todos os controles adicionados ao Panel.

Na Listagem 9.6 temos o código para o exemplo proposto.

### Listagem 9.6 – O controle Panel – chap9ex6.aspx.

```
<%@ Page Language="C#" %>

<html>
<head>
<script runat="server">
 void Page_Load(Object sender, EventArgs e) {

 // Exibe/Oculta o controle Panel.

 if (Check1.Checked)
 {
 Panel1.Visible=false;
 }
 else
 {
 Panel1.Visible=true;
 }
 // Adiciona controles do tipo Label ao controle Panel.

 int numlabels = Int32.Parse(DropDown1.SelectedItem.Value);

 for (int i=1; i<=numlabels; i++)
 {

```

```

// O nome da variável do tipo Label é um ele (1) e não o número um (1).

Label l = new Label();
l.Text = "Rótulo " + (i).ToString();
l.ID = "Rótulo " + (i).ToString();
Panel1.Controls.Add(l);
Panel1.Controls.Add(new LiteralControl("
"));
}

// Adiciona controles do tipo Caixa de Texto ao controle Panel.

int numtexts = Int32.Parse(DropDown2.SelectedItem.Value);

for (int i=1; i<=numtexts; i++)
{
 TextBox t = new TextBox();
 t.Text = "Caixa de Texto " + (i).ToString();
 t.ID = "Caixa de Texto" + (i).ToString();
 Panel1.Controls.Add(t);
 Panel1.Controls.Add(new LiteralControl("
"));
}
}

</script>
</head>
<body>
<h3>Exemplo do controle Panel!!</h3>

<form runat=server>

<asp:Panel id="Panel1" runat="server"
 BackColor="gainsboro"
 Height="200px"
 Width="400px"
 style="color:White;
 background-color:Black;
 border-color:Red;
 border-width:6px;
 border-style:Double;">

```

```
>

 Panel1: Conteúdo estático, definido no próprio controle

 <p>

</asp:Panel>

<p>Adicionar quantos rótulos?:

<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>

Adicionar quantas caixas de texto?:

<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="Ocultar o Painel" runat="server"/>

<p>

<asp:Button Text="Atualizar o Painel" runat="server"/>

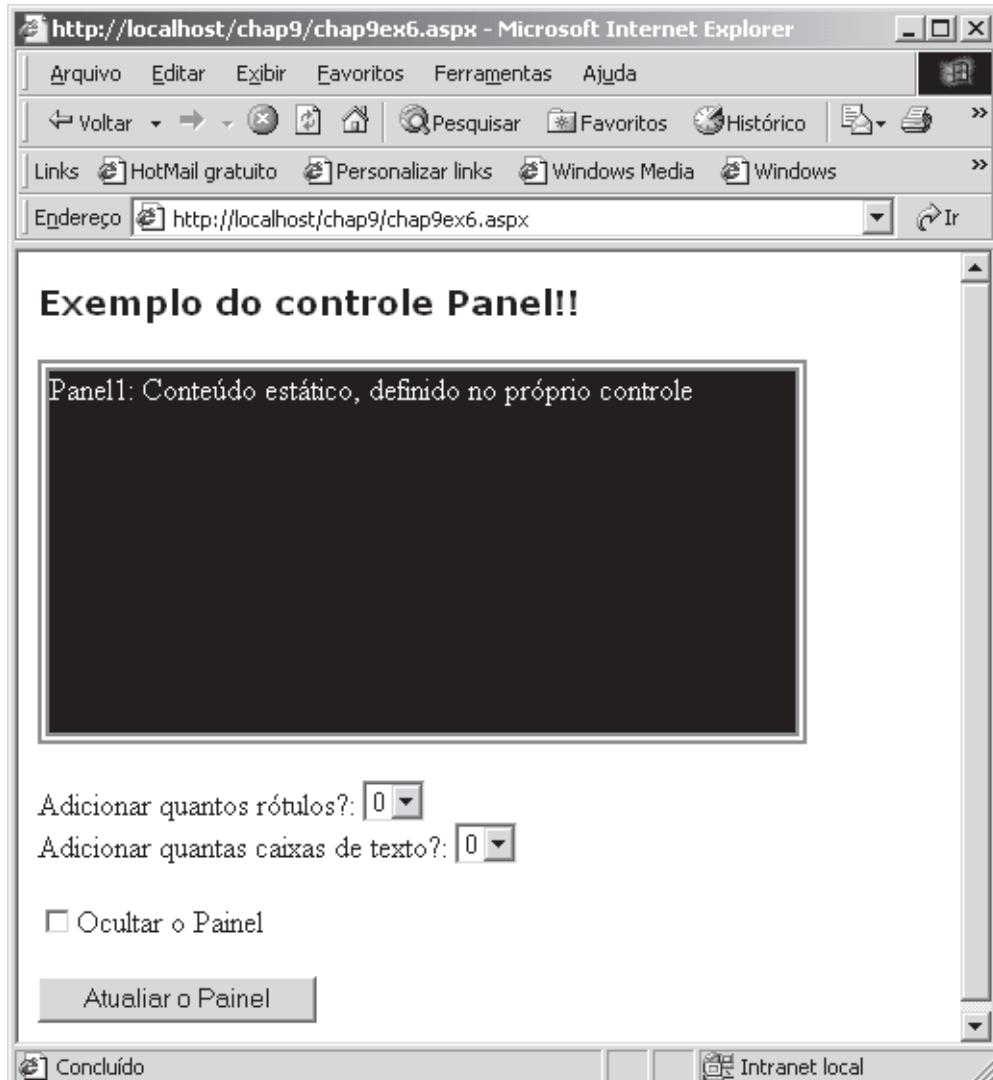
</form>

</body>
</html>
```

Digite o código da Listagem 9.6 e salve o mesmo em um arquivo chamado chap9ex6.aspx, na pasta chap9, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap9/chap9ex6.aspx>

É carregado o formulário com o controle Panel, dois controles Caixa de Combinação e um controle do tipo CheckBox, além do botão enviar, evidentemente. Você vai obter o resultado indicado na Figura 9.8.



**Figura 9.8: Utilizando o controle Panel – chap9ex6.aspx.**

Na lista “Adicionar quantos rótulos?”, selecione 3.

Na lista “Adicionar quantas caixas de texto?”, selecione 2.

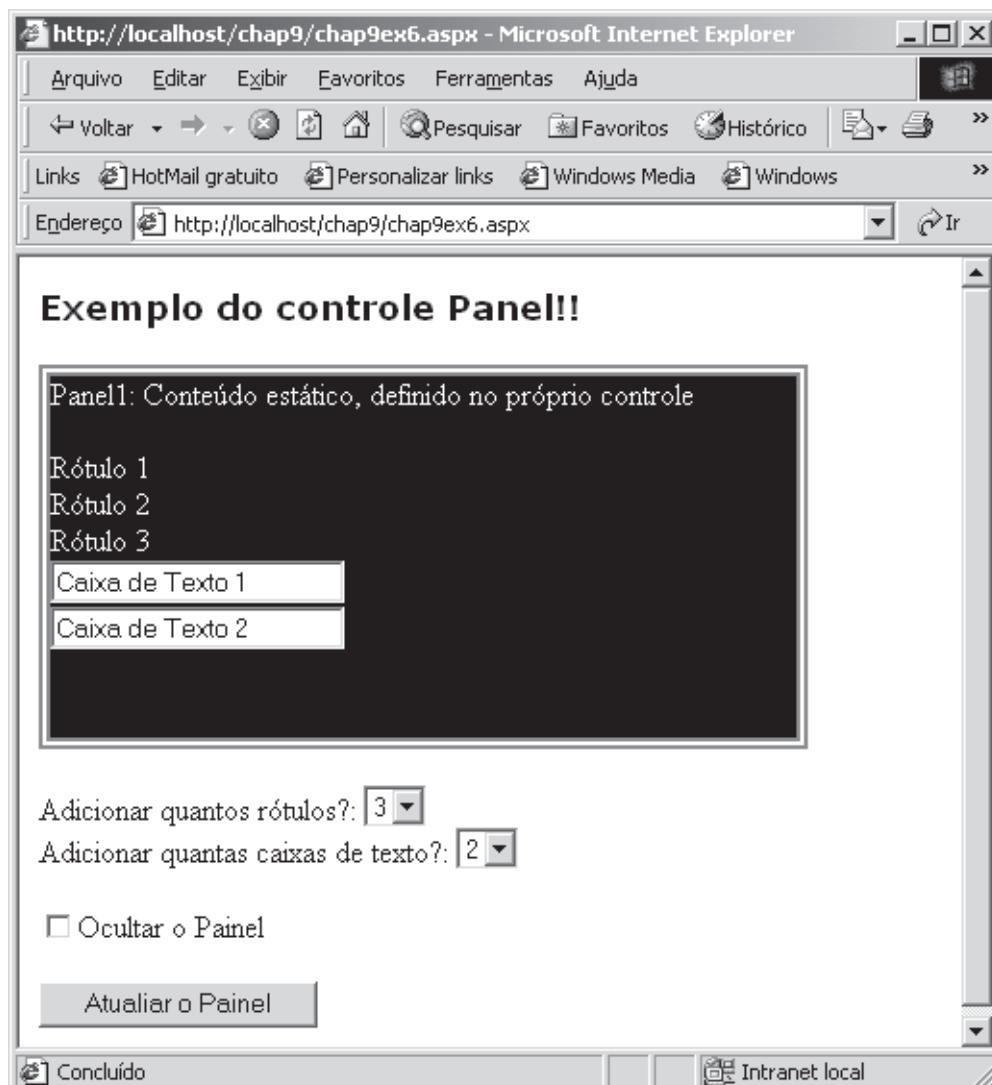
Certifique-se de que a opção “Ocultar Painel” esteja desmarcada.

Dê um clique no botão Enviar; você obterá os resultados indicados na Figura 9.9.

Agora marque a opção “Ocultar Painel” e dê um clique no botão Enviar. Observe que o controle Panel foi ocultado e também todos os controles a ele adicionados.

Alguns comentários sobre o código:

- ◆ No início do procedimento Load, testamos se o controle Check1 foi selecionado. Em caso afirmativo, tornamos a propriedade Visible do controle Panel1 igual a False. Isso faz com que o controle Panel e todos os seus controles sejam ocultados.



**Figura 9.9: Controles adicionados dinamicamente, através do evento Load da página.**

- ◆ Para adicionar o número de rótulos e o número de caixas de texto selecionados pelo usuário, utilizamos o evento Load da página.

O código para adicionar rótulos é o seguinte:

```
int numlabels = Int32.Parse(DropDown1.SelectedItem.Value);

for (int i=1; i<=numlabels; i++)

{
 Label 1 = new Label();
 1.Text = "Rótulo " + (i).ToString();
```

```

 l.ID = "Rótulo " + (i).ToString();

 Panel1.Controls.Add(l);

 Panel1.Controls.Add(new LiteralControl("
"));

}

```

Em primeiro lugar declaramos e inicializamos uma variável numlabels, do tipo int. Atribuímos o valor selecionado na lista DropDownList para a variável numlabels. Observe a utilização do método Parse, da estrutura Int32. Este método é utilizado para converter a String retornada pelo controle DropDownList, em um número Int32, equivalente.

Em seguida iniciamos um laço for. A cada passada do laço um controle l, do tipo Label, é criado. Depois definimos as suas propriedades Rótulo e ID. Em seguida utilizamos o método Add, da coleção Controls, do controle Panel, para adicionar o rótulo recém-criado – Panel1.Controls.Add(l). Para finalizar, utilizamos LiteralControl para enviar uma tag <br>, para fazer uma quebra de linha. Sem esta tag <br>, os rótulos seriam colocados um ao lado do outro e não um em cada linha.

As mesmas observações são válidas para a adição dos controles do tipo Caixa de texto.

- ◆ Este exemplo salienta os recursos que temos à disposição, quando utilizamos os Web Server Controls. Observe que temos acesso a uma infinidade de métodos e propriedades, todas bem definidas e documentadas na documentação do Framework .NET, no item “.NET Framework Class Library.”

## Image Web Server Control

Este controle é utilizado para adicionar imagens em uma página ASP.NET. Podemos definir um texto que será exibido no lugar da imagem, se por algum motivo o arquivo da imagem não estiver disponível.

Para definir este texto utilizamos a propriedade AlternateText.

A sintaxe para este controle é a seguinte:

```

<asp:Image
 id="Identificação_no_código"
 runat="server"
 ImageUrl="Endereço onde está a figura a ser incorporada na
 página."
 AlternateText="Texto"
 ImageAlign="NotSet|AbsBottom|AbsMiddle|Baseline|Bottom|Left|Middle|Right|TextTop|Top"
/>

```

Na Listagem 9.7 temos um pequeno exemplo de utilização do controle Image.

**IMPORTANTE: 0**  
**Controle Image Web**  
**Server Control é derivado**  
**da classe base Image, do**  
**namespace**  
**System.Web.UI.WebControls.**  
**Para uma referência**  
**completa a todas as**  
**propriedades e métodos**  
**deste controle, consulte a**  
**documentação do Frame-**  
**work .NET.**

**Listagem 9.7 – O controle Image – chap9ex7.aspx.**

```
<html>
<head>
</head>
<body>
<form runat="server">
 <h2>SQL Server 2000 - Curso Completo!!</h2>
 <h3>Por: Júlio Battisti</h3>
 <h3>
 Editora Axcel Books</h3>

 <asp:Image
 id="image1"
 runat="server"
 AlternateText="SQL Server 2000 - Curso Completo"
 ImageAlign="left"
 ImageUrl="livrosql2000.jpg"
 />

</form>
</body>
</html>
```

Digite o código da Listagem 9.7 e salve o mesmo em um arquivo chamado chap9ex7.aspx, na pasta chap9, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap9/chap9ex7.aspx>

Você vai obter o resultado indicado na Figura 9.10.

Observe que, embora não tenhamos nenhum código na seção de código (na verdade nem temos seção de código neste exemplo), mesmo assim utilizamos a extensão .aspx e a página foi processada normalmente.

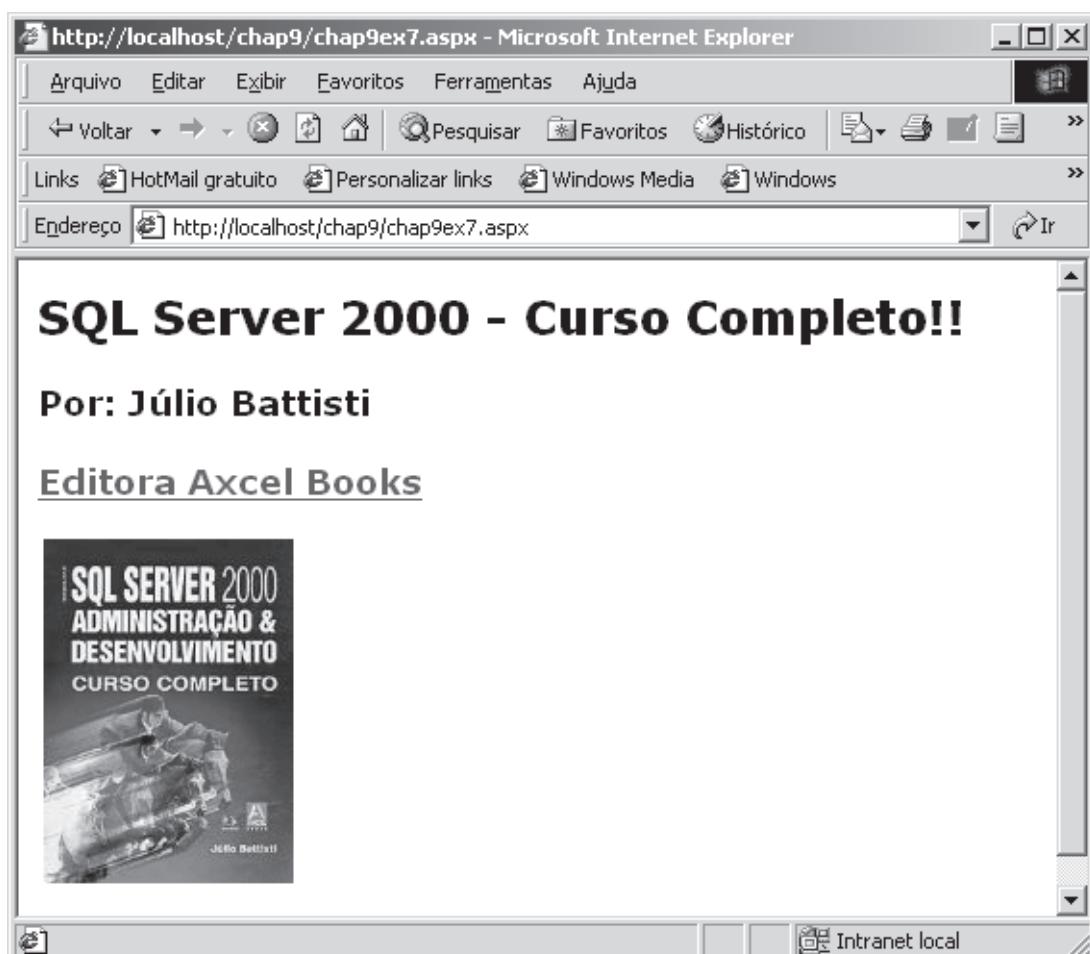


Figura 9.10: Utilizando o controle Image – chap9ex7.aspx.

## HyperLink Web Server Control

Este controle é utilizado para a criação de links em uma página ASP.NET. Por ser um controle do tipo Web Server Control, temos acesso a uma série de métodos e eventos do controle HyperLink.

A sintaxe para este controle é a seguinte:

```
<asp:HyperLink
 id="Identificação_no_código."
 NavigateUrl="Endereço de destino do link."
 Text="Texto do link"
 ImageUrl="Imagen de destino do link, se for o caso."
 Target="Onde o destino será aberto, especialmente útil,
 quando utilizamos frames."
 runat="server"
/>
ou
```

**NOTA:** O Controle HyperLink Web Server Control é derivado da classe base HyperLink, do namespace System.Web.UI.WebControls. Para uma referência completa a todas as propriedades e métodos deste controle, consulte a documentação do Framework .NET.

```
<asp:HyperLink
 id="Identificação_no_código."
 NavigateUrl="Endereço de destino do link."
 ImageUrl="Imagen de destino do link, se for o caso."
 Target="Onde o destino será aberto, especialmente útil, quando utilizamos
frames."
 runat="server"

 Texto do Link.

</asp:HyperLink>
```

Na Listagem 9.8 temos um pequeno exemplo de utilização do controle HyperLink.

### Listagem 9.8 – O controle HyperLink – chap9ex8.aspx.

```
<html>
<head>
</head>
<body>
<h3>Exemplo do controle HyperLink. </h3>
Clique no link abaixo:

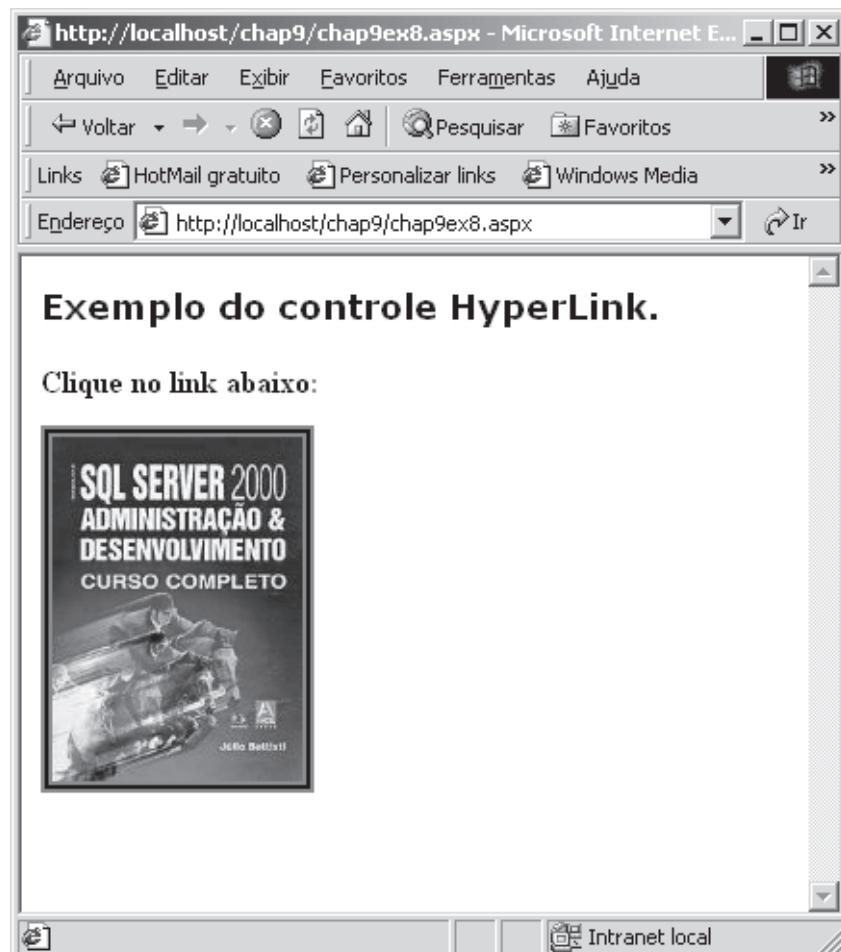
 <asp:HyperLink
 id="hyperLink1"
 ImageUrl="livrosql2000.jpg"
 NavigateUrl="http://www.axcel.com.br/descrição.cfm?id_livro=202"
 Text="SQL Server 2000 - Curso Completo!!"
 Target="_new"
 runat="server"
 style="color:White;
 background-color:Black;
 border-color:Red;
 border-width:6px;
 border-style:Double;"
 />

</body>
</html>
```

Digite o código da Listagem 9.8 e salve o mesmo em um arquivo chamado chap9ex8.aspx, na pasta chap9, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap9/chap9ex8.aspx>

Você vai obter o resultado indicado na Figura 9.11.



**Figura 9.11: Utilizando o controle HyperLink – chap9ex8.aspx.**

Neste caso, a figura é o link. Ao clicar na figura será aberta uma nova janela (conforme definido na propriedade target=\_new), onde será carregada a página definida na propriedade NavigateUrl=[http://www.axcel.com.br/descricao.cfm?id\\_livro=202](http://www.axcel.com.br/descricao.cfm?id_livro=202).

## LinkButton Web Server Control

Este controle é utilizado para a criação de um controle com aparência de link e com funcionalidade de botão. Este controle pode ser do tipo Submit ou do tipo Command.

Um LinkButton do tipo Enviar não tem um nome de comando (definido pela propriedade CommandName) associado com o botão. Este botão simplesmente envia a página para processamento no servidor. Por padrão um controle Button é do tipo Enviar. Podemos definir um procedimento que executa em resposta ao evento Click do botão, como já fizemos em diversos exemplos deste livro.

Um LinkButton do tipo Command possui um nome de comando associado com o botão, nome este que é definido na propriedade CommandName do controle. Isso permite que sejam criados múltiplos controles do tipo LinkButton, em um formulário (Web Form) de uma página ASP.NET. Através de código podemos determinar qual o botão que foi “clicado” e, para cada botão, podemos definir o código que será executado em resposta ao evento Click do respectivo botão.

A sintaxe para este controle é a seguinte:

```
<asp:LinkButton
 id="Identificação_no_código"
 Text="Texto do LinkButton"
 Command=" Nome de comando. Define o controle como sendo do
 tipo Command "
 CommandArgument="CommandArgument"
 OnClick="Método que executa em resposta ao evento Click do
 controle."
 runat="server"/>
```

ou

```
<asp:LinkButton
 id="Identificação_no_código"
 Command=" Nome de comando. Define o controle como sendo do
 tipo Command "
 CommandArgument="CommandArgument"
 OnClick="Método que executa em resposta ao evento Click do controle."
 runat="server"
 >
 Texto do LinkButton
</asp:LinkButton>
```

Na Listagem 9.9 temos um pequeno exemplo de utilização do controle LinkButton.

#### Listagem 9.9 – O controle LinkButton – chap9ex9.aspx.

```
<html>
<head>
 <script language="C#" runat="server">
```

**NOTA:** O Controle LinkButton Web Server Control é derivado da classe base LinkButton, do namespace System.Web.UI.WebControls. Para uma referência completa a todas as propriedades e métodos deste controle, consulte a documentação do Framework .NET.

```

void LinkButton1_Click(Object sender, EventArgs e)
{
 Label1.Text="Você clicou em um controle LinkButton";
}

</script>

</head>
<body>
<form runat=server>

<h3>Exemplo do controle LinkButton!!</h3>

<asp:LinkButton
 id="LinkButton1"
 Text="Clique, por favor!"
 Font-Name="Verdana"
 Font-Size="14pt"
 OnClick="LinkButton1_Click"
 runat="server"
/>

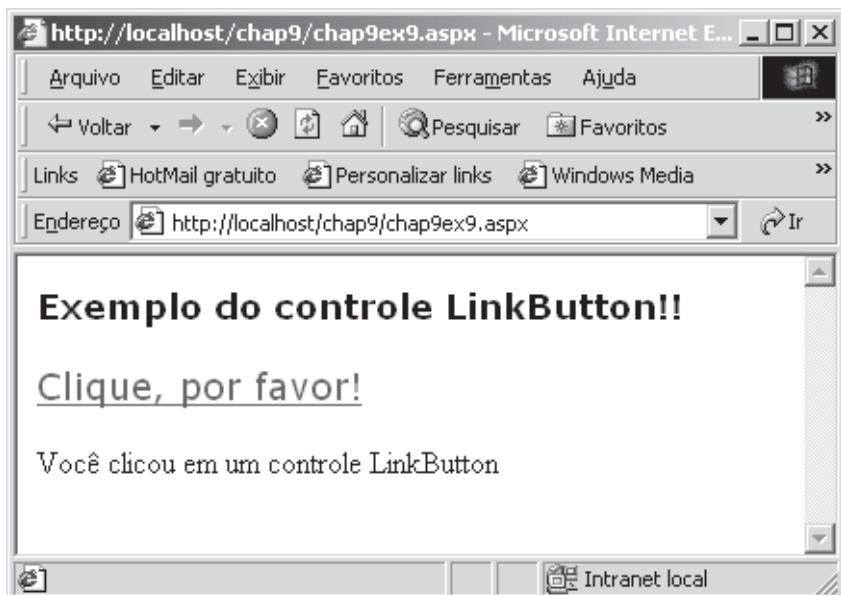
<asp:Label
 id="Label1"
 runat=server
/>
</form>
</body>
</html>

```

Digite o código da Listagem 9.9 e salve o mesmo em um arquivo chamado chap9ex9.aspx, na pasta chap9, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap9/chap9ex9.aspx>

Na página que é carregada, dê um clique no link “Clique, por favor!”. Você obterá o resultado indicado na Figura 9.12.



**Figura 9.12: Utilizando o controle LinkButton – chap9ex9.aspx.**

Observe que podemos definir um método em resposta ao evento Click do controle LinkButton, conforme descrevemos antes: Aparência de Link e funcionalidade de botão.

## ImageButton Web Server Control

Este controle é utilizado para detectar a região de uma determinada imagem, onde o usuário clicou com o mouse. Com isso podemos criar a mesma funcionalidade de uma imagem mapeada, no HTML.

Utilizamos o evento OnClick para determinar as coordenadas onde o usuário clicou na imagem. Com isso, dependendo do valor das coordenadas onde o usuário clicou, podemos tomar diferentes ações, como, por exemplo, carregar diferentes endereços. Com isso estamos criando, na prática, uma imagem mapeada, onde diferentes regiões da imagem estão vinculadas a diferentes endereços.

Podemos utilizar o evento OnCommand para fazer com que a imagem tenha o mesmo comportamento de um botão de comando. Podemos associar um nome de comando com o botão, nome este que é definido na propriedade CommandName do controle. Isso permite que sejam criados múltiplos controles do tipo LinkButton, em um formulário (Web Form) de uma página ASP.NET. Através de código podemos determinar qual o botão que foi “clicado” e, para cada botão, podemos definir o código que será executado em resposta ao evento Click do respectivo botão.

A sintaxe para este controle é a seguinte:

```
<asp:ImageButton
 id="Identificação_no_código"
 ImageUrl="Endereço do arquivo de imagem."/>
```

**NOTA: O Controle ImageButton Web Server Control é derivado da classe base ImageButton, do namespace System.Web.UI.WebControls. Para uma referência completa a todas as propriedades e métodos deste controle, consulte a documentação do Framework .NET.**

```
Command="Command"
CommandArgument="CommandArgument"
OnClick="Método que será executado em resposta ao evento Click do controle."
runat="server"
/>>
```

Na Listagem 9.10 temos um exemplo de utilização do controle ImageButton.

**Listagem 9.10 – O controle ImageButton – chap9ex10.aspx.**

```
<%@ Page Language="C#" %>

<html>
<head>

<script runat="server">

 void ImageButton_Click(object Source, ImageClickEventArgs e)

 {
 Label1.Text="Você clicou nas seguintes coordenadas:" +
 "(" + e.X.ToString() + ", " +
 e.Y.ToString() + ")";
 }
</script>
</head>

<body>
<form runat="server">

 <h3>Exemplo do controle ImageButton!! </h3>
 Clique em qualquer ponto da imagem.

 <asp:ImageButton
 id="imagebutton1"
 AlternateText="Axcel Books.">
</form>
```

```
 ImageAlign="left"
 ImageUrl="livrosql2000.jpg"
 OnClick="ImageButton_Click"
 runat="server"
 />

<asp:Label
 id="Label1"
 runat="server"
/>

</form>

</body>
</html>
```

Digite o código da Listagem 9.10 e salve o mesmo em um arquivo chamado chap9ex10.aspx, na pasta chap9, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap9/chap9ex10.aspx>

Na página que é carregada, dê um clique em qualquer ponto da imagem; serão informadas as coordenadas do ponto onde você clicou, conforme indicado na Figura 9.13.

O método ImageButton\_Click é executado em resposta ao evento Click do controle ImageButton. Um dos argumentos (argumento e) deste método é do tipo ImageClickEventArgs. Este argumento é um objeto baseado na classe ImageClickEventArgs. Esta classe possui dois campos:

**NOTA:** A origem das coordenadas (0,0) é no canto superior esquerdo.

- ◆ **X:** Retorna a coordenada horizontal, em relação à origem.
- ◆ **Y:** Retorna a coordenada vertical, em relação à origem.

Com estes dois campos podemos acessar as coordenadas do ponto onde o usuário clicou.

Para acessar a coordenada X, utilizamos o seguinte comando:

```
e.X.ToString()
```

Para acessar a coordenada Y, utilizamos o seguinte comando:

```
e.Y.ToString()
```

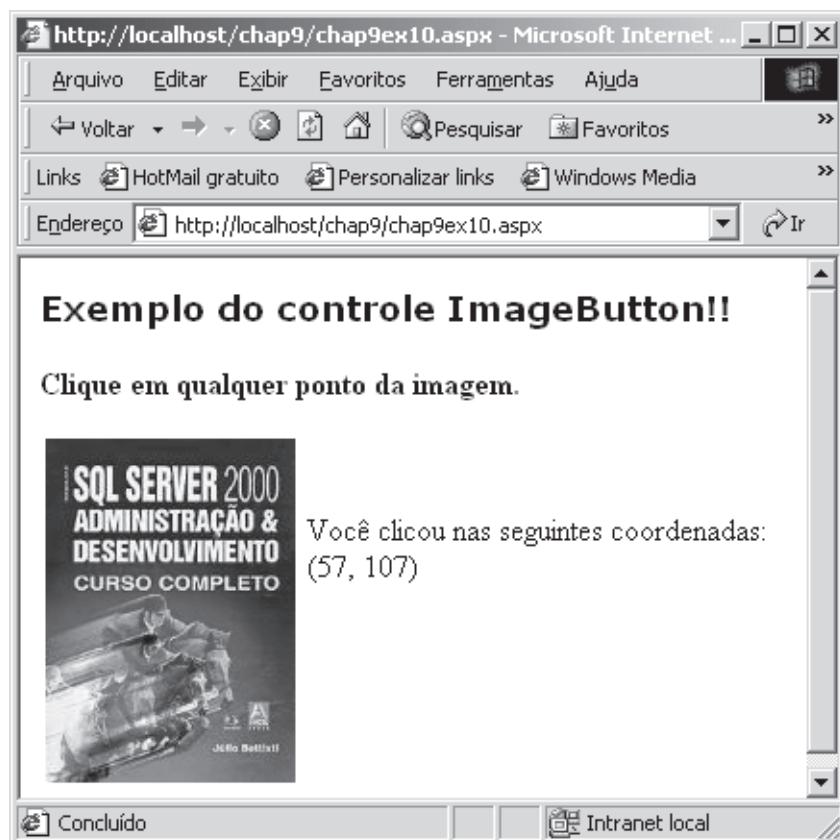


Figura 9.13: Utilizando o controle ImageButton – chap9ex10.aspx.

## Conclusão

Iniciamos o capítulo estudando a classe base para todos os Web Server Controls: WebControl.

Em seguida, aprendemos a utilizar controles pertencentes aos chamados Web Server Controls. Estudamos e apresentamos exemplos sobre os seguintes controles:

- ◆ TextBox
- ◆ Label
- ◆ CheckBox
- ◆ RadioButton
- ◆ Button
- ◆ ListBox
- ◆ Table, TableCell e TableRow
- ◆ Panel
- ◆ Image
- ◆ HyperLink

**NOTA:** Utilizamos o método `ToString`, para converter o valor inteiro, retornado pelos campos X e Y, na string correspondente.

- ◆ LinkButton
- ◆ ImageButton

A principal questão que pode surgir é a seguinte: “Por que precisamos de mais um conjunto de controles, se já temos os HtmlServer Controls e os Validation Server Controls (que na verdade fazem parte dos Web Server Controls)?

O principal motivo, conforme foi ressaltado durante o capítulo, é que os Web Form Controls fornecem um conjunto mais rico e variado de propriedades, métodos e eventos do que os HtmlServer Controls. Esta variedade de métodos, propriedades e eventos permite a criação de páginas ASP.NET mais sofisticadas e nas quais temos um controle apurado sobre os elementos da página, controle esse exercido através de código de programação.

Agora já conhecemos os elementos básicos do Framework .NET (Capítulos 1 e 2), os elementos básicos da linguagem C# (Capítulos 3, 4 e 5) e os elementos básicos para a criação de páginas ASP.NET (Capítulos 7, 8 e 9). A partir do próximo capítulo vamos estudar a conexão de páginas ASP.NET com diversas fontes de dados, através da utilização de ADO.NET.

# Introdução

Você lembra do tempo em que para construir um site era só criar um “monte” de páginas HTML? Eu lembro. Se eu tenho saudades? Sinceramente, não. Embora fosse muito mais simples e fácil, o que se pode fazer apenas com HTML é muito pouco.

Hoje, a realidade é bem diferente. Vivemos em um mundo com milhões de pessoas conectadas à Internet. O comércio eletrônico entre empresas (B2B – Business To Business) e o comércio eletrônico entre empresas e consumidores finais (B2C – Business To Consumer) já é uma realidade.

Mas a Internet não é utilizada apenas para comércio eletrônico. Sistemas de gerenciamento das relações com o cliente – CRM – Customer Relationship Management, software para ser utilizado pela Internet, mediante o pagamento de uma taxa mensal – ASP – Application Services Providers, são apenas alguns exemplos dos serviços que se tornaram realidade graças à utilização da Internet.

Novas ofertas e serviços surgem diariamente. Mas o que têm em comum todos estes sites?

A tecnologia? De maneira alguma. Existem tecnologias para os mais variados gostos, desde soluções 100% Java, passando pelo software livre, com a utilização do servidor Apache, Linux e da linguagem PHP, até o mundo Microsoft, antes com a tecnologia ASP e o padrão COM/COM+ e agora com o Framework .NET.

O que todos estes sites e aplicações Web têm em comum é a necessidade, cada vez maior, do acesso a dados das mais variadas fontes. Existem exemplos de aplicações Web que apresentam, na mesma página, dados oriundos do Mainframe, de um servidor SQL Server ou ORACLE e de uma planilha do Excel. O fato é que o acesso aos dados é uma necessidade. Com o uso da Informática um volume cada vez maior de dados é gerado, diariamente, nas empresas. Para transformar todos estes dados, em informações úteis, geradoras de negócios e lucros, precisamos de tecnologias que facilitem o acesso às mais diversas e variadas fontes de dados.

Com o ASP 3.0 temos a tecnologia ADO/OLE-DB (Activex Data Objects/ OLE Database) para acesso a fontes de dados. Com ASP.NET temos um novo conjunto de classes para acesso a dados, conjunto este conhecido como ADO.NET.

É importante salientar que ADO.NET não é uma nova versão de ADO. As duas tecnologias podem ser utilizadas em conjunto. ADO dando suporte a páginas e aplicações Web criadas com ASP 3.0; e para páginas ASP.NET utilizamos

# CAPÍTULO 10

## Acessando Bancos de Dados com ASP.NET – Parte 1

ADO.NET. Esta “convivência” entre as tecnologias, facilita a migração das páginas criadas com ASP 3.0 para ASP.NET, uma vez que as páginas ASP 3.0 continuarão funcionando, mesmo após a instalação do Framework .NET.

Como acessar bancos de dados a partir de páginas ASP.NET é o assunto deste capítulo. Vamos apresentar os namespaces onde estão as classes para acesso a dados. Iremos estudar as principais classes destes namespaces. Também aprenderemos a acessar bancos de dados do Microsoft SQL Server e do Microsoft Access. Veremos como apresentar os dados obtidos, utilizando os novos controles do ASP.NET.

Veremos diversos exemplos de utilização das classes estudadas. Sempre que for pertinente, faremos a comparação entre a maneira como uma determinada operação é realizada com ASP 3.0 e a maneira como passamos a realizar a mesma operação com ASP.NET. O conteúdo visto neste capítulo é a base para os assuntos apresentados nos Capítulos 11 e 12.

## Uma Visão Geral do Acesso a Dados

Existe uma frase que resume bem a necessidade de acesso a dados: “A informação certa, para a pessoa certa, na quantidade certa e no momento certo”.

Esta frase apresenta alguns aspectos importantes:

- ◆ **A informação certa:** A informação deve estar correta, isto é, deve ser confiável. A informação é utilizada para a tomada de decisões, nos mais variados níveis de uma empresa. Se a informação estiver incorreta, as decisões serão equivocadas e não irão gerar os resultados esperados. Pior do que não ter a informação é ter informação incorreta, não confiável.
- ◆ **Para a pessoa certa:** Independente do nível hierárquico, quer seja o operário da fábrica ou o presidente da empresa, todos precisam de informação para trabalhar e alcançar os resultados desejados. A informação correta precisa estar à disposição para as pessoas que dela necessitam.
- ◆ **Na quantidade certa:** Informação demais também é um problema e pode atrapalhar ao invés de ajudar. E hoje as informações vêm das mais diversas fontes. Internet, e-mail, relatórios, banco de dados da empresa, arquivos do Office, jornais, revistas, livros, manuais técnicos. Precisamos garimpar as informações que realmente são necessárias ao nosso trabalho.
- ◆ **No momento certo:** Este é um aspecto fundamental. Lembro-me da época em que somente tínhamos o Mainframe e um “bando” de terminais espalhados pela empresa. Você solicitava um novo relatório para a turma do CPD. Duas semanas depois vinha o relatório. Hoje, evidentemente, esta situação é inaceitável. Precisamos da informação instantânea, sempre à disposição.

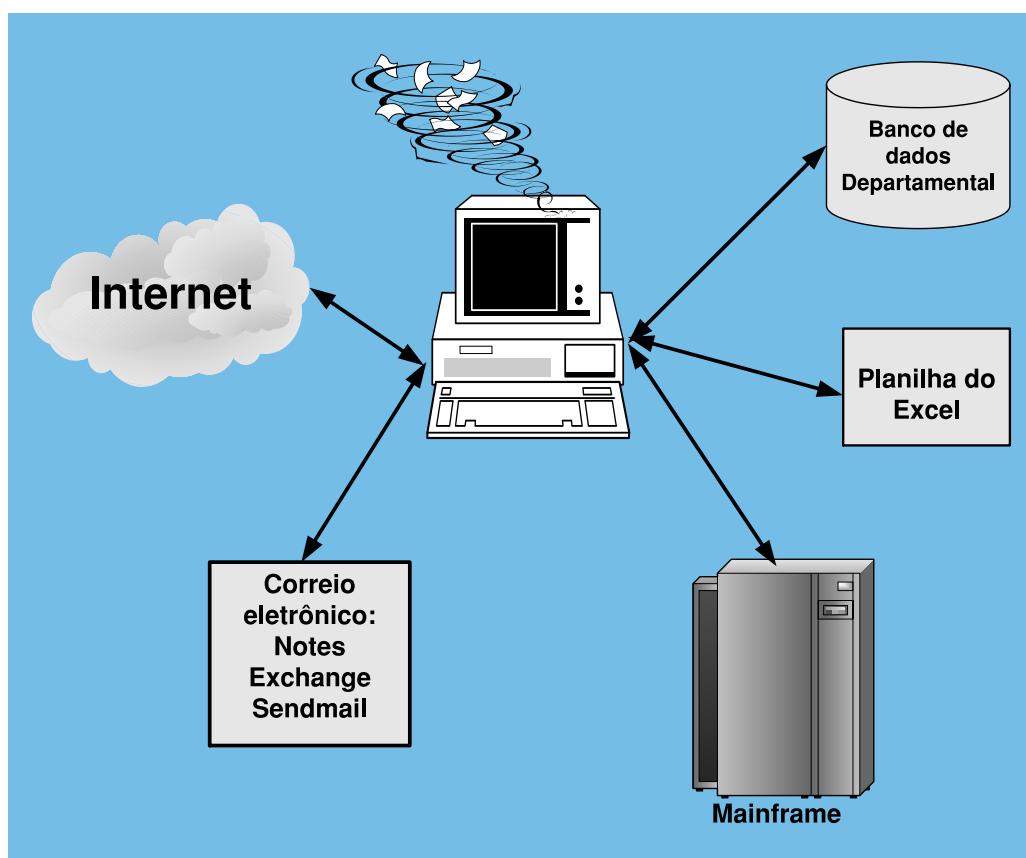
Quando as empresas começaram a descentralizar seus ambientes computacionais, passando do modelo Mainframe/Terminal para um modelo Cliente/Servidor com redes locais, novas possibilidades, e também problemas, começaram a surgir. Bancos de dados departamentais não integrados, diferentes formatos de dados, aplicações Cliente/Servidor rodando em cada estação de trabalho, etc., isso fez com que a informação ficasse distribuída por toda a empresa e não mais centralizada no Mainframe.

Um ambiente descentralizado é mais flexível, mas em contrapartida é mais difícil de se gerenciar e de manter o controle. À medida que o volume de informações começou a crescer, ficou difícil ter acesso aos dados de diferentes departamentos de uma maneira consistente e rápida. Na Figura 10.1 temos uma pequena ilustração deste ambiente:

Além de múltiplas fontes de dados, temos diferentes formatos de dados. Evidentemente que a estrutura de um arquivo de mensagens de correio é muito diferente da estrutura de uma planilha do Excel, por exemplo.

Para que possamos criar aplicações que acessam dados de diversas fontes, precisamos de tecnologias capazes de acessar dados de fontes tão diversas como o Mainframe e um documento do Microsoft Word ou uma planilha do Microsoft Excel. O Framework .NET fornece, principalmente através do namespace System.Data, uma série de classes para conexão e manipulação de dados dos mais variados formatos. Este conjunto de classes também é conhecido como ADO.NET.

Neste capítulo vamos estudar as classes que dão acesso a fontes de dados estruturadas, que seguem o modelo de dados relacionais. Podemos citar como exemplos de bancos de dados relacionais o Microsoft Access, o SQL Server 2000, o ORACLE, o DB2 da IBM, etc.



**Figura 10.1: Dados de múltiplas e heterogêneas fontes.**

**NOTA:** Para maiores detalhes sobre o modelo de dados relacionais, consulte o Anexo II.

## Quais as Principais Diferenças do ADO.NET em Relação ao ADO?

Com ADO estabelecíamos uma conexão com uma fonte de dados e utilizávamos, ou o método Open do Objeto Connection, ou um objeto do tipo Command, para executar um comando SQL e retornar dados, no formato de uma tabela, para um objeto do tipo Recordset. A característica principal é que, para trabalhar com os dados e fazer alterações nos mesmos, precisamos estabelecer uma conexão com o banco de dados.

Com ADO.NET temos um modelo de dados “desconectado”. O protocolo HTTP é conhecido como Connection Less. Ao solicitar uma página, é estabelecida uma conexão com o servidor Web; os arquivos e imagens necessários são solicitados e enviados para o navegador do cliente e a conexão é fechada. Por isso o termo Connection Less, pois não é mantida uma conexão permanente entre o servidor Web e o navegador do cliente. Esta natureza “desconectada” da Web causa alguns inconvenientes quando desenvolvemos aplicações de banco de dados.

ADO.NET resolve este problema, fornecendo uma série de objetos que permitem que os dados, uma vez carregados para o navegador do cliente, sejam manipulados, mesmo sem existir uma conexão permanente com o banco de dados. Uma vez feitas as alterações, inclusões ou exclusões necessárias, podemos, facilmente, sincronizar os dados que estão no cliente com o banco de dados no servidor. Veremos nos Capítulos 10 e 11 exemplos de métodos para sincronização dos dados.

O principal objeto do modelo ADO, para trabalhar com dados, é o objeto RecordSet. Para exibir os dados de um objeto RecordSet, precisamos escrever uma boa quantia de código. Com ADO.NET, o principal objeto, que funciona como um Conteiner para dados, é o DataSet, o qual pode conter uma ou mais tabelas (o objeto RecordSet somente pode conter uma tabela) e também os relacionamentos entre as tabelas e informações sobre chaves primárias e chaves estrangeiras.

Com o modelo “desconectado” do ADO.NET, não utilizamos cursores, nem no lado cliente, nem no lado servidor. As classes de acesso a dados fornecem os mecanismos necessários para a manipulação dos dados. O formato adotado para armazenar os dados no cliente é o XML, um padrão amplamente aceito pela indústria. Com ADO, ao utilizar tecnologias como RDS para trabalhar com dados desconectados, temos um formato proprietário para os dados, formato este que somente é aceito pelo Internet Explorer.

Neste capítulo vamos aprender a utilizar alguns objetos básicos para o acesso a dados relacionais, mais especificamente, a dados do SQL Server e do Microsoft Access. Veremos como estabelecer uma conexão com o banco de dados, acessar dados de uma ou mais tabelas e exibir estes dados em uma página ASP.NET.

Para exibição dos resultados obtidos, estaremos utilizando o Web Server Control – DataGrid. Estudaremos este controle em detalhes. Veremos que o controle DataGrid facilita, enormemente, a tarefa de exibir dados em uma página ASP.NET. No nosso exemplo mais simples, veremos que, o que no ASP 3.0 exige várias linhas de código, no ASP.NET, com o controle DataGrid, pode ser feito em uma única linha de código.

**NOTA:** Para maiores informações sobre Tabelas, Atributos, Chaves Primárias, Chaves Estrangeiras e Relacionamentos entre tabelas, consulte o Anexo II.

# Bancos de Dados Utilizados nos Exemplos

Para os exemplos deste e dos próximos capítulos utilizaremos um banco de dados do Microsoft Access e outro do SQL Server 2000.

## O Banco de Dados do Microsoft Access – NorthWind.mdb

O banco de dados NorthWind.mdb é fornecido pela Microsoft e instalado juntamente com o Microsoft Access 2000. É um banco de dados de exemplo, para controle de vendas de uma pequena empresa. Nele são armazenadas informações sobre pedidos, clientes, funcionários, produtos e fornecedores.

A arquivo NorthWind.mdb, por padrão, é instalado na subpasta Samples, da pasta de instalação do Office 2000. Para os exemplos deste livro, faremos uma cópia deste arquivo, na pasta C:\Meus documentos e utilizaremos esta cópia nos exemplos.

Neste banco de dados encontramos as seguintes tabelas:

- ◆ Categorias
- ◆ Clientes
- ◆ Detalhes do pedido
- ◆ Fornecedores
- ◆ Funcionários
- ◆ Pedidos
- ◆ Produtos
- ◆ Transportadoras

Na Figura 10.2 temos o diagrama Entidades x Relacionamentos para este banco de dados:

**NOTA:** Os exemplos deste capítulo serão criados na pasta D:\Inetpub\wwwroot\Chap10. Para acessar uma página, dentro desta pasta, por exemplo: Chap10Ex1.aspx, utilize o seguinte endereço: <http://localhost/Chap10/Chap10Ex1.aspx>. A seguir descrevo os bancos de dados que serão utilizados nos exemplos deste e dos próximos capítulos.

**IMPORTANTE:** Os nomes de empresas, produtos, pessoas, personagens e/ou dados apresentados neste banco de dados são fictícios e não representam de forma alguma qualquer indivíduo, produto, empresa ou evento, salvo menção contrária.

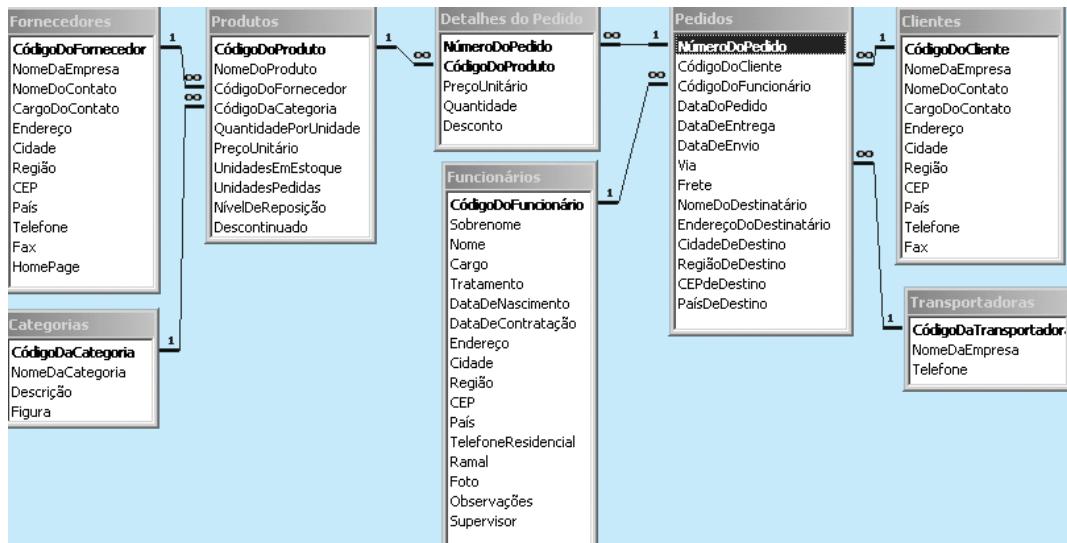


Figura 10.2: O Banco de dados NorthWind.mdb do Microsoft Access.

## O Banco de Dados do SQL Server – Pubs

Ao instalarmos o Framework .NET é instalada, digamos assim, uma “miniversão” do SQL Server. Nesta “miniversão” é disponibilizado o banco de dados pubs, conforme indicado na Figura 10.3.

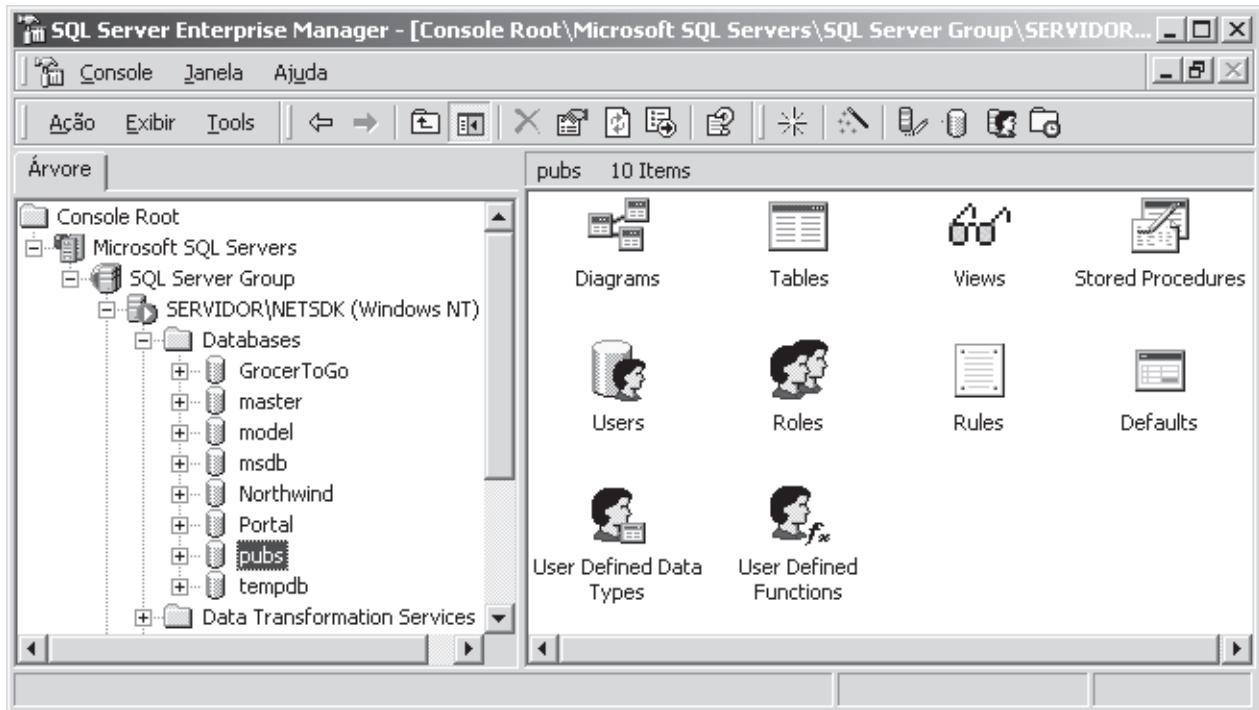


Figura 10.3: O Banco de dados pubs do SQL Server.

Ao instalar o Framework .NET podemos ter duas situações distintas:

1. O SQL Server ainda não está instalado: Neste caso é criada uma instância chamada NETSDK. Para estabelecermos uma conexão, conforme veremos nos exemplos deste capítulo, fornecemos o nome da instância – NETSDK, como valor para o parâmetro server.
2. O SQL Server já está instalado: Neste caso é criada uma nova instância chamada SERVIDOR\NETSDK, onde SERVIDOR é o nome do computador que você está utilizando. Para estabelecermos uma conexão, conforme veremos nos exemplos deste capítulo, fornecemos o nome completo da instância – SERVIDOR\NETSDK, como valor para o parâmetro server. O computador que estou utilizando para os exemplos encaixa-se neste segundo caso. Nos exemplos deste capítulo vou utilizar – SERVIDOR\NETSDK, como valor para o parâmetro Servidor, nos objetos onde este parâmetro for necessário. Se você estiver utilizando um computador com um nome diferente, substitua SERVIDOR pelo nome do computador que você está utilizando.

**IMPORTANTE:** Os nomes de empresas, produtos, pessoas, personagens e/ou dados apresentados neste banco de dados são fictícios e não representam de forma alguma qualquer indivíduo, produto, empresa ou evento, salvo menção contrária.

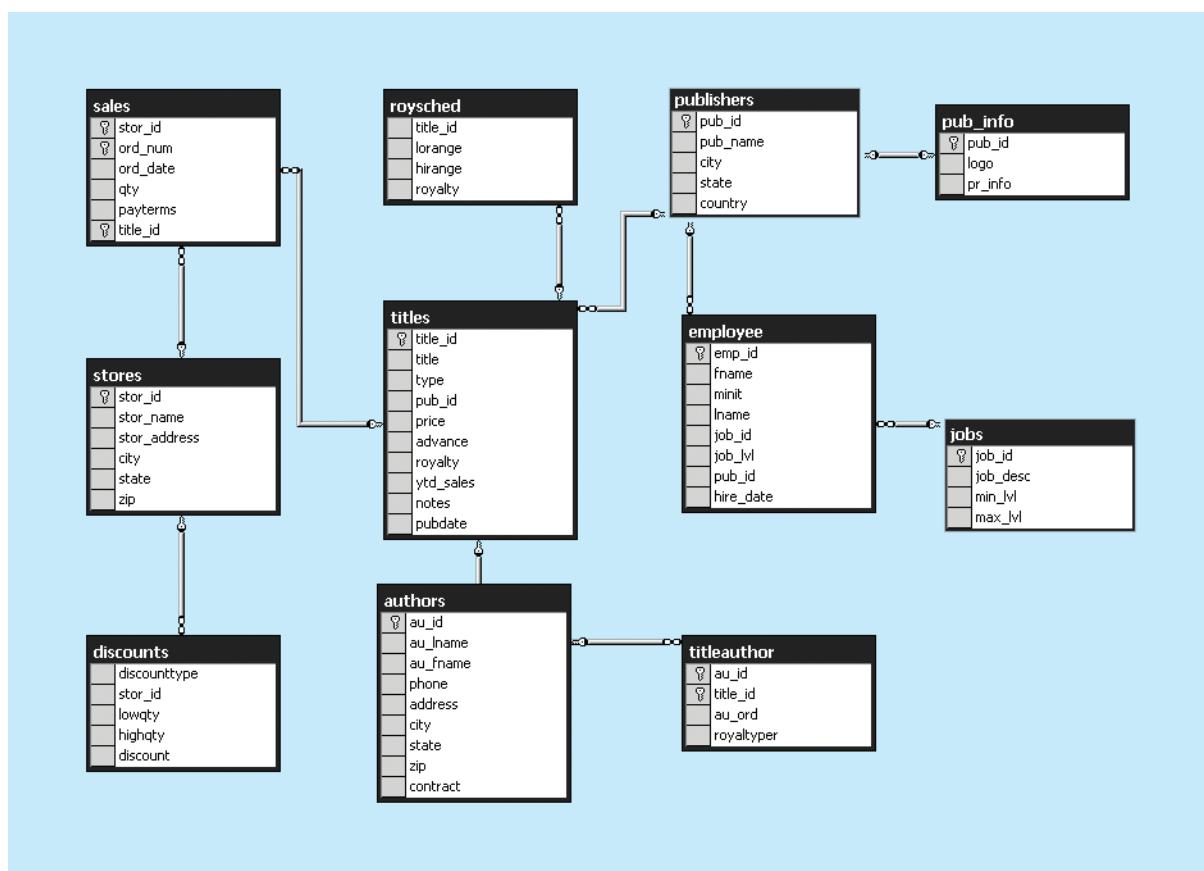
As principais tabelas deste banco de dados são as seguintes:

- ◆ authors (autores)
- ◆ discounts (descontos)
- ◆ employee (funcionários, empregados)
- ◆ jobs (funções, cargos)
- ◆ publishers (editoras)
- ◆ sales (vendas)
- ◆ stores (lojas, livrarias)
- ◆ titles (livros, títulos)
- ◆ titleauthor (relaciona os livros de cada autor)

**NOTA:** Os nomes de tabelas e campos do Banco de dados pubs estão em inglês, conforme fornecido na instalação do Framework .NET.

O Banco de dados pubs é utilizado por uma rede de livrarias, para o controle de vendas e pesquisa dos títulos existentes no catálogo da livraria.

Na Figura 10.4 temos o diagrama Entidades x Relacionamentos para este Banco de dados.



**Figura 10.4: O diagrama Entidades x Relacionamentos do Banco de dados pubs.**

## Uma Introdução ao ADO.NET

Agora vamos iniciar o estudo das principais classes para acesso a dados. O conjunto destas classes é conhecido como ADO.NET. As classes que iremos estudar fazem parte dos seguintes namespaces da biblioteca de classes do Framework .NET:

- ◆ **System.Data:** Contém as principais classes utilizadas para acessar bases de dados relacionais. A classe DataSet faz parte deste namespace.
- ◆ **System.Data.SqlClient:** Classes utilizadas para acessar bancos de dados do SQL Server 2000. As classes deste namespace fornecem melhor desempenho para acesso ao SQL Server, pois utilizam a interface TDS – Tabular Data Stream, nativa do SQL Server 2000.
- ◆ **System.Data.Common:** Contém as classes onde são definidas as propriedades e métodos básicos, herdados por classes de outros namespaces.
- ◆ **System.Data.OleDb:** Neste namespace encontramos as classes para acesso a fontes de dados, via OLE-DB Providers.

**NOTA:** Os campos indicados por uma pequena chave amarela são campos do tipo Chave Primária. Os campos indicados pelo sinal de infinito (um 8 deitado) são campos do tipo Chave Estrangeira. Para detalhes sobre os conceitos de Chave Primária e Chave Estrangeira, consulte o Anexo II.

Neste capítulo vamos estudar diversas classes dos namespaces da lista anterior. Veremos como estabelecer uma conexão com um banco de dados, como retornar dados a partir desta conexão e como exibir estes dados em uma página ASP.NET.

No ASP 3.0, utilizando ADO, o objeto que utilizamos para retornar dados é o RecordSet. No ADO.NET não temos o objeto RecordSet. De início você que já estava acostumado com o objeto RecordSet pode estranhar um pouco, mas conforme veremos nos exemplos deste capítulo, os objetos do ADO.NET além de mais poderosos são também mais fáceis de utilizar. A principal facilidade é notada no momento de exibir os resultados obtidos, quando podemos utilizar alguns Web Server Controls bastante poderosos, mais especificamente o controle DataGrid.

### Informando que Você Deseja Utilizar Classes de um Determinado Namespace

Você deve ter notado, nos exemplos dos capítulos anteriores, que utilizamos uma série de classes e estruturas do namespace System; todavia não fizemos nenhuma referência a este namespace, no código das páginas ASP.NET de exemplo. Como então uma página é capaz de acessar os métodos de um namespace, sem ter feito referência ao mesmo? Isso somente acontece com os namespaces básicos, como é o caso do namespace System. Já com os namespaces que contêm as classes para acesso a dados, a história é diferente. Para que possamos utilizar classes de um destes namespaces precisamos, explicitamente, fazer referência aos mesmos, no início da página ASP.NET. O que eu chamei de fazer referência é chamado pelo Framework .NET de “importar” um namespace.

Temos duas maneiras diferentes para “importar” (ou fazer referência, como preferirem) um namespace, em uma página ASP.NET.

- Utilizando a diretiva @Import, no início da página. No fragmento de código a seguir, estamos importando os namespaces System.Data, System.Data.SqlClient e System.OleDb:

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

- Utilizando o comando using do C#. No fragmento de código a seguir, estamos importando os namespaces System.Data e System.Data.SqlClient:

```
using System.Data;
using System.Data.SqlClient;
```

Uma vez feitas as devidas importações (ou referências), estamos aptos a utilizar as classes dos namespaces referenciados.

## Classe ou Objeto; Objeto ou Classe?

Muitas vezes, os termos classe e objeto são utilizados, indiscriminadamente, com o mesmo sentido. Vamos fazer uma definição formal para estes termos, através de um exemplo:

No namespace System.Data existe uma classe chamada DataSet. Quando declaramos e inicializamos uma variável, como sendo do tipo DataSet, estamos criando um objeto baseado na classe DataSet. No trecho de código a seguir, temos um exemplo onde criamos um objeto chamado MeusDados, o qual é baseado na classe DataSet:

```
DataSet MeusDados = new DataSet();
```

## Estabelecendo Conexões

Uma coisa não mudou no ADO.NET, em relação ao ADO: o primeiro passo é estabelecer uma conexão com o banco de dados. Embora com ADO.NET tenhamos um modelo desconectado, conforme descrito anteriormente, o primeiro passo é fazer uma conexão com o banco de dados. Uma vez estabelecida a conexão, obtemos os dados desejados e podemos trabalhar com estes dados diretamente no navegador, desconectados do banco de dados. Uma vez feitas as alterações necessárias, sincronizamos os dados com o banco de dados.

Para estabelecer uma conexão com um banco de dados do SQL Server, devemos utilizar, preferencialmente, a classe SqlConnection, do namespace System.Data.SqlClient.

Para acessar um banco de dados utilizando o OLE-DB Provider correspondente, utilizamos a classe OleDbConnection do namespace System.Data.OleDb. Como existe um OLE-DB Provider para o SQL Server, também poderíamos utilizar a classe OleDbConnection para fazer uma conexão com o SQL Server, porém a classe SqlConnection é bem mais eficiente, em termos de desempenho, pois foi especificamente projetada para trabalhar com o SQL Server 2000, assim como foram todas as classes do namespace System.Data.SqlClient.

Vamos estudar estas duas classes.

## Estabelecendo uma Conexão com o SQL Server 2000 – SqlConnection

A classe SqlConnection faz parte do namespace System.Data.SqlClient. Esta classe é utilizada para estabelecer uma conexão com um servidor SQL Server. Diferente do que acontecia com ADO, não podemos executar um comando SQL, utilizando a classe SqlConnection. Ainda neste capítulo estudaremos as classes utilizadas para executar comandos em um banco de dados.

A seguir temos um exemplo de criação de um objeto do tipo SqlConnection:

```
SqlConnection MinhaConexão = new SqlConnection("server=SERVIDOR\\NETSDK;" +
 " uid=sa;pwd=;database=pubs");
```

O parâmetro passado é uma string que contém as informações necessárias para estabelecer a conexão com o banco de dados. Mais adiante veremos um exemplo de utilização e criação de uma conexão.

Na Tabela 10.1 temos uma descrição das principais propriedades da classe SqlConnection:

**Tabela 10.1 Principais propriedades da classe SqlConnection.**

Propriedade	Descrição
ConnectionString	É utilizada para definir ou retornar uma string de texto onde são informados os diversos parâmetros para estabelecer a conexão, como por exemplo o nome do servidor, a instância, o nome do banco de dados, o nome de login e senha.
ConnectionTimeOut	Define por quanto tempo é feita a tentativa de estabelecer a conexão. Após o tempo definido nesta propriedade, a tentativa é cancelada e um erro é gerado.
Database	É utilizada para definir ou retornar o nome do banco de dados que será utilizado quando a conexão for estabelecida.
DataSource	Retorna o nome da instância do SQL Server com a qual foi estabelecida uma conexão.
ServerVersion	Retorna uma string informando a versão do servidor SQL Server com o qual a conexão foi estabelecida.
State	Retorna o estado atual da conexão.

Vamos apresentar um pequeno exemplo, onde criamos uma conexão com o Banco de dados pubs da instância SERVIDOR\NETSDK (conforme descrito anteriormente). Uma vez estabelecida a conexão, vamos exibir as propriedades desta conexão em um Web Server Control do tipo TextArea. Vamos utilizar o evento Page\_Load, da página ASP.NET, para estabelecer a conexão com o Banco de dados pubs.

Na Listagem 10.1 temos o código para o exemplo proposto.

**Listagem 10.1 – A classe SqlConnection – chap10ex1.aspx.**

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

```

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

protected void Page_Load(Object Src, EventArgs E)
{

 // Crio uma conexão com o banco de dados pubs localizado no servidor local.
 // Vamos acessar a instância SERVIDOR\NETSDK.

 SqlConnection MinhaConexão = new
 SqlConnection("server=SERVIDOR\\NETSDK;uid=sa;pwd=;database=pubs");

 // Declaro uma variável do tipo String: auxPropriedades.
 // A variável auxPropriedades irá conter o valor das propriedades
 // da conexão minhaConexão.

 String auxPropriedades;
 auxPropriedades = "Propriedades da conexão:";

 auxPropriedades = auxPropriedades + "\n\n" + "ConnectionString: " +
 MinhaConexão.ConnectionString.ToString();

 auxPropriedades = auxPropriedades + "\n\n" + "Database: " +
 MinhaConexão.Database.ToString();

 auxPropriedades = auxPropriedades + "\n\n" + "DataSource: " +
 MinhaConexão.DataSource.ToString();

 auxPropriedades = auxPropriedades + "\n\n" + "State: " +
 MinhaConexão.State.ToString();

 ExibePropriedades.Font.Bold=true;
 ExibePropriedades.Text=auxPropriedades;

}

</script>

<body>

<h3>Classe SqlConnection!!!</h3>

```

```

<asp:TextBox
 runat=server
 id="ExibePropriedades"
 Text=""
 Rows="10"
 Cols="70"
 Font_Face="Arial" Font_Size="3"
 BackColor="lightblue"
 TextMode="MultiLine"
/>

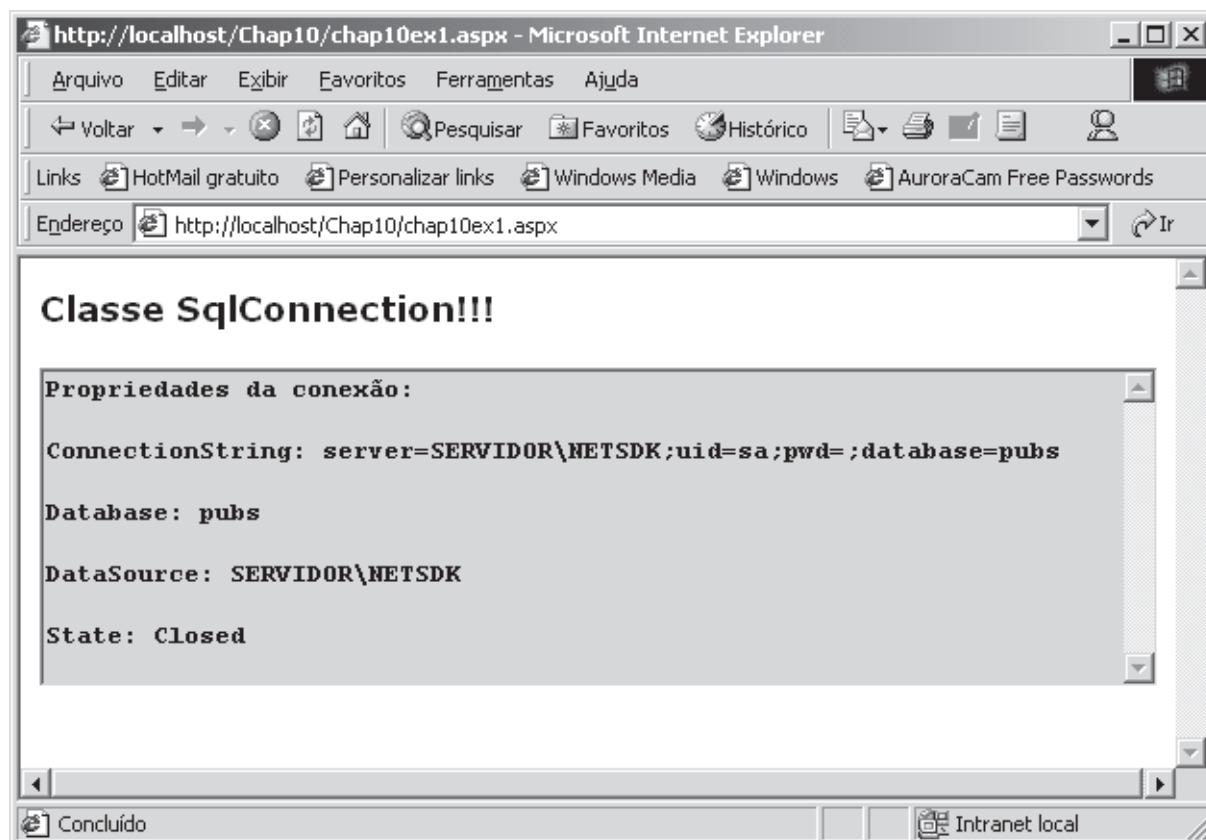
</body>
</html>

```

Digite o código da Listagem 10.1 e salve o mesmo em um arquivo chamado chap10ex1.aspx, na pasta chap10, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap10/chap10ex1.aspx>

Ao carregar a página você irá obter uma página semelhante à página indicada na Figura 10.5.



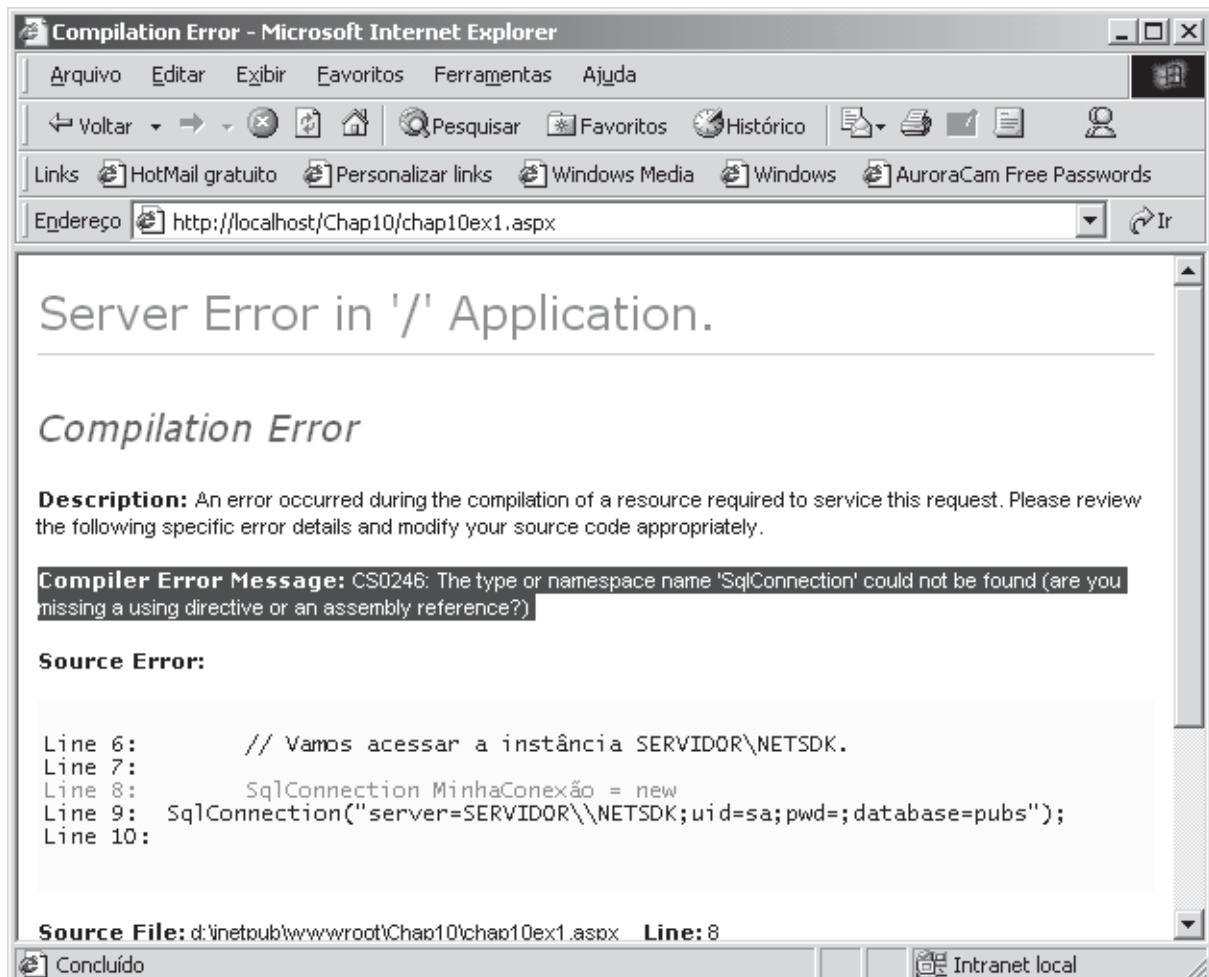
**Figura 10.5: A classe SqlConnection.**

Comentários sobre o código do exemplo – Chap10Ex1.aspx.

- ◆ Observe que a primeira coisa que fizemos foi referenciar os namespaces System.Data e System.Data.SqlClient:

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

Se não fizéssemos essa referência, obteríamos um erro de compilação, ao carregar a página, conforme indicado na Figura 10.6.



**Figura 10.6: Erro por não referenciar os namespaces necessários.**

Observe o texto em destaque, onde é dito que não foi possível encontrar SqlConnection. Isto acontece porque esta classe faz parte do namespace System.Data.SqlClient, o qual não foi referenciado na página ASP.NET.

- ◆ Em seguida criamos um variável MinhaConexão, baseada na classe SqlConnection:

```
SqlConnection MinhaConexão = new
SqlConnection("server=SERVIDOR\\NETSDK;uid=sa;pwd=;database=pubs");
```

Observe que este é um comando único, que foi dividido em duas linhas por falta de espaço. Declaramos a variável MinhaConexão como sendo do tipo SqlConnection, ao mesmo tempo que inicializamos esta variável, passando como parâmetro uma string de conexão. Na string passada como parâmetro são definidas as seguintes informações:

**Nome da instância do SQL Server: SERVIDOR\NETSDK.**

**Nome do usuário: uid=sa.**

**Senha: pwd= , neste caso significa senha em branco.**

**Banco de dados para fazer a conexão: database=pubs.**

É importante salientar que o objeto do tipo SqlConnection foi criado, porém ainda não foi estabelecida a conexão, conforme pode ser confirmado pelo valor da propriedade State=Closed, na Figura 10.5.

- ◆ No restante do evento Load, montamos uma string (auxPropriedades), onde vamos concatenando o nome e o valor das propriedades do objeto MinhaConexão. Observe que anexamos dois caracteres de nova linha – \n\n. Isso é feito para ir para uma nova linha a cada propriedade (um \n) e para deixar uma linha em branco entre a exibição de cada propriedade (mais um \n).

Após termos montado a string auxPropriedades, definimos a fonte do controle ExibePropriedades para negrito e atribuímos o valor da variável auxPropriedades, à propriedade Text do controle ExibePropriedades, conforme indicado no fragmento a seguir:

```
ExibePropriedades.Font.Bold=true;
ExibePropriedades.Text=auxPropriedades;
```

- ◆ O controle ExibePropriedades é um Web Server Control do tipo TextBox com múltiplas linhas. Para maiores informações sobre este controle, consulte o Capítulo 9.

Na Tabela 10.2 temos uma descrição dos principais métodos da classe SqlConnection:

**Tabela 10.2 Principais métodos da classe SqlConnection.**

Método	Descrição
<b>Open</b>	<b>Abre a conexão de acordo com as definições da propriedade ConnectionString.</b>
<b>Close</b>	<b>Fecha a conexão com o banco de dados.</b>
<b>ChangeDatabase</b>	<b>Altera o banco de dados associado com a conexão.</b>

O principal evento do objeto SqlConnection é o evento StateChange. Este evento ocorre quando o estado da conexão é alterado de Open para Closed ou vice-versa.

No exemplo da Listagem 10.1, podemos adicionar o seguinte comando, logo após a criação da conexão MinhaConexão:

```
MinhaConexão.Open();
```

Este comando abre a conexão MinhaConexão. Após inserirmos este comando na Listagem 10.1 e recarregarmos a página, obteremos o resultado indicado na Figura 10.7.

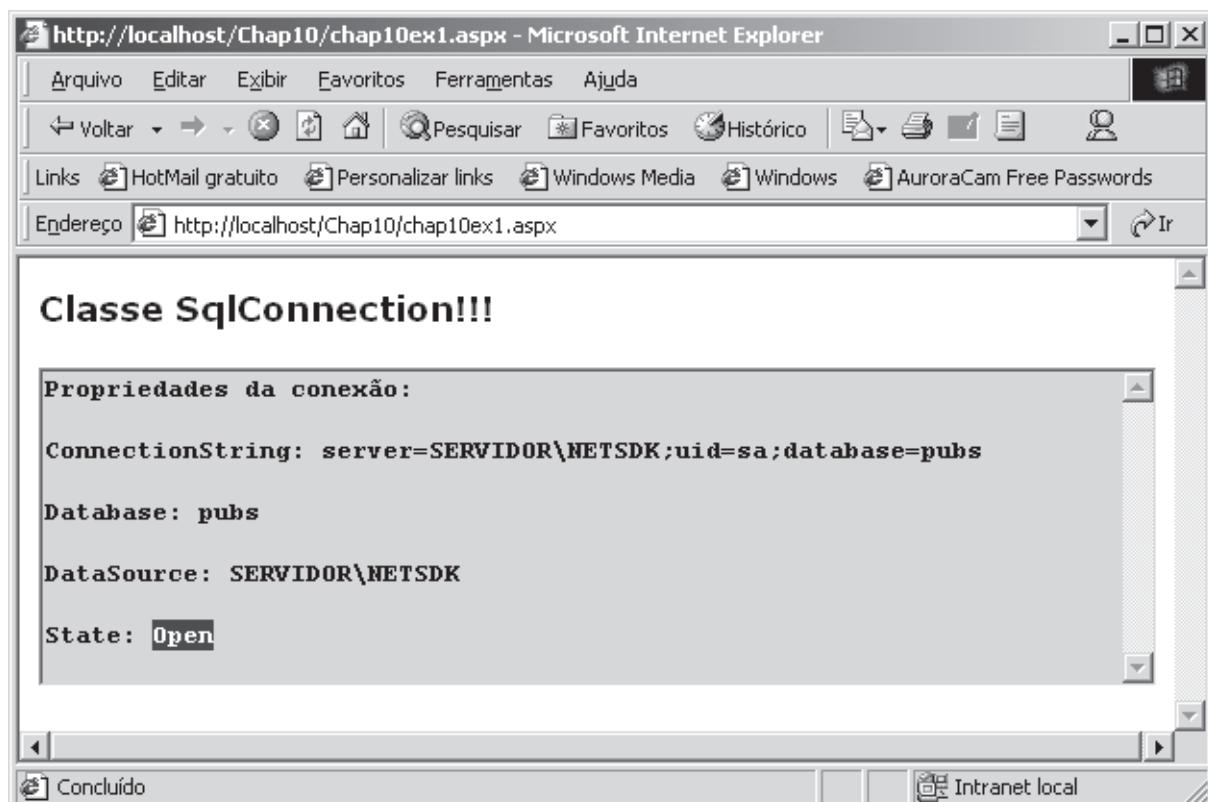


Figura 10.7: Utilizando o método Open da classe SqlConnection.

## Estabelecendo uma Conexão com o Microsoft Access – OleDbConnection

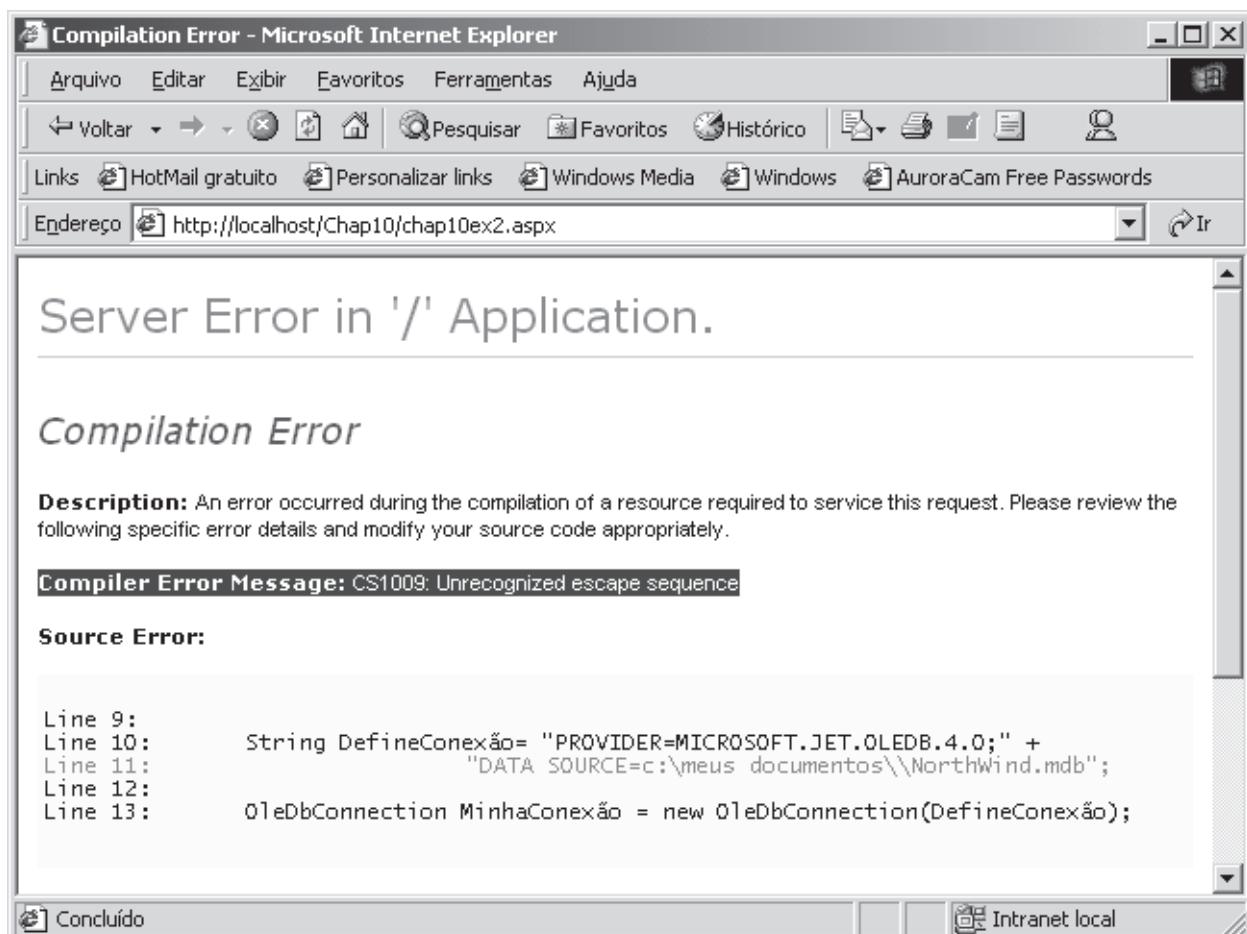
A classe OleDbConnection faz parte do namespace System.Data.OleDb. Esta classe é utilizada para estabelecer uma conexão com uma fonte de dados, para a qual exista um OLE-DB Provider. Diferente do que acontecia com ADO, não podemos executar um comando SQL, utilizando a classe OleDbConnection; ao invés disso devemos utilizar um objeto Command. Ainda neste capítulo estudaremos as classes utilizadas para executar comandos em um banco de dados.

A seguir temos um exemplo de criação de um objeto do tipo OleDbConnection:

```
String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
 "DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";

OleDbConnection MinhaConexão = new OleDbConnection(DefineConexão);
```

Primeiro definimos uma string chamada DefineConexão. Esta string contém as informações necessárias para estabelecer a conexão com um banco de dados do Microsoft Access. É importante salientar a utilização das duas barras invertidas (\\\), ao invés de uma única barra. Como a barra invertida é utilizada para caracteres de escape, no C#, como por exemplo: \n para quebra de linha, quando queremos representar uma barra e não um caractere de escape, precisamos colocar duas barras. Se você colocar somente uma barra, irá obter o erro indicado na Figura 10.8, quando tentar carregar a página:



**Figura 10.8: Ero por usarmos um única barra (\) ao invés de duas barras (\).\.**

Observe a mensagem em destaque, na Figura 10.8, informando que temos um caractere de escape, inválido. Em seguida criamos um objeto do tipo OleDbConnection, chamado MinhaConexão e passamos a string DefineConexão como parâmetro para o método construtor do objeto MinhaConexão.

Na Tabela 10.3 temos os componentes que podem fazer parte da string de conexão (ConnectionString), para o OLE-DB Provider do Microsoft Access.

**Tabela 10.3 Definindo a propriedade ConnectionString para uma fonte do Microsoft Access.**

Propriedade	Descrição
Provider	Deve ser especificado o OLE DB Provider para o Microsoft Access. Para esta propriedade utilizamos o seguinte valor: Microsoft.Jet.OLEDB.4.0.
Data Source	Informamos o caminho para o arquivo .mdb. Por exemplo, C:\\Arquivos de programas\\NorthWind.mdb.
User ID	Especifica o nome do usuário com a qual a conexão será estabelecida. Caso esta propriedade não seja informada, a mesma será definida como “admin”, o qual é o usuário padrão para o Microsoft Access.

Propriedade	Descrição
Password	Informa a senha para o usuário que fará a conexão. Caso esta propriedade não seja informada, a mesma será definida como “ ”, ou seja, senha em branco.

Na Tabela 10.4 temos uma descrição das principais propriedades da classe OleDbConnection:

**Tabela 10.4 Principais propriedades da classe OleDbConnection.**

Propriedade	Descrição
ConnectionString	É utilizada para definir ou retornar uma string de texto onde são informados os diversos parâmetros para estabelecer a conexão, como por exemplo o nome do servidor, a instância, o nome do banco de dados, o nome de login e senha.
ConnectionTimeout	Define por quanto tempo é feita a tentativa de estabelecer a conexão. Após o tempo definido nesta propriedade, a tentativa é cancelada e um erro é gerado.
Database	É utilizada para definir ou retornar o nome do banco de dados que será utilizado quando a conexão for estabelecida. Utilizada, normalmente, para conexão com o SQL Server ou ORACLE.
State	Retorna o estado atual da conexão.

Vamos apresentar um pequeno exemplo, onde criamos uma conexão com o banco de dados C:\Meus documentos\NorthWind.mdb. Uma vez estabelecida a conexão, vamos exibir as propriedades desta conexão em um Web Server Control do tipo TextArea. Vamos utilizar o evento Page\_Load, da página ASP.NET, para estabelecer a conexão com o Banco de dados pubs.

Na Listagem 10.2 temos o código para o exemplo proposto.

**Listagem 10.2 – A classe OleDbConnection – chap10ex2.aspx.**

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>

<html>

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

 protected void Page_Load(Object Src, EventArgs E)
 {
 // Crio uma conexão com o banco de dados pubs localizado no servidor local.
 // Vamos acessar a instância SERVIDOR\NETSDK.
```

```
String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
 "DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";

OleDbConnection MinhaConexão = new OleDbConnection(DefineConexão);

MinhaConexão.Open();

// Declaro uma variável do tipo String: auxPropriedades.
// A variável auxPropriedades irá conter o valor das propriedades
// da conexão minhaConexão.

String auxPropriedades;
auxPropriedades = "Propriedades da conexão:";

auxPropriedades = auxPropriedades + "\\n\\n" + "ConnectionString: " +
 MinhaConexão.ConnectionString.ToString();

auxPropriedades = auxPropriedades + "\\n\\n" + "Database: " +
 MinhaConexão.Database.ToString();

auxPropriedades = auxPropriedades + "\\n\\n" + "DataSource: " +
 MinhaConexão.DataSource.ToString();

auxPropriedades = auxPropriedades + "\\n\\n" + "State: " +
 MinhaConexão.State.ToString();

ExibePropriedades.Font.Bold=true;
ExibePropriedades.Text=auxPropriedades;

}

</script>

<body>
 <h3>Classe OleDbConnection!!!</h3>
```

```

<asp:TextBox
 runat=server
 id="ExibePropriedades"
 Text=""
 Rows="10"
 Cols="70"
 Font_Face="Arial"
 Font_Size="3"
 BackColor="lightblue"
 TextMode="MultiLine"
/>

```

</body>

</html>

Digite o código da Listagem 10.2 e salve o mesmo em um arquivo chamado chap10ex2.aspx, na pasta chap10, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap10/chap10ex2.aspx>

Você irá obter uma página semelhante à página indicada na Figura 10.9.

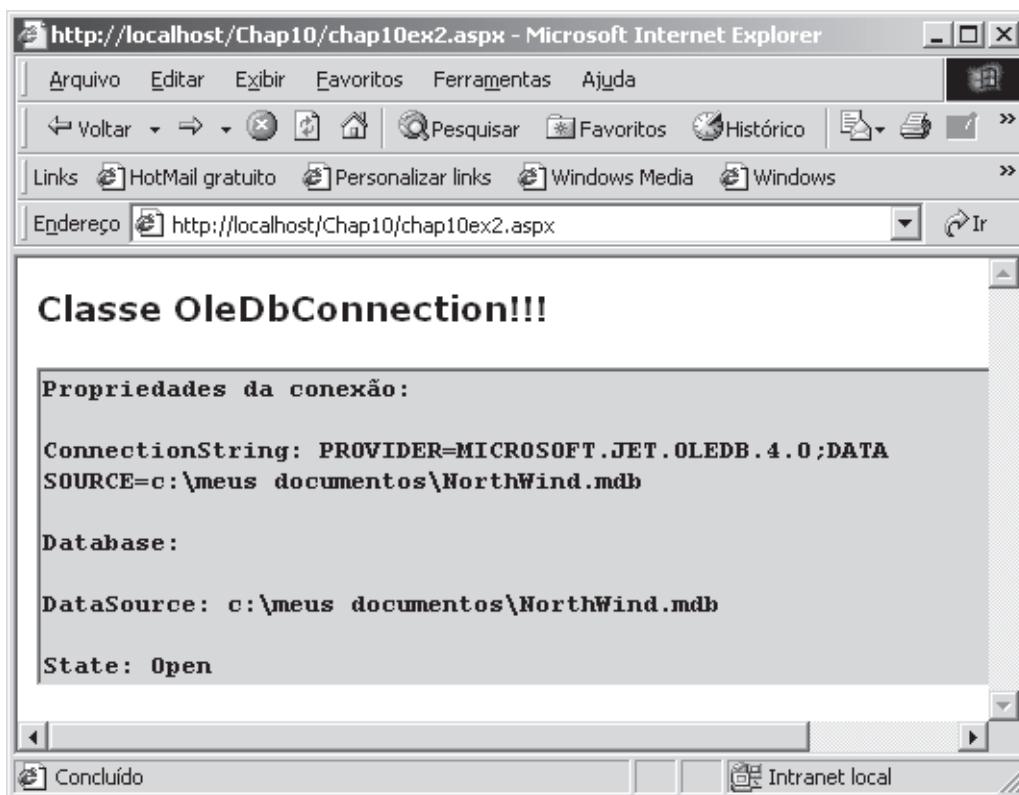


Figura 10.9: Propriedades da classe OleDbConnection.

Comentários sobre o código do exemplo – Chap10Ex2.aspx.

Os mesmos comentários feitos para o exemplo da Listagem 10.1.

Na Tabela 10.5 temos uma descrição dos principais métodos da classe OleDbConnection:

**Tabela 10.5 Principais métodos da classe OleDbConnection.**

Método	Descrição
Open	Abre a conexão de acordo com as definições da propriedade ConnectionString.
Close	Fechá a conexão com o banco de dados.
ChangeDatabase	Altera o banco de dados associado com a conexão. Utilizado para conexões com o SQL Server ou ORACLE.

O principal evento do objeto OleDbConnection é o evento StateChange. Este evento ocorre quando o estado da conexão é alterado de Open para Closed ou vice-versa.

Muito bem, já sabemos estabelecer uma conexão com um banco de dados. E agora? Ainda precisamos aprender como fazer as seguintes operações:

- ◆ Definir um comando para ser executado no banco de dados.
- ◆ Retornar dados de uma ou mais tabelas.
- ◆ Retornar informações sobre os relacionamentos entre as tabelas.
- ◆ Exibir os resultados obtidos.

Nos próximos tópicos aprenderemos a realizar estas operações.

## Uma Visão Geral do Processo de Acesso a Dados

Existem muitas maneiras de estabelecer uma conexão com uma fonte de dados, retornar um conjunto de dados e exibir estes dados em uma página ASP.NET.

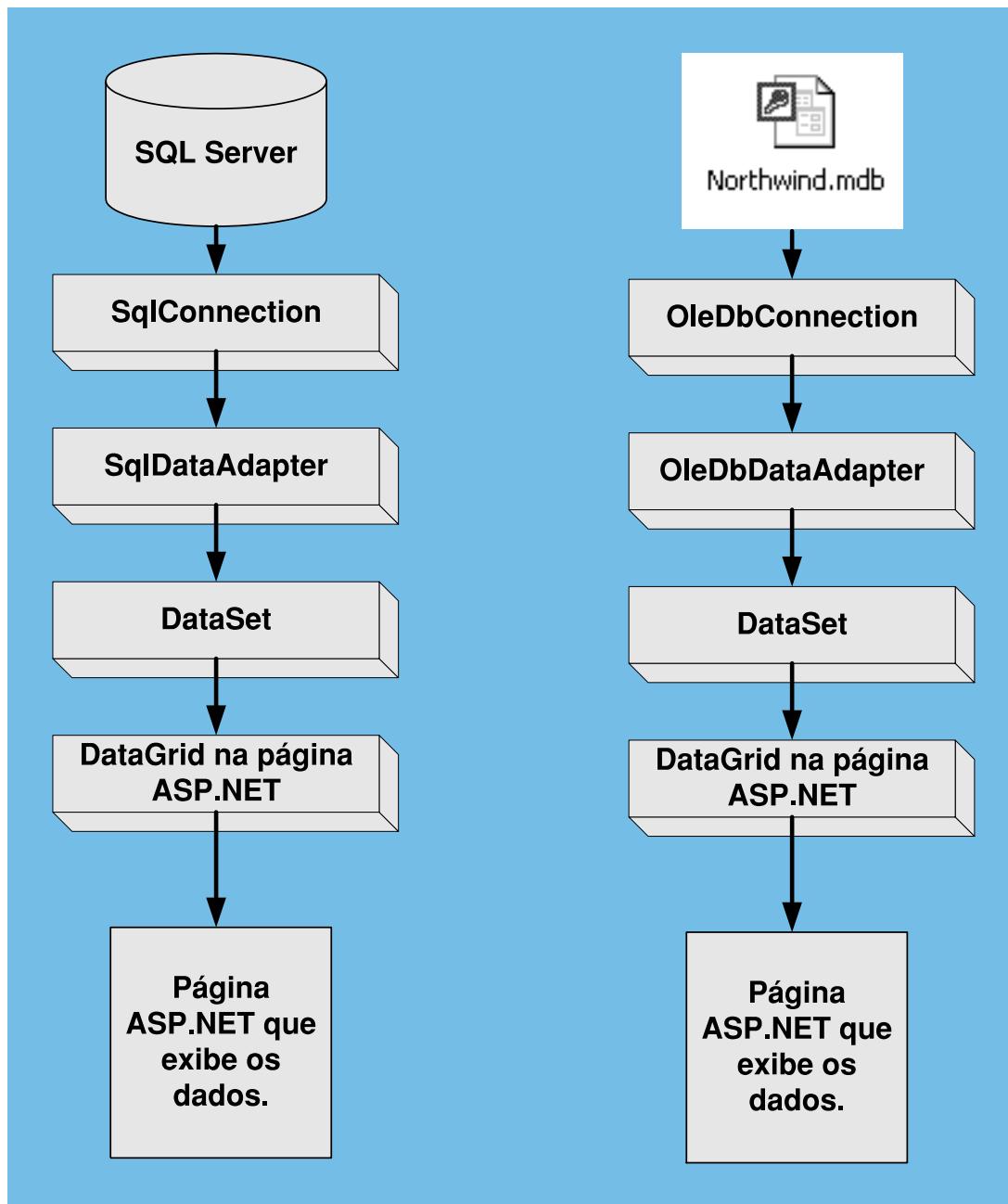
Neste capítulo estaremos utilizando a seguinte abordagem:

- ◆ Criar uma conexão com uma fonte de dados utilizando SqlConnection ou OleDbConnection.
- ◆ Definir um comando SQL a ser executado no banco de dados. O comando é definido utilizando um objeto do tipo SqlDataAdapter ou OleDbDataAdapter. O comando SQL define os dados que serão retornados a partir do banco de dados. Também podemos utilizar um objeto SqlCommand ou OleDbCommand. Uma opção é criar um objeto Command e depois a propriedade SelectCommand do objeto DataAdapter como sendo igual ao objeto Command. Outra opção é passar o comando SQL como parâmetro do objeto DataAdapter, o que faz

com que não seja necessária a criação explícita de um objeto `Command`. O objeto `DataAdapter` faz a ligação de um ou mais objetos do tipo `Command` com um objeto do tipo `DataSet`. Por exemplo, podemos ligar vários objetos `Command` com um único `DataSet`. Iremos detalhar todos estes aspectos nos exemplos deste capítulo.

- ◆ Utilizamos os dados retornados pelo(s) comando(s) SQL para preencher um objeto do tipo `DataSet`.
- ◆ Vamos utilizar um Web Server Control – `Datagrid`, para exibir os dados associados ao objeto do tipo `DataSet`.

Na Figura 10.10 temos uma visão geral deste processo.



**Figura 10.10: Passos para acessar e exibir dados em uma página ASP.NET.**

É importante salientar que esta é apenas uma das maneiras de acessarmos dados e exibi-los em uma página ASP.NET. Nos Capítulos 11 e 12 veremos outras maneiras de ter acesso a variadas fontes de dados. No restante deste capítulo vamos estudar os objetos indicados no diagrama da Figura 10.10.

# Criando Objetos Command

Para criar objetos Command temos duas opções:

- ◆ **SqlCommand:** Este objeto é utilizado para definir um comando que será executado através de uma conexão do tipo SqlConnection.
- ◆ **OleDbCommand:** Utilizado para definir um comando que será executado através de uma conexão do tipo OleDbConnection.

Conforme descrevemos no tópico anterior, podemos criar um objeto Command, no qual definimos um comando SQL a ser executado através de uma conexão. As propriedades dos objetos SqlCommand e OleDbCommand são semelhantes.

Na Tabela 10.6 temos uma descrição das principais propriedades das classes SqlCommand/OleDbCommand:

**Tabela 10.6 Principais propriedades das classes SqlCommand/OleDbCommand.**

Propriedade	Descrição
CommandText	Esta propriedade é utilizada para definir ou retornar o comando SQL ou o nome de um Stored Procedure associado com o comando.
CommandTimeOut	Define por quanto tempo o Framework .NET tenta executar o comando. Se no tempo definido por esta propriedade, o comando não for executado com sucesso, o mesmo é suspenso e uma mensagem de erro é retornada.
CommandType	Pode ser definida como Text, que é o valor padrão e significa que o valor atribuído à propriedade CommandText é um comando SQL. Também pode assumir o valor StoredProcedure; neste caso significa que o valor atribuído à propriedade CommandText é o nome de um Stored Procedure.
Connection	Esta propriedade é utilizada para definir ou retornar a conexão através da qual o comando é executado.

Na Tabela 10.7 temos uma descrição dos principais métodos das classes SqlCommand/OleDbCommand:

**Tabela 10.7 Principais métodos das classes SqlCommand/OleDbCommand.**

Método	Descrição
Cancel	Cancela a execução do comando.
CreateParameter	Utilizada para a criação e definição de parâmetros. Podemos utilizar parâmetros quando estamos executando um Stored Procedure ou uma consulta parametrizada do Microsoft Access.
ExecuteNonQuery	Executa o comando definido na propriedade CommandText, através da conexão definida na propriedade Connection, para consultas que não retornam dados. Exemplo de consultas que não retornam dados são consultas de atualização, adição ou exclusão. Este método retorna um valor inteiro, valor este que indica o número de registros afetados pela execução do comando.

Vamos apresentar um pequeno exemplo, onde criamos uma conexão com o banco de dados C:\Meus documentos\NorthWind.mdb. Uma vez estabelecida a conexão vamos criar um objeto OleDbCommand associado com esta conexão e vamos exibir as propriedades do objeto OleDbCommand em um Web Server Control do tipo TextArea. Vamos utilizar o evento Page\_Load, da página ASP.NET, para estabelecer a conexão com o Banco de dados pubs do SQL Server.

Na Listagem 10.3 temos o código para o exemplo proposto.

**Listagem 10.3 – A classe OleDbCommand – chap10ex3.aspx.**

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>

<html>

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

protected void Page_Load(Object Src, EventArgs E)
{
 // Crio uma conexão com o banco de dados Northwind.mdb
 // localizado na pasta C:\Meus documentos.

 String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
 "DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";

 OleDbConnection MinhaConexão = new OleDbConnection(DefineConexão);

 // Agora crio um objeto OleDbCommand chamado MeuComando.
 // Primeiro defino o texto do comando em uma variável TextoDoComando.
 // Em seguida crio um objeto do tipo OleDbCommand e passo esta
 // variável como parâmetro.

 string TextoDoComando = "SELECT NúmeroDoPedido, CódigoDoCliente,"
 + "PaísDeDestino FROM Pedidos";

 OleDbCommand MeuComando = new OleDbCommand(TextoDoComando,MinhaConexão);

 MinhaConexão.Open();
}
```

```
// Declaro uma variável do tipo String: auxPropriedades.
// A variável auxPropriedades irá conter o valor das propriedades
// do comando MeuComando.

String auxPropriedades;
auxPropriedades = "Propriedades do objeto OleDbCommand:";

auxPropriedades = auxPropriedades + "\n\n" + "CommandType: " +
 MeuComando.CommandType.ToString();

auxPropriedades = auxPropriedades + "\n\n" + "CommandText: " +
 MeuComando.CommandText.ToString();

auxPropriedades = auxPropriedades + "\n\n" + "Timeout: " +
 MeuComando.CommandTimeout.ToString();

auxPropriedades = auxPropriedades + "\n\n" + "Connection: " +
 MeuComando.Connection.ToString();

ExibePropriedades.Font.Bold=true;
ExibePropriedades.Text=auxPropriedades;

}

</script>

<body>

<h3>Classe OleDbCommand!!!</h3>

<asp:TextBox
 runat=server
 id="ExibePropriedades"
 Text=""
 Rows="10"
 Cols="70"
```

```

Font_Face="Arial"
Font_Size="3"
BackColor="lightblue"
TextMode="MultiLine"
/>

```

</body>

</html>

Digite o código da Listagem 10.3 e salve o mesmo em um arquivo chamado chap10ex3.aspx, na pasta chap10, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap10/chap10ex3.aspx>

Você irá obter uma página semelhante à página indicada na Figura 10.11.

Comentários sobre o código do exemplo – Chap10Ex3.aspx.

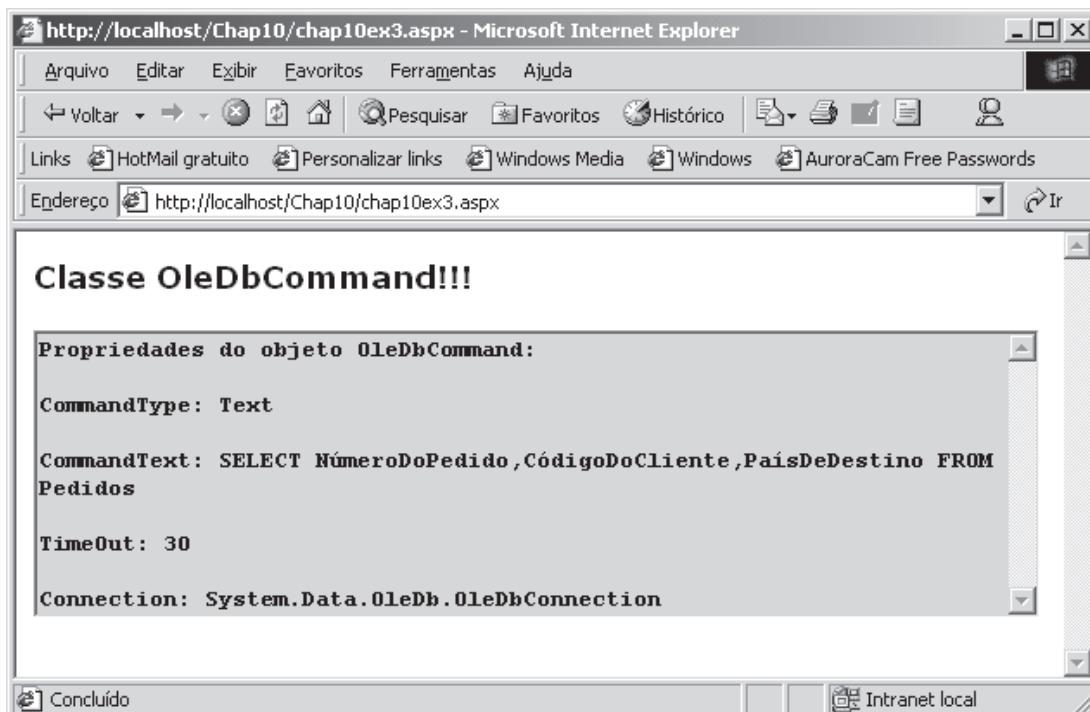
- ◆ Neste exemplo criamos um objeto do tipo OleDbCommand, conforme indicado no fragmento de código a seguir:

```

string TextoDoComando = "SELECT NúmeroDoPedido, CódigoDoCliente,"
+ "PaísDeDestino FROM Pedidos";

```

```
OleDbCommand MeuComando = new OleDbCommand(TextoDoComando, MinhaConexão);
```



**Figura 10.11: Propriedades da classe OleDbCommand.**

Observe que primeiro criamos uma string TextoDoComando, a qual atribuímos o comando SQL que será associado com o objeto OleDbCommand. Em seguida criamos o objeto MeuComando e passamos como parâmetros, para o método construtor do objeto, primeiro a string TextoDoComando e em seguida o nome da conexão, através da qual o comando deve ser executado.

É importante salientar que, neste momento, o comando ainda não foi executado e, portanto, ainda não foram retornados dados.

- ◆ Em seguida começamos a concatenar os valores das propriedades do objeto MeuComando em uma string auxPropriedades. No final, o valor desta string é exibido em um controle do tipo TextArea.

- ◆ Ao invés dos objetos OleDbConnection e OleDbCommand, poderíamos utilizar os objetos SqlConnection e SqlCommand para conectar com o SQL Server, conforme indicado no código a seguir:

```
SqlConnection MinhaConexão = new
SqlConnection("server=SERVIDOR\\NETSDK;uid=sa;pwd=;database=pubs");

string TextoDoComando = "SELECT au_id, au_lname, au_fname FROM authors";
OleDbCommand MeuComando = new OleDbCommand(TextoDoComando, MinhaConexão);

MinhaConexão.Open();
```

Apenas definir um objeto OleDbCommand ou SqlCommand não faz com que dados sejam retornados através da conexão definida. Quando os dados são retornados, os mesmos são armazenados em um objeto do tipo DataSet. Mas precisamos de uma maneira de “ligar” o objeto Command com o objeto DataSet, através dos seguintes passos:

- ◆ O comando definido na propriedade CommandText, do objeto Command é executado.
- ◆ Os dados retornados são ligados a um objeto do tipo DataSet.

O objeto que faz esta ligação é o DataAdapter. Vamos estudar este objeto em detalhes. O objeto DataAdapter (SqlDataAdapter ou OleDbDataAdapter) trabalha em conjunto com o objeto DataSet. Nos exemplos do próximo tópico veremos a utilização do objeto DataAdapter e apresentaremos apenas alguns aspectos básicos do objeto DataSet. No tópico seguinte iremos detalhar o objeto DataSet. Também estaremos utilizando alguns elementos básicos do objeto DataGrid, o qual será explicado em detalhes no final do Capítulo.

## Retornando Dados com DataAdapter

Para criar objetos DataAdapter temos duas opções:

- ◆ **SqlDataAdapter:** Este objeto é utilizado para executar um ou mais comandos ou Stored Procedures, em um banco de dados do SQL Server e associar os resultados obtidos com um objeto do tipo DataSet.
- ◆ **OleDbDataAdapter:** Utilizado para executar um ou mais comandos em uma fonte de dados, utilizando o OLE-DB Provider respectivo, e associar os resultados obtidos com um objeto do tipo DataSet.

Os objetos SqlDataAdapter/OleDbDataAdapter funcionam como uma ponte entre uma fonte de dados e o objeto DataSet, tanto para acesso quanto para alterações nos dados. Esta ponte pode ser estabelecida de duas maneiras:

1. Através da utilização do método Fill do objeto DataAdapter, para retornar dados de uma fonte de dados e colocar estes dados em um objeto DataSet.
2. Através da utilização do método Update do objeto DataAdapter, método este que sincroniza os dados da fonte de dados original, com as modificações feitas nos dados do objeto DataSet.

Na Tabela 10.8 temos uma descrição das principais propriedades das classes SqlDataAdapter/OleDbDataAdapter:

**Tabela 10.8 Principais propriedades das classes SqlDataAdapter/OleDbDataAdapter.**

Propriedade	Descrição
DeleteCommand	Utilizada para definir ou retornar um comando SQL para exclusão de dados, normalmente um comando DELETE.
InsertCommand	Utilizada para definir ou retornar um comando SQL para inserção de novos dados, normalmente um comando INSERT.
SelectCommand	Utilizada para definir ou retornar um comando SQL, utilizado para retornar dados, normalmente um comando SELECT. Também podemos atribuir, a esta propriedade, um objeto Command previamente criado.
UpdateCommand	Utilizada para definir ou retornar um comando SQL que atualiza dados. Normalmente um comando UPDATE.

Na Tabela 10.9 temos uma descrição dos principais métodos das classes SqlDataAdapter/OleDbDataAdapter:

**Tabela 10.9 Principais métodos das classes SqlDataAdapter/OleDbDataAdapter:**

Método	Descrição
Fill	Com certeza é o método mais utilizado. Este método executa o comando definido na propriedade SelectCommand. Os dados retornados pela execução do comando definido na propriedade SelectCommand são associados com um objeto do tipo DataSet.
FillSchema	Este método retorna uma tabela em branco, isto é, com zero registro, porém com a mesma estrutura da tabela original. Na prática o que este método faz é copiar a estrutura de uma tabela.
Update	Com ASP.NET trabalhamos com um modelo de dados desconectados, conforme descrito no início deste capítulo. Uma vez retornados os dados em um objeto do tipo DataSet, a conexão com o banco de dados é fechada. Alterações podem ser feitas nos dados desconectados, porém estas alterações precisam ser enviadas para o banco de dados, quer seja o SQL Server, quer seja um arquivo .mdb do Microsoft Access. O método Update é utilizado para enviar estas alterações para o banco de dados. O método executa os comandos InsertCommand, UpdateCommand e DeleteCommand para cada inserção, atualização ou exclusão, feitas nos dados desconectados, de tal forma que estas alterações sejam enviadas para a fonte de dados original. Em poucas palavras: sincroniza os dados do objeto DataSet com o conjunto de dados originais.

Na Tabela 10.10 temos uma descrição dos principais eventos das classes SqlDataAdapter/OleDbDataAdapter:

**Tabela 10.10 Principais eventos das classes SqlDataAdapter/OleDbDataAdapter:**

Evento	Descrição
FillError	Ocorre quando um erro é retornado durante a execução do método Fill.
RowUpdated	Ocorre durante uma atualização, após o respectivo comando ter sido executado na fonte de dados original.
RowUpdating	Ocorre durante uma atualização, antes do respectivo comando ter sido executado na fonte de dados original.

## O Objeto DataSet

Os objetos SqlDataAdapter/OleDbDataAdapter, descritos no tópico anterior, são utilizados para executar um comando SQL ou um Stored Procedure, em um banco de dados e retornar um ou mais conjuntos de dados. Precisamos de uma estrutura capaz de receber e manipular os dados retornados; esta estrutura é o objeto DataSet. O objeto DataSet é derivado da classe DataSet, do namespace System.Data.

O objeto DataAdapter executa um comando através de uma conexão e retorna os dados para um objeto DataSet. A conexão é desfeita, pois o objeto DataSet fornece as funcionalidades necessárias para acessarmos e manipularmos os dados, estando desconectados do servidor. Acessamos e alteramos os dados conforme necessário e depois as alterações efetuadas são sincronizadas com o servidor.

DataSet é, sem dúvida, o principal objeto do ADO.NET, assim como o objeto RecordSet é o principal objeto do ADO.

Um objeto DataSet é formado por uma coleção de objetos do tipo DataTable, os quais pertencem à coleção Tables. Para representar o relacionamento entre dois objetos do tipo DataTable, utilizamos um objeto do tipo DataRelation. Por exemplo, podemos criar um DataSet que contém duas tabelas: Pedidos e Clientes. A tabela Clientes relaciona-se com a tabela Pedidos, através de um relacionamento do tipo um para vários, ou seja, um cliente pode fazer vários pedidos. Podemos representar este relacionamento utilizando um objeto do tipo DataRelation.

Podemos garantir a integridade dos dados através da definição de campos do tipo Chave Primária – utilizando o objeto UniqueConstraint; e da definição de Chaves Estrangeiras – utilizando o objeto ForeignKeyConstraint.

O objeto DataSet lê e grava dados e a estrutura dos dados no formato de documentos XML, os quais podem ser enviados pela Internet via protocolo HTTP, o que facilita a troca de informações com sistemas de outras empresas, sistemas estes também habilitados ao XML.

**NOTA:** Para maiores detalhes sobre o modelo relacional de dados e relacionamentos entre tabelas, consulte o Anexo II. Neste anexo também são descritos os conceitos de Chave primária, Chave Estrangeira, normalização e integridade de dados.

Na Tabela 10.11 temos uma descrição das principais propriedades da classe DataSet:

**Tabela 10.11** Principais propriedades da classe DataSet.

Propriedade	Descrição
DataSetName	Utilizada para definir ou retornar o nome do DataSet.
EnforceConstraints	Pode ser utilizada para definir ou retornar um valor True ou False. Se o valor desta propriedade for True, as regras de integridade serão observadas em uma operação de atualização. Por exemplo, se alterarmos o código do cliente em um pedido da tabela pedidos para 01010 e não existir o cliente com o código 01010 na tabela Clientes, a operação não será realizada. Se o valor da propriedade EnforceConstraints for False, a operação descrita no nosso exemplo será realizada sem que nenhuma verificação seja feita.
Relations	Retorna uma coleção com todos os relacionamentos existentes entre as tabelas do DataSet.
Tables	Retorna uma coleção com todas as tabelas do DataSet.

Na Tabela 10.12 temos uma descrição dos principais métodos da classe DataSet:

**Tabela 10.12** Principais métodos da classe DataSet.

Método	Descrição
AcceptChanges	Torna definitivas todas as alterações feitas nas tabelas ou relacionamentos do DataSet, desde que o mesmo foi inicializado ou desde a última vez que o método AcceptChanges foi chamado.
Clear	Remove todos os dados do DataSet, zerando todas as linhas de todas as tabelas.
Clone	Faz uma cópia idêntica do DataSet, inclusive dos seus dados, para um outro objeto do tipo DataSet.
GetChanges	Retorna um objeto do tipo DataSet, contendo todas as alterações que foram feitas desde a inicialização do DataSet original, ou desde a última chamada do método AcceptChanges.
HasChanges	Retorna um valor do tipo Booleano. True indica que houve alterações nos dados do DataSet – adições, alterações ou exclusões. False indica que não houve alterações.

Neste momento já somos capazes de estabelecer uma conexão com o banco de dados, definir um objeto DataAdapter para executar um comando através da conexão estabelecida e preencher um objeto DataSet com os dados retornados. A próxima etapa é exibir os dados obtidos. A exibição dos dados é a única coisa que o usuário vê, ao acessar a

página. Existem maneiras variadas para exibir os dados em uma página ASP.NET. Neste capítulo estaremos utilizando o controle DataGrid. Em seguida apresentaremos um exemplo simples de conexão com um banco de dados do Microsoft Access. Neste exemplo vamos utilizar o controle DataGrid. Mais adiante, neste capítulo, iremos estudar este controle, em detalhes.

**IMPORTANTE:** Também utilizaremos um objeto do tipo **DataView** que será detalhado mais adiante.

Vamos conectar com o banco de dados C:\Meus documentos\NorthWind.mdb. Definiremos um comando SQL que retorna os seguintes campos da tabela Clientes:

- ◆ CódigoDoCliente
- ◆ NomeDaEmpresa
- ◆ Cidade
- ◆ País

Na Listagem 10.4 temos o código para o exemplo proposto.

#### Listagem 10.4 – Um exemplo completo – chap10ex4.aspx.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>

<html>

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

protected void Page_Load(Object Src, EventArgs E)
{
 // Crio uma conexão com o banco de dados pubs localizado no servidor local.
 // Vamos acessar a instância SERVIDOR\NETSDK.

 String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
 "DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";

 OleDbConnection MinhaConexão = new OleDbConnection(DefineConexão);

 // Utilizamos um objeto DataAdapter para executar um comando SQL,
 // o qual retorna todos os dados da tabela "Clientes".
```

```

OleDbDataAdapter MeuComando = new OleDbDataAdapter("SELECT CódigoDoCliente,"
+ "NomeDaEmpresa, Cidade, País FROM Clientes", MinhaConexão);

// Criamos e preenchemos um objeto DataSet.
// Observe que não temos mais o objeto Recordset,
// como era de praxe com o ASP 3.0.

DataSet ds = new DataSet();

// Utilizo o método Fill do objeto DataAdapter, para preencher
// o objeto DataSet, com os dados retornados pelo comando SQL.

MeuComando.Fill(ds);

// Conectamos um controle DataGrid com o DataSet criado anteriormente.
// MinhaGrade é o id (nome) de um controle do tipo
// DataGrid que está na seção de apresentação da página.

DataView source = new DataView(ds.Tables[0]);

MinhaGrade.DataSource = source ;
MinhaGrade.DataBind();
}

</script>

<body>

<%— Exibe as informações do DataGrid no corpo da página. —%>

<h3>Clientes da empresa North Wind!!!</h3>

<ASP:DataGrid id="MinhaGrade" runat="server"
Width="700"
BackColor="#ccccff"
BorderColor="black"

```

```
 ShowFooter="false"
 CellPadding=3
 CellSpacing="0"
 Font-Name="Verdana"
 Font-Size="8pt"
 HeaderStyle-BackColor="#aaaadd"
 MaintainState="false"
/>

```

```
</body>
</html>
```

Digite o código da Listagem 10.4 e salve o mesmo em um arquivo chamado chap10ex4.aspx, na pasta chap10, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap10/chap10ex4.aspx>

Você irá obter a página indicada na Figura 10.12.

Comentários sobre o código do exemplo – Chap10Ex4.aspx.

- ◆ A primeira coisa que fizemos foi importar os namespaces necessários – System.Data e System.Data.OleDb:

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>
```

- ◆ Utilizamos o evento Load da página para executar os comandos que estabelecem a conexão com o banco de dados e retornam dados da tabela Clientes – Page\_Load.
- ◆ Para estabelecer uma conexão com o banco de dados NorthWind.mdb, utilizamos um objeto OleDbCommand. Para maiores detalhes sobre este objeto consulte o tópico respectivo, no início deste capítulo.
- ◆ Uma vez definida a conexão, utilizamos um objeto OleDbDataAdapter, chamado MeuComando, para abrir a conexão e executar um comando SQL que retorna alguns campos da tabela Clientes:

```
OleDbDataAdapter MeuComando = new OleDbDataAdapter("SELECT CódigoDoCliente," +
"NomeDaEmpresa,País,Cidade FROM Clientes", MinhaConexão);
```

Conforme descrevemos anteriormente, podemos utilizar uma abordagem diferente: Primeiro criar um objeto OleDbCommand (1 e 2 a seguir), depois declaramos um objeto DataAdapter e passamos o objeto Command para a propriedade SelectCommand do objeto OleDbDataAdapter (3 e 4 a seguir):

```
1. string TextoDoComando = "SELECT CódigoDoCliente," +
"NomeDaEmpresa,País,Cidade FROM Clientes";
```

```
2. OleDbCommand MeuComando = new OleDbCommand(TextoDoComando,MinhaConexão);
```

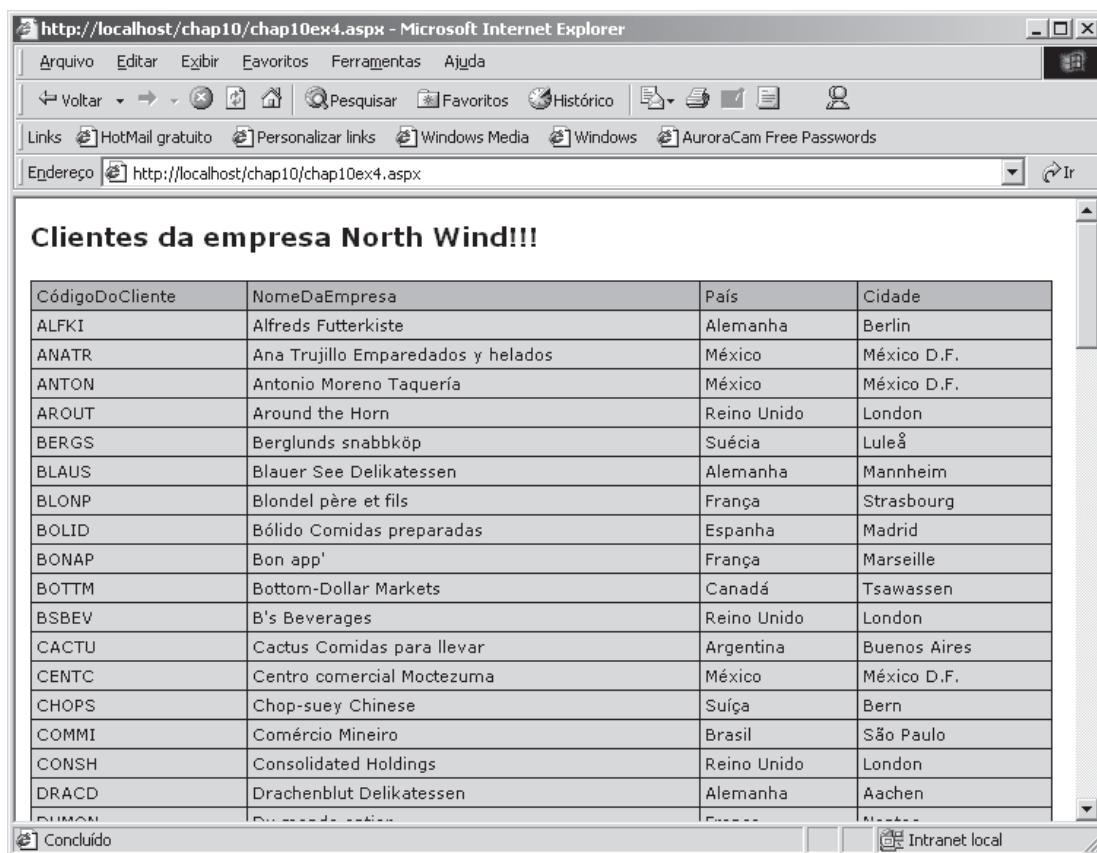


Figura 10.12: Exibindo dados do Banco de dados NorthWind.mdb.

```
3. OleDbDataAdapter MeuDataAdapter = new OleDbDataAdapter();
```

```
4. MeuDataAdapter.SelectCommand = MeuComando;
```

- ◆ O próximo passo é declarar um objeto do tipo DataSet, chamado ds, e utilizar o método Fill, do objeto OleDbDataAdapter, para preencher o objeto DataSet com os dados retornados a partir do banco de dados NorthWind:

```
DataSet ds = new DataSet();
```

```
MeuDataAdapter.Fill(ds);
```

O nome do objeto DataSet é passado como parâmetro para o método Fill.

- ◆ Na seqüência criamos um objeto do tipo DataView. A principal função de um objeto DataView é permitir a ligação de uma fonte de dados com um controle do tipo Web Form Controls, como é o caso do controle DataGrid. Um objeto DataView representa uma visão de uma tabela de um objeto DataSet. A visão representada pelo objeto DataView pode ser utilizada para pesquisar, ordenar, editar e navegar pelos registros da tabela. Na criação do objeto DataView, associamos o mesmo com a primeira tabela do objeto DataSet, o que é feito utilizando a coleção Tables do objeto DataSet. Observe que a primeira tabela da coleção possui índice zero (ds.Tables[0]), a segunda tabela da coleção possui índice um (ds.Tables[1]), e assim por diante.

```
DataView source = new DataView(ds.Tables[0]);
```

Em seguida definimos a propriedade DataSource do controle DataGrid (MinhaGrade), como sendo igual ao objeto DataView recém-criado:

```
MinhaGrade.DataSource = source ;
```

O passo final é chamar o método DataBind do controle DataGrid:

```
MinhaGrade.DataBind();
```

Feito isso, o controle DataGrid, colocado na seção de apresentação da página, irá exibir os registros retornados, no formato de uma tabela. Observe que não precisamos iniciar um laço While para navegar por cada registro do objeto DataView. Também não precisamos utilizar um “monte” de comandos Response.Write para montar a página de saída, como fazímos no ASP 3.0. Todo este trabalho é feito, automaticamente, pelo controle DataGrid.

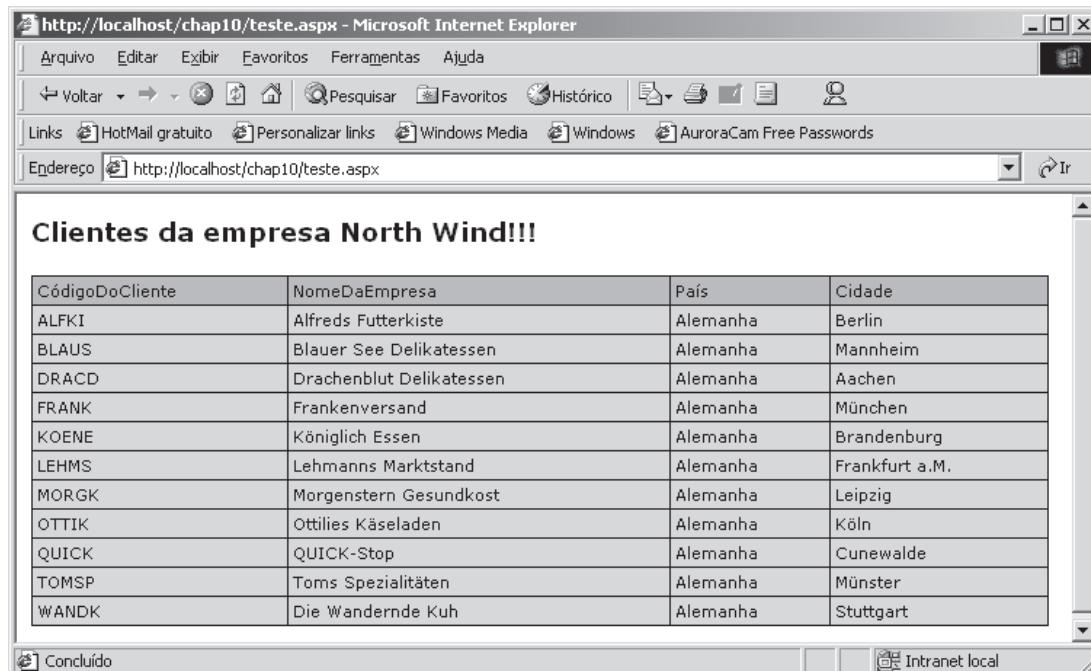
Este exemplo salienta bem o poder e flexibilidade dos Web Server Controls, como é o caso do controle DataGrid.

- ◆ A título de exemplo, vamos supor que você deseja exibir apenas os clientes da Alemanha. Para tal, basta alterar o comando SQL, da seguinte maneira:

```
OleDbDataAdapter MeuComando = new
OleDbDataAdapter("SELECT CódigoDoCliente,"
+ " NomeDaEmpresa, Cidade, País FROM Clientes where
País='Alemanha'", MinhaConexão);
```

**NOTA:** Ainda neste capítulo estudaremos o objeto DataView e o controle DataGrid, em maiores detalhes.

Você obterá o resultado indicado na Figura 10.13.



The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost/chap10/teste.aspx - Microsoft Internet Explorer". The page content is titled "Clientes da empresa North Wind!!!". A DataGrid is displayed with the following data:

CódigoDoCliente	NomeDaEmpresa	País	Cidade
ALFKI	Alfreds Futterkiste	Alemanha	Berlin
BLAUS	Blauer See Delikatessen	Alemanha	Mannheim
DRACD	Drachenblut Delikatessen	Alemanha	Aachen
FRANK	Frankenversand	Alemanha	München
KOENE	Königlich Essen	Alemanha	Brandenburg
LEHMS	Lehmanns Marktstand	Alemanha	Frankfurt a.M.
MORGK	Morgenstern Gesundkost	Alemanha	Leipzig
OTTIK	Ottilie's Käseladen	Alemanha	Köln
QUICK	QUICK-Stop	Alemanha	Cunewalde
TOMSP	Toms Spezialitäten	Alemanha	Münster
WANDK	Die Wandernde Kuh	Alemanha	Stuttgart

Figura 10.13: Exibindo os clientes da Alemanha.

## O Objeto DataView

Para entender a função do objeto DataView, vamos fazer uma recaptação dos objetos que já estudamos neste capítulo:

- ◆ Utilizamos um objeto OleDbConnection ou SqlConnection para estabelecer uma conexão com o banco de dados.

- ◆ O objeto DataAdapter faz a ponte entre um ou mais objetos Command (SqlCommand ou OleDbCommand) e um objeto DataSet. Ao final deste processo, os dados estão armazenados em um objeto DataSet.
- ◆ O objeto DataView faz a ponte entre o objeto DataSet e um Web Form ou Web Form Control, onde serão exibidos os dados. Na Figura 10.14, temos uma visão geral deste processo:

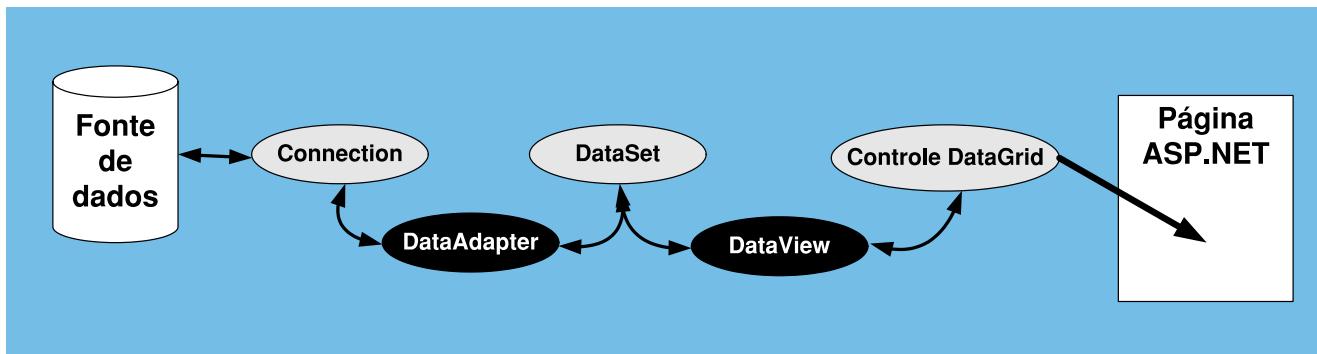


Figura 10.14: Acessando dados com ASP.NET.

Nesta figura fica bem destacado o papel de ponte entre o objeto DataSet e o controle DataGridView, exercido pelo objeto DataView. No exemplo da figura, utilizamos o controle DataGridView, mas poderia ser qualquer outro Web Form Control capaz de acessar dados de um DataView.

Um objeto DataView pode ser configurado para retornar apenas uma parte dos dados de um objeto DataTable. Podemos, por exemplo, ter dois objetos DataView, ligados com o mesmo objeto DataTable, porém exibindo diferentes dados. Vamos estudar mais alguns detalhes sobre o objeto DataView.

Na Tabela 10.13 temos uma descrição das principais propriedades da classe DataView:

**Tabela 10.13 Principais propriedades da classe DataView.**

Propriedade	Descrição
AllowDelete	Propriedade do tipo Booleana. Se contiver o valor True, podemos excluir registros; se contiver o valor False, não são permitidas exclusões. Esta propriedade pode ser utilizada para definir ou para retornar um valor True ou False.
AllowEdit	Utilizada para definir ou retornar um valor do tipo Booleano. Se o valor da propriedade for True, são permitidas alterações nos dados; caso contrário os dados não poderão ser alterados.
AllowNew	Utilizada para definir ou retornar um valor do tipo Booleano. Se o valor da propriedade for True, poderemos adicionar novos registros, utilizando o método AddNew; caso contrário novos registros não poderão ser adicionados.
Count	Retorna o número de registros no DataView.

Propriedade	Descrição
RowFilter	Utilizada para definir ou retornar uma expressão que determina quais os dados do objeto DataView que serão exibidos.
RowStateFilter	Com ADO.NET é mantido um histórico das alterações feitas nos dados. O objeto DataView mantém as versões anteriores de registros que foram alterados ou excluídos. Podemos utilizar esta propriedade para retornar ou definir uma expressão que filtra apenas os registros em um determinado estado, como por exemplo: alterados, excluídos, etc.
Sort	Utilizada para definir ou retornar informações sobre a(s) coluna(s) e a ordem de classificação dos dados.
Table	Utilizada para definir ou retornar a tabela a partir da qual o DataView obtém os dados.

Na Tabela 10.14 temos uma descrição dos principais métodos da classe DataView:

**Tabela 10.14** Principais métodos da classe DataView.

Método	Descrição
AddNew	Adiciona um novo registro ao DataView.
Delete	Exclui um registro do DataView.
Find	Localiza um determinado registro, com base no valor da chave primária.

Vamos apresentar alguns exemplos de utilização do objeto DataView.

Exemplo 1: Neste exemplo faremos uma conexão com o banco de dados C:\Meus documentos\Northwind.mdb. Vamos retornar os seguintes campos da tabela Funcionários:

- ◆ CódigoDoFuncionário
- ◆ Nome
- ◆ Cargo
- ◆ DataDeNascimento
- ◆ Cidade

Os dados serão apresentados na página, utilizando um controle DataGrid. Também apresentaremos um controle ListBox, onde o usuário poderá selecionar um campo pelo qual os dados serão ordenados em ordem Crescente. Por padrão, os dados são classificados pelo CódigoDoFuncionário.

Na Listagem 10.5 temos o código para o exemplo proposto.

Listagem 10.5 – Um exemplo completo com DataView.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>

<html>

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

protected void Page_Load(Object Src, EventArgs E)
{
 // Crio uma conexão com o banco de dados do Microsoft Access.
 // C:\Meus documentos\NorthWind.mdb.

 String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
 "DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";

 OleDbConnection MinhaConexão = new OleDbConnection(DefineConexão);

 // Utilizamos um objeto DataAdapter para executar um comando SQL,
 // o qual retorna todos os dados da tabela "Clientes".

 OleDbDataAdapter MeuComando = new OleDbDataAdapter("SELECT
CódigoDoFuncionário," + "Nome,Cargo,DataDeNascimento,Cidade
FROM Funcionários", MinhaConexão);

 // Criamos e preenchemos um objeto DataSet.
 // Observe que não temos mais o objeto Recordset,
 // como era de praxe com o ASP 3.0.

 DataSet ds = new DataSet();

 // Utilizo o método Fill do objeto DataAdapter, para preencher
 // o objeto DataSet, com os dados retornados pelo comando SQL.
```

```
MeuComando.Fill(ds);

// Conectamos um controle DataGrid com o DataSet criado anteriormente.
// MinhaGrade é o id (nome) de um controle do tipo
// DataGrid que está na seção de apresentação da página.

DataView source = new DataView(ds.Tables[0]);

// Utilizamos uma instrução Switch para definir a
// classificação dos dados com base no valor selecionado
// na lista CampoClassificar.

string aux = CampoClassificar.SelectedItem.Value;

switch (aux)
{
 case "Cargo":
 source.Sort="Cargo ASC";
 break;

 case "Cidade":
 source.Sort="Cidade ASC";
 break;

 case "Código do Funcionário":
 source.Sort="CódigoDoFuncionário ASC";
 break;

 case "Data de Nascimento":
 source.Sort="DataDeNascimento ASC";
 break;

 case "Nome":
 source.Sort="Nome ASC";
 break;
 default:
 // outras opções.
```

```

 break;

 }

 MinhaGrade.DataSource = source ;
 MinhaGrade.DataBind();
}

</script>
<body>

<%— Exibe as informações do DataGrid no corpo da página. —%>
<h3>Funcionários da empresa North Wind!!!</h3>

<form runat=server>

<h3>Classificar por:</h3>

 <asp:ListBox id="CampoClassificar"
 Rows="1"
 Width="200px"
 runat="server">

 <asp:ListItem>Cargo</asp:ListItem>
 <asp:ListItem>Cidade</asp:ListItem>
 <asp:ListItem selected="True">Código do Funcionário</asp:ListItem>
 <asp:ListItem>Data de Nascimento</asp:ListItem>
 <asp:ListItem>Nome</asp:ListItem>
 </asp:ListBox>

 <ASP:DataGrid id="MinhaGrade" runat="server"
 Width="700"
 BackColor="#ccccff"
 BorderColor="black"
 ShowFooter="false"
 CellPadding=3
 CellSpacing="0"
 Font-Name="Verdana"
 Font-Size="8pt"
 HeaderStyle-BackColor="#aaaadd"
 MaintainState="false"
 />

```

&lt;BR&gt;

```

Clique para Classificar --></td>

<input type=submit value="Classificar" runat="server">

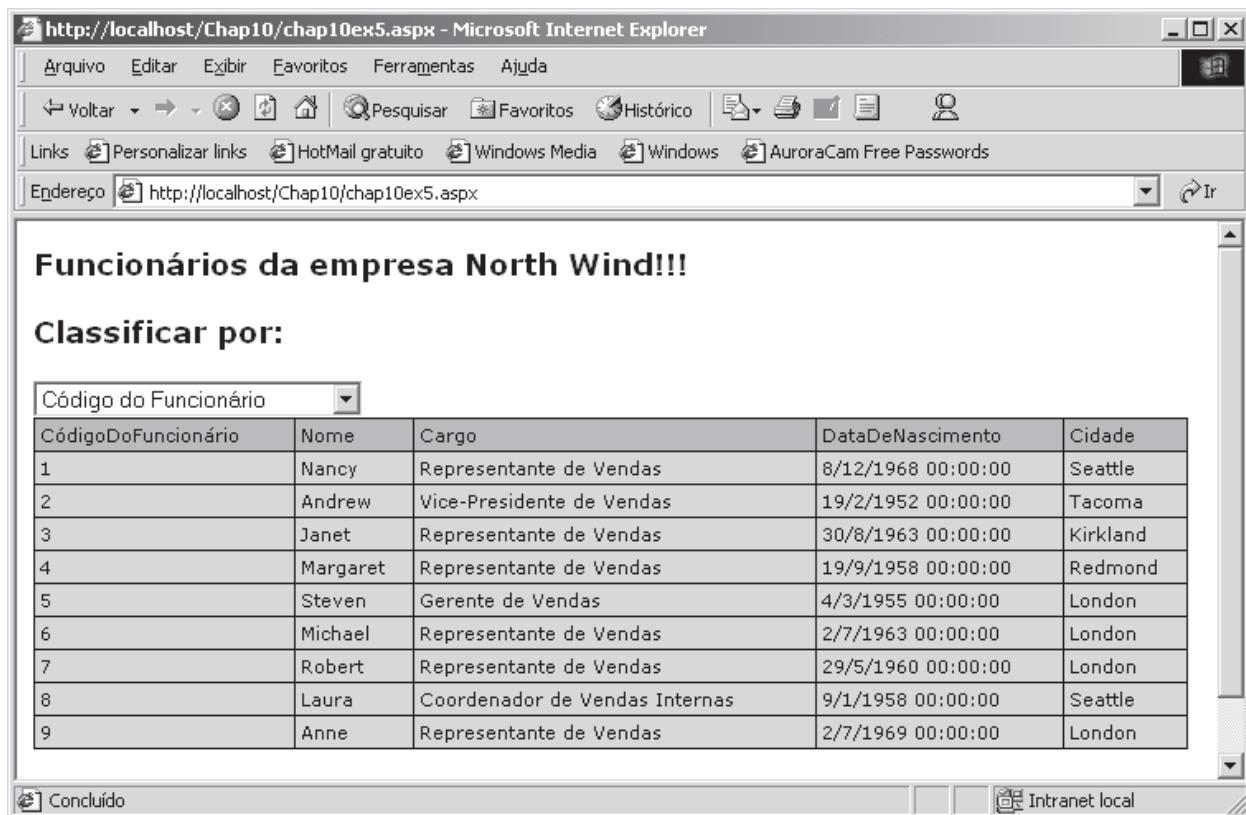
</form>
</body>
</html>

```

Digite o código da Listagem 10.5 e salve o mesmo em um arquivo chamado chap10ex5.aspx, na pasta chap10, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap10/chap10ex5.aspx>

Você irá obter a página indicada na Figura 10.15.



**Figura 10.15: Relação de funcionários – classificada pelo Código do Funcionário.**

Observe que a listagem está classificada em ordem crescente do Código do Funcionário. Vamos testar o nosso exemplo. Na lista “Classificar Por”, selecione Nome e dê um clique no botão Classificar. Você obterá uma listagem classificada pelo nome do funcionário, conforme indicado na Figura 10.16:

Comentários sobre o código do exemplo – Chap10Ex5.aspx.

◆ O código para conectar com o banco de dados e retornar dados é muito semelhante ao código utilizado nos exemplos anteriores. Para maiores detalhes sobre esta parte do código consulte os exemplos das Listagens 10.1, 10.2, 10.3 e 10.4.

◆ Na seção de apresentação da página criamos um formulário e adicionamos um Web Server Control do tipo ListBox chamado CampoClassificar. Este controle exibe as opções de classificação. O código que define o controle ListBox está indicado a seguir:

```
<asp:ListBox id="CampoClassificar"
 Rows="1"
 Width="200px"
 runat="server">

 <asp:ListItem>Cargo</asp:ListItem>
 <asp:ListItem>Cidade</asp:ListItem>
 <asp:ListItem selected="True">Código do Funcionário</asp:ListItem>
 <asp:ListItem>Data de Nascimento</asp:ListItem>
 <asp:ListItem>Nome</asp:ListItem>
</asp:ListBox>
```

The screenshot shows a Microsoft Internet Explorer window displaying a list of employees from the North Wind database. The title bar reads "http://localhost/Chap10/chap10ex6.aspx - Microsoft Internet Explorer". The page content is titled "Funcionários da empresa North Wind!!!". It features a button labeled "Clique para Classificar -->" followed by a "Classificar" button. Below this, there are three dropdown menus for sorting: "Classificar primeiro por:", "Em seguida por:", and "Em qual ordem?". The first dropdown is set to "NúmeroDoPedido", the second to "DataDoPedido", and the third has radio buttons for "Crescente" (selected) and "Decrescente". A table below lists 14 employees with columns: NúmeroDoPedido, DataDoPedido, CidadeDeDestino, and PaísDeDestino. The data is as follows:

NúmeroDoPedido	DataDoPedido	CidadeDeDestino	PaísDeDestino
10248	4/7/1996 00:00:00	Reims	França
10249	5/7/1996 00:00:00	Münster	Alemanha
10250	8/7/1996 00:00:00	Rio de Janeiro	Brasil
10251	8/7/1996 00:00:00	Lyon	França
10252	9/7/1996 00:00:00	Charleroi	Bélgica
10253	10/7/1996 00:00:00	Rio de Janeiro	Brasil
10254	11/7/1996 00:00:00	Bern	Suíça
10255	12/7/1996 00:00:00	Genève	Suíça
10256	15/7/1996 00:00:00	Resende	Brasil
10257	16/7/1996 00:00:00	San Cristóbal	Venezuela
10258	17/7/1996 00:00:00	Graz	Áustria
10259	18/7/1996 00:00:00	México D.F.	México
10260	19/7/1996 00:00:00	Köln	Alemanha
10261	19/7/1996 00:00:00	Rio de Janeiro	Brasil
10262	22/7/1996 00:00:00	Albuquerque	EUA
10263	23/7/1996 00:00:00	Graz	Áustria

At the bottom left is a "Concluido" button, and at the bottom right is a "Intranet local" link.

Figura 10.16: Relação de funcionários – classificada pelo Nome.

Observe que, por padrão, ao carregarmos a página, a opção Código do Funcionário vem selecionada – selected=”True”. Para maiores informações sobre o controle ListBox, consulte o Capítulo 9.

- ♦ Na seção de código precisamos verificar qual a opção selecionada no ListBox e, com base na opção selecionada, classificar os dados do DataView. Para isso utilizamos a instrução switch/case do C#, conforme indicado a seguir:

```
string aux = CampoClassificar.SelectedItem.Value;
<1
switch (aux)
{
 case "Cargo":
 source.Sort="Cargo ASC";
 break;

 case "Cidade":
 source.Sort="Cidade ASC";
 break;

 case "Código do Funcionário":
 source.Sort="CódigoDoFuncionário ASC";
 break;

 case "Data de Nascimento":
 source.Sort="DataDeNascimento ASC";
 break;

 case "Nome":
 source.Sort="Nome ASC";
 break;

 default:
 // outras opções.
 break;
}
```

Primeiro criamos uma string aux e atribuímos a esta string o valor do item selecionado na ListBox CampoClassificar:

```
string aux = CampoClassificar.SelectedItem.Value;
```

Em seguida, utilizamos a string aux como variável de comparação para a instrução switch/case. Com base no valor da string aux, definimos a classificação dos dados do DataView. Para definir a classificação utilizamos a propriedade Sort do objeto source, objeto este que é do tipo DataView, como no exemplo a seguir:

```
source.Sort="CódigoDoFuncionário ASC";
```

A propriedade Sort é uma String que contém o nome do campo seguido de um espaço e da palavra ASC para classificação crescente ou DESC para classificação decrescente.

Exemplo 2: Neste exemplo faremos uma conexão com o banco de dados C:\Meus documentos\Northwind.mdb. Vamos retornar os seguintes campos da tabela Pedidos:

- ◆ NúmeroDoPedido
- ◆ DataDoPedido
- ◆ CidadeDeDestino
- ◆ PaísDeDestino

**IMPORTANTE:** Para maiores informações sobre a instrução switch/case e sobre as demais instruções de controle de fluxo do C#, consulte o Capítulo 3.

Os dados serão apresentados na página, utilizando um controle DataGridView. Também apresentaremos dois controles ListBox. O usuário poderá classificar os dados por um ou dois campos. Caso deseje classificar apenas por um campo, basta selecionar “Não classificar”, na segunda lista. Por padrão os dados são classificados apenas pelo NúmeroDoPedido. Também apresentaremos dois controles do tipo RadioButton, onde o usuário pode selecionar classificação Crescente ou Decrescente.

Na Listagem 10.6 temos o código para o exemplo proposto.

#### Listagem 10.6 – Um exemplo completo com DataView e o método Sort com dois campos.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>

<html>

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

 protected void Page_Load(Object Src, EventArgs E)
 {

 // Crio uma conexão com o banco de dados do Microsoft Access.
 // C:\Meus documentos\NorthWind.mdb.

 String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +

```

```
"DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";

OleDbConnection MinhaConexão = new OleDbConnection(DefineConexão);
// Utilizamos um objeto DataAdapter para executar um comando SQL,
// o qual retorna todos os dados da tabela "Clientes".

OleDbDataAdapter MeuComando = new OleDbDataAdapter("SELECT NúmeroDoPedido," +
"DataDoPedido,CidadeDeDestino,PaísDeDestino FROM Pedidos", MinhaConexão);

// Criamos e preenchemos um objeto DataSet.
// Observe que não temos mais o objeto Recordset,
// como era de praxe com o ASP 3.0.

DataSet ds = new DataSet();

// Utilizo o método Fill do objeto DataAdapter, para preencher
// o objeto DataSet, com os dados retornados pelo comando SQL.

MeuComando.Fill(ds);

// Conectamos um controle DataGridView com o DataSet criado anteriormente.
// MinhaGrade é o id (nome) de um controle do tipo
// DataGridView que está na seção de apresentação da página.

DataGridView source = new DataGridView(ds.Tables[0]);

// Crio uma variável string chamada aux.
// Na variável aux vamos concatenando os valores
// Selecionados nas listas CampoClassificar1 e
// CampoClassificar2, além das opções ASC ou DESC,
// dependendo da escolha do usuário.

string aux = CampoClassificar1.SelectedItem.Value;

// Se o usuário selecionou Crescente, concateno ASC,
// caso contrário, concateno DESC.
```

```

if (Crescente.Checked)
{
 aux = aux + " ASC,";
}
else
{
 aux = aux + " DESC,";
}

aux = aux + CampoClassificar2.SelectedItem.Value;

if (Crescente.Checked)
{
 aux = aux + " ASC";
}
else
{
 aux = aux + " DESC";
}

// Ordeno passando aux para a propriedade Sort.

source.Sort = aux;

MinhaGrade.DataSource = source ;
MinhaGrade.DataBind();
}

</script>

<body>

<%-- Exibe as informações do DataGrid no corpo da página. --%>

<h3>Funcionários da empresa North Wind!!!</h3>

```

```
<form runat=server>

 Clique para Classificar -></td>
 <input type=submit value="Classificar" runat="server">

 <table>

 <tr>
 <td>

 Classificar primeiro por:
 </td>

 <td>

 Em seguida por:
 </td>

 <td>
 Em qual ordem?
 </td>
 </tr>

 <tr>
 <td>
 <asp:ListBox id="CampoClassificar1"
 Rows="1"
 Width="200px"
 runat="server">
 <asp:ListItem>CidadeDeDestino</asp:ListItem>
 <asp:ListItem>DataDoPedido</asp:ListItem>
 <asp:ListItem selected="True">NúmeroDoPedido</asp:ListItem>
 <asp:ListItem>PaísDeDestino</asp:ListItem>
 </asp:ListBox>
 </td>
 </tr>
 </table>
</form>
```

```
</td>

<td>

 <asp:ListBox id="CampoClassificar2"
 Rows="1"
 Width="200px"
 runat="server">
 <asp:ListItem>CidadeDeDestino</asp:ListItem>
 <asp:ListItem selected="True">DataDoPedido</asp:ListItem>
 <asp:ListItem>NúmeroDoPedido</asp:ListItem>
 <asp:ListItem>PaísDeDestino</asp:ListItem>
 </asp:ListBox>
</td>

<td>
 <asp:RadioButton
 id="Crescente"
 Text="Crescente"
 Checked="True"
 GroupName="Ordem"
 runat="server"
 />

 <asp:RadioButton
 id="Decrescente"
 Text="Decrescente"
 GroupName="Ordem"
 runat="server"
 />
</td>

</tr>
```

```
</table>

<asp:DataGrid
 id="MinhaGrade"
 runat="server"
 Width="700"
 BackColor="#ccccff"
 BorderColor="black"
 ShowFooter="false"
 CellPadding=3
 CellSpacing="0"
 Font-Name="Verdana"
 Font-Size="8pt"
 HeaderStyle-BackColor="#aaaadd"
 MaintainState="false"
/>

</form>

</body>
</html>
```

Digite o código da Listagem 10.6 e salve o mesmo em um arquivo chamado chap10ex6.aspx, na pasta chap10, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap10/chap10ex6.aspx>

Você irá obter a página indicada na Figura 10.17.

Observe que a listagem está classificada primeiro em ordem crescente do NúmeroDoPedido, em seguida em ordem Crescente da DataDoPedido. Vamos testar o nosso exemplo. Na primeira lista selecione PaísDeDestino e na segunda lista selecione CidadeDeDestino. Para ordem de classificação clique na opção Decrescente. Dê um clique no botão Classificar. Você obterá uma listagem classificada pelo nome de Pedidos. Parte do resultado está indicado na Figura 10.18.

10957	18/3/1998 00:00:00	San Cristóbal	Venezuela
10960	19/3/1998 00:00:00	San Cristóbal	Venezuela
10976	25/3/1998 00:00:00	San Cristóbal	Venezuela
11055	28/4/1998 00:00:00	San Cristóbal	Venezuela
10405	6/1/1997 00:00:00	I. de Margarita	Venezuela
10485	25/3/1997 00:00:00	I. de Margarita	Venezuela
10638	20/8/1997 00:00:00	I. de Margarita	Venezuela
10697	8/10/1997 00:00:00	I. de Margarita	Venezuela
10729	4/11/1997 00:00:00	I. de Margarita	Venezuela
10811	2/1/1998 00:00:00	I. de Margarita	Venezuela
10838	19/1/1998 00:00:00	I. de Margarita	Venezuela
10840	19/1/1998 00:00:00	I. de Margarita	Venezuela
10919	2/3/1998 00:00:00	I. de Margarita	Venezuela
10954	17/3/1998 00:00:00	I. de Margarita	Venezuela
11014	10/4/1998 00:00:00	I. de Margarita	Venezuela
11039	21/4/1998 00:00:00	I. de Margarita	Venezuela
10268	30/7/1996 00:00:00	Caracas	Venezuela
10785	18/12/1997 00:00:00	Caracas	Venezuela
10283	16/8/1996 00:00:00	Barquisimeto	Venezuela
10296	3/9/1996 00:00:00	Barquisimeto	Venezuela

Figura 10.17: Relação de pedidos – opção para classificar por dois campos diferentes.

Comentários sobre o código do exemplo – Chap10Ex6.aspx.

- Utilizamos uma tabela para fazer o alinhamento dos controles, no início da página. Para fazer um controle mais refinado do alinhamento, utilizamos o caracter &nbsp – non break space. Este é um caracter especial do HTML utilizado para espaço em branco.

NúmeroDoPedido	DataDoPedido	CidadeDeDestino	PaísDeDestino
10248	4/7/1996 00:00:00	Reims	França
10249	5/7/1996 00:00:00	Münster	Alemanha
10250	8/7/1996 00:00:00	Rio de Janeiro	Brasil
10251	8/7/1996 00:00:00	Lyon	França
10252	9/7/1996 00:00:00	Charleroi	Bélgica
10253	10/7/1996 00:00:00	Rio de Janeiro	Brasil
10254	11/7/1996 00:00:00	Bern	Suíça
10255	12/7/1996 00:00:00	Genève	Suíça
10256	15/7/1996 00:00:00	Resende	Brasil
10257	16/7/1996 00:00:00	San Cristóbal	Venezuela

Figura 10.18: Relação de Pedidos – classificada por País e dentro do país por Cidade.

- ◆ No formulário, na seção de apresentação da página, adicionamos dois controles do tipo ListBox, onde o usuário pode selecionar por quais campos deseja classificar a listagem. Também acrescentamos um grupo chamado Ordem, com dois controles do tipo RadioButton, onde o usuário pode selecionar Crescente ou Decrescente. Com base nos valores selecionados nestes controles, definimos o conteúdo de uma variável string – aux. Uma vez definido o conteúdo da string aux, utilizamos esta para definir o valor da propriedade Sort do objeto source, o qual é um objeto do tipo DataView:

```
string aux = CampoClassificar1.SelectedItem.Value;

// Se o usuário selecionou Crescente, concateno ASC,
// caso contrário, concateno DESC.

if (Crescente.Checked)
{
 aux = aux + " ASC,";

}
else
{
 aux = aux + " DESC,";
}

aux = aux + CampoClassificar2.SelectedItem.Value;

if (Crescente.Checked)
{
 aux = aux + " ASC";
}
else
{
 aux = aux + " DESC";
}

// Ordeno passando aux para a propriedade Sort.

source.Sort = aux;
```

Observe que, no segundo if, não acrescentamos a vírgula após ASC ou DESC. No primeiro if acrescentamos, pois, quando temos dois ou mais campos para definir a propriedade Sort, precisamos separar estes campos por vírgula. Se o usuário selecionar PaísDeDestino na primeira lista, CidadeDeDestino na segunda lista e Crescente, a variável aux terá o seguinte valor:

```
aux = "PaísDeDestino ASC, CidadeDeDestino ASC"
```

Este valor é passado para a propriedade Sort para que a classificação seja feita de acordo com os critérios selecionados pelo usuário.

- ◆ Os demais elementos do exemplo Chap10ex6.aspx já foram vistos e explicados em exemplos anteriores. Para maiores detalhes sobre os mesmos consulte os exemplos destes e dos demais capítulos.
- ◆ Novamente utilizamos o evento Load da página Page\_Load, para executar a conexão com o banco de dados, retornar os dados, definir a string aux e ordenar os dados de acordo com os critérios estabelecidos pelo usuário.

Exercício: Vou propor um exercício para o amigo leitor. Caso você tenha alguma dificuldade para resolvê-lo, é só entrar em contato através do e-mail: batisti@hotmail.com ou batisti@juliobattisti.com.br.

Na página Chap10ex6.aspx podemos definir somente um critério de classificação para os campos selecionados nas duas listas, ou seja, somente Crescente ou somente Decrescente. O exercício que fica para o leitor é tornar o nosso exemplo um pouco mais flexível, de tal maneira que possamos escolher uma ordem de classificação independente para cada campo. Explico um pouco melhor: Selecionar classificação Crescente para o campo PaísDeDestino e, dentro de um mesmo país, classificação Decrescente por CidadeDeDestino.

Fica o desafio. Em caso de dúvida é só entrar em contato.

## Um Pouco Mais Sobre o Controle DataGrid

O controle DataGrid, sem sombra de dúvidas, é assunto para um capítulo inteiro. Como não dispomos de tanto espaço, caso contrário teríamos um livro de mais de 2000 páginas, vamos apresentar as principais características deste poderoso controle, através de alguns exemplos práticos. Neste tópico veremos como algumas propriedades e métodos do controle DataGrid são capazes de efetuar verdadeiras maravilhas. Nos Capítulos 11 e 12 estudaremos mais sobre o controle DataGrid, principalmente sobre como fazer alterações e exclusões nos dados exibidos pelo controle.

### Primeira Maravilha do DataGrid: Paginação

Quando obtemos um número grande de registros é interessante dividir a exibição do mesmo em páginas, onde exibimos um determinado número de registros por vez. Isso é o que chamamos de paginação. Por exemplo, podemos fazer com que 1000 registros sejam divididos em páginas onde são exibidos 20 registros por vez.

Fazer paginação com ASP 3.0 não era uma tarefa das mais fáceis. Um boa quantia de código era necessária e tudo tinha que ser implementado manualmente. Com o controle DataGrid, definir paginação está resumido a configurar duas propriedades:

- ◆ **AllowPaging:** Esta propriedade é do tipo Boleana. Se for True, a paginação é habilitada; se for False, a paginação é desabilitada. Por padrão esta propriedade é False. Ao tornarmos AllowPaging True, na última linha do DataGrid são exibidos os símbolos “<” para voltar à página anterior e “>” para ir à próxima página. Podemos alterar estes valores utilizando as propriedades PagerStyle-NextPageText para alterar o texto do link que vai para a próxima página e PagerStyle-PrevPageText, para alterar o texto do link que volta para a página anterior.

Quando estamos na primeira página, o link para a página anterior é automaticamente desabilitado, pois não existe página anterior à primeira. Quando estamos na última página, o link para a próxima página é automaticamente desabilitado, pois não existe página após a última.

- ◆ **PageSize:** O valor desta propriedade define o número de registros que serão exibidos, por vez. Se esta propriedade não for definida, serão exibidos 10 registros por página.

Para definir que sejam exibidos 20 registros por página, sendo “Próxima página >>”, o texto do link para a próxima página e “<< Página anterior”, o texto para o link para a página anterior, definimos as seguintes propriedades/valores do controle DataGrid:

```
AllowPaging="True"
PageSize="20"
PagerStyle-NextPageText="Próxima página >>"
PagerStyle-PrevPageText="<< Página anterior"
```

Inserindo a definição para essas propriedades, a tag que define o controle DataGrid ficaria da seguinte maneira:

```
<ASP:DataGrid
 id="MeuDataGrid"
 runat="server"
 Width="600"
 BackColor="#bbccff"
 AllowPaging="True"
 PageSize="20"
 PagerStyle-NextPageText="Próxima página >>"
 PagerStyle-PrevPageText="<< Página anterior"
 BorderColor="black"
 ShowFooter="false"
 CellPadding=3
 CellSpacing="0"
 Font-Name="Verdana"
 Font-Size="8pt"
 HeaderStyle-BackColor="#aaaadd"
 MaintainState="false"
/>
```

## Segunda Maravilha do DataGrid: Mais do que um Conjunto de Dados na Mesma Página

Vamos ver um exemplo prático, onde exibiremos dados de duas tabelas, do banco de dados NorthWind, na mesma página ASP.NET. Para isso faremos uso de dois controles do tipo DataGrid.

O exemplo Proposto: Exibir, na mesma página, informações da tabela Funcionários e da tabela Transportadores.

Na Listagem 10.7 temos o código para o exemplo proposto.

**Listagem 10.7 – Exibindo dados de múltiplas tabelas em uma página ASP.NET.**

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>

<html>

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

 protected void Page_Load(Object Src, EventArgs E)
 {

 // Crio uma conexão com o banco de dados do Microsoft Access.
 // C:\Meus documentos\NorthWind.mdb.

 String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
 "DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";

 OleDbConnection MinhaConexão = new OleDbConnection(DefineConexão);

 // Para podermos acessar múltiplas tabelas vamos criar
 // um objeto Command, conforme indicado a seguir:

 OleDbCommand MeuComando = new OleDbCommand();

 // defino algumas propriedades do objeto Command.

 MeuComando.Connection = MinhaConexão;
 MeuComando.CommandType = CommandType.Text;
```

**IMPORTANTE:** Para que a paginação seja feita, além de definir a propriedade AllowPaging como True, temos que criar código para trocar de página. O método que faz a troca de página é definido na propriedade OnPageIndexChanged do DataGrid. Aprenderemos a implementar a paginação no Capítulo 11, onde veremos mais detalhes sobre o controle DataGrid.

```
// Utilizamos um objeto DataAdapter para executar dois comandos SQL.
// Um comando para retornar dados da tabela Funcionários.
// Um comando para retornar dados da tabela Transportadoras.

OleDbDataAdapter MeuDataAdapter = new OleDbDataAdapter();

// Definimos o comando a ser utilizado pelo objeto DataAdapter.
// Para isso definimos a sua propriedade SelectCommand.

MeuDataAdapter.SelectCommand = MeuComando;

// Criamos e preenchemos um objeto DataSet.
// Vamos preencher o DataSet com dados das tabelas
// Funcionários e Transportadores.
// Para isso precisamos executar dois comandos SQL.
// *****
// Na prática vamos chamar o método Fill do DataAdapter
// duas vezes.
// Antes de cada chamada alteramos a propriedade
// CommandText do objeto Command.

DataSet ds = new DataSet();

MeuComando.CommandText="Select CódigoDoFuncionário, Nome, Sobrenome, Cargo
From Funcionários";

MeuDataAdapter.Fill(ds,"Funcionários");

// Altero a propriedade CommandText para retornar dados
// da tabela Transportadores.
// Chamo novamente o método Fill.

MeuComando.CommandText = "Select * From Transportadoras";
MeuDataAdapter.Fill(ds,"Transportadoras");

// Conectamos um controle DataGrid com
```

```
// cada tabela do DataSet criado anteriormente.
// *****
// GradeFuncionários é o id (nome) de um controle do tipo
// DataGrid que está na seção de apresentação da página.
// Este controle exibirá dados dos Funcionários.
// *****
// GradeTransportadoras é o id (nome) de um controle do tipo
// DataGrid que está na seção de apresentação da página.
// Este controle exibirá dados das Transportadoras.

DataView Funcionários = new DataView(ds.Tables[0]);
GradeFuncionários.DataSource = Funcionários ;
GradeFuncionários.DataBind();

DataView Transportadoras = new DataView(ds.Tables[1]);
GradeTransportadoras.DataSource = Transportadoras ;
GradeTransportadoras.DataBind();
}

</script>

<body>

<h3>Funcionários da empresa North Wind!!!</h3>
<HR>

<ASP:DataGrid
 id="GradeFuncionários"
 runat="server"
 Width="450"
 BackColor="#bbccff"
 BorderColor="blue"
 ShowFooter="false"
 CellPadding=3
 CellSpacing="0"
 Font-Name="Verdana"
 Font-Size="8pt"
```

```
 HeaderStyle-BackColor="#aaaadd"
HeaderStyle-Font-Bold="True"
 MaintainState="false"
/>

<HR>

<h3>Transportadoras da empresa North Wind!!!</h3>

<ASP:DataGrid
 id="GradeTransportadoras"
 runat="server"
 Width="400"
 BackColor="#bbddff"
 BorderColor="blue"
 ShowFooter="false"
 CellPadding=3
 CellSpacing="0"
 Font-Name="Verdana"
 Font-Size="8pt"
 HeaderStyle-BackColor="#aaaadd"
HeaderStyle-Font-Bold="True"
 MaintainState="false"
/>
<HR>

</body>
</html>
```

Digite o código da Listagem 10.7 e salve o mesmo em um arquivo chamado chap10ex7.aspx, na pasta chap10, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap10/chap10ex7.aspx>

Você irá obter a página indicada na Figura 10.20.

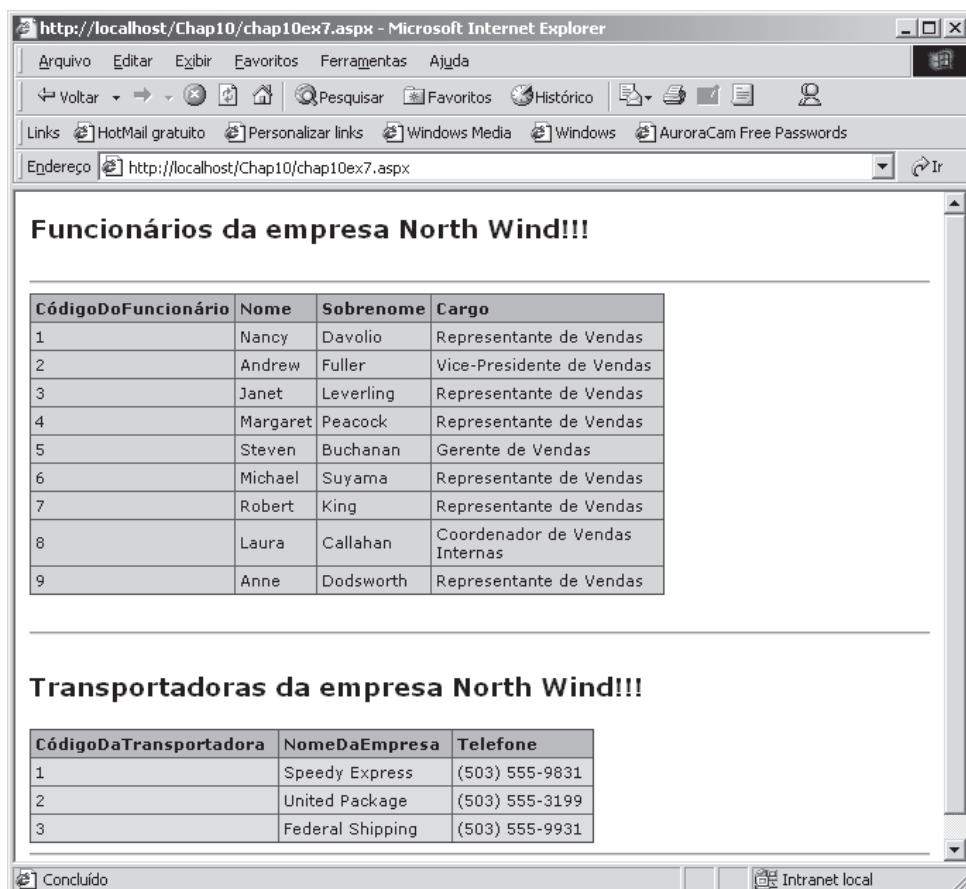


Figura 10.19: Exibindo dados de múltiplas tabelas em uma página ASP.NET.

Comentários sobre o código do exemplo – Chap10Ex7.aspx.

- ◆ Vamos descrever os passos utilizados para acessar e exibir dados das tabelas Funcionários e Pedidos. Alguns passos já foram explicados em exemplos anteriores, mas iremos repetir a explicação, para fazermos uma revisão do conteúdo deste capítulo:
1. Utilizamos o evento Pge\_Load para colocar o código necessário ao nosso exemplo.
  2. Iniciamos estabelecendo uma conexão com o banco de dados NorthWind.mdb:
- ```
String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
    "DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";
```
- ```
OleDbConnection MinhaConexão = new OleDbConnection(DefineConexão);
```
3. A partir deste ponto temos algumas mudanças em relação aos exemplos anteriores. Primeiro vamos criar um objeto OleDbCommand e definir as propriedades Connection e CommandType do objeto OleDbCommand:
- ```
OleDbCommand MeuComando = new OleDbCommand();
```
- ```
// defino algumas propriedades do objeto Command.
```
- ```
MeuComando.Connection = MinhaConexão;
```
- ```
MeuComando.CommandType = CommandType.Text;
```

4. Agora criamos um objeto DataAdapter e definimos a sua propriedade SelectCommand. Ao definirmos esta propriedade informamos qual o objeto OleDbCommand que estará associado ao DataAdapter:

```
OleDbDataAdapter MeuDataAdapter = new OleDbDataAdapter();
MeuDataAdapter.SelectCommand = MeuComando;
```

5. Declaramos um objeto ds do tipo DataSet. O objeto DataSet será o Conteiner para as tabelas Funcionários e Transportadoras. Recapitulando o que estudamos anteriormente, o objeto DataSet pode conter uma ou mais tabelas, as quais estão contidas na coleção Tables do objeto.

```
DataSet ds = new DataSet();
```

6. Chegamos ao ponto principal do nosso exemplo. Utilizamos a seguinte técnica: definimos a propriedade CommandText do objeto MeuComando. Esta propriedade contém a string SQL que acessa dados de uma determinada tabela:

```
MeuComando.CommandText="Select CódigoDoFuncionário, Nome, Sobrenome, Cargo
From Funcionários";
```

Em seguida chamamos o método Fill do objeto DataAdapter. Ao chamarmos este método, o comando definido na propriedade CommandText é executado e os dados retornados são passados para o DataSet definido no primeiro parâmetro. O segundo parâmetro é o nome do conjunto de dados no DataSet. No nosso exemplo utilizamos o mesmo nome da tabela, no banco de dados NorthWind, porém isso não é obrigatório:

```
MeuDataAdapter.Fill(ds,"Funcionários");
```

Repetimos os mesmos passos para retornar dados da tabela Transportadoras e colocá-los no DataSet ds:

```
MeuComando.CommandText = "Select * From Transportadoras";
MeuDataAdapter.Fill(ds,"Transportadoras");
```

Após a execução destes comandos, a coleção Tables, do DataSet ds, contém duas tabelas com dados retornados a partir do Banco de dados NorthWind.mdb.

7. O próximo passo é exibir os dados do objeto DataSet na página ASP.NET. Isto é feito utilizando um objeto do tipo DataView para cada tabela a ser exibida. Ao criarmos o objeto DataView, já passamos como parâmetro a tabela associada ao objeto. Em seguida definimos a propriedade DataSource do controle DataGrid como sendo igual ao objeto DataView recém-criado. O passo final é chamar o método DataBind do controle DataGrid:

```
DataView Funcionários = new DataView(ds.Tables[0]);
GradeFuncionários.DataSource = Funcionários ;
GradeFuncionários.DataBind();
```

```
DataView Transportadoras = new DataView(ds.Tables[1]);
GradeTransportadoras.DataSource = Transportadoras ;
GradeTransportadoras.DataBind();
```

Para cada tabela do DataSet criamos um objeto DataView. Para o DataView Funcionários, passamos como parâmetro: ds.Tables[0], ou seja, a primeira tabela, da coleção de tabelas do DataSet ds. Para o DataView Transportadoras, passamos como parâmetro ds.Tables[1], ou seja, a segunda tabela, da coleção de tabelas do DataSet ds. Depois é só ligar cada DataView com o respectivo DataGrid.

8. Na seção de apresentação, utilizamos algumas propriedades do DataGrid para definir a sua aparência. Por exemplo:
  - ◆ **HeaderStyle-BackColor="#aaaadd":** Define a cor de segundo plano da primeira linha do DataGrid, a linha que contém os títulos das colunas.
  - ◆ **HeaderStyle-Font-Bold="True":** Define que o texto da primeira linha deve ser exibido com fonte em negrito.

Este exemplo demonstra, mais uma vez, o poder e flexibilidade dos novos objetos para acesso a dados oferecidos pelo ADO.NET e também demonstra o poder dos Web Server Controls, mais especificamente do controle DataGrid.

## Conclusão

Neste capítulo aprendemos a conectar páginas ASP.NET com bancos de dados. Utilizamos classes, basicamente, dos seguintes namespaces:

- ◆ System.Data
- ◆ System.Data.OleDb
- ◆ System.Data.SqlClient

Estudamos, em detalhes, diversas classes destes namespaces:

- ◆ SqlConnection/OleDbConnection
- ◆ SqlCommand/OleDbCommand
- ◆ SqlDataAdapter/OleDbDataAdapter
- ◆ DataSet
- ◆ DataView

Também estudamos algumas características do poderoso controle: DataGrid.

Nos próximos capítulos estudaremos mais sobre estas classes e sobre controles que podem ser conectados com dados.

No Capítulo 10 aprendemos a conectar páginas ASP.NET com bancos de dados. Trabalhamos com classes dos namespaces System.Data, System.Data.SqlClient e System.Data.OleDb. Para exibir os dados em um página ASP.NET utilizamos, basicamente, o controle DataGrid.

Existem diversos Web Server Controls que podem exibir dados a partir de um objeto DataView ou DataReader (será visto neste capítulo). Iniciaremos este capítulo estudando os seguintes Web Server Controls:

- ◆ CheckBoxList
- ◆ DropDownList
- ◆ RadioButtonList

Estes controles também são conhecidos como: Data-bound list controls. Todos possuem uma propriedade DataSource, que define a fonte de dados para o controle.

Em seguida estudaremos um pouco mais sobre o controle DataGrid. No Capítulo 10 aprendemos a utilizar o controle DataGrid para efetuar as seguintes ações:

- ◆ Exibir dados em uma página.
- ◆ Ordenar dados.
- ◆ Fazer paginação.

Neste capítulo aprenderemos a utilizar o controle DataGrid para efetuar as seguintes operações:

- ◆ Filtrar dados.
- ◆ Criar as colunas do DataGrid manualmente.

Veremos que com poucas linhas de código somos capazes de realizar operações que, com o ASP 3.0, demandam uma boa quantidade de codificação. Uma das grandes vantagens do ASP.NET é a disponibilidade de um conjunto de controles mais poderoso e flexível. Para que possamos usufruir destas vantagens é importante que saibamos utilizar estes controles. Com o estudo feito neste capítulo, mais o que foi visto nos Capítulos 7, 8, 9 e 10, o leitor terá um amplo entendimento dos novos controles do ASP.NET. Porém o assunto é bastante extenso; são centenas de métodos e propriedades. A melhor fonte de informações para você aprofundar seus estudos é a documentação do Framework .NET.

Antes de iniciarmos o estudo dos Data-bound controls, iremos apresentar o conceito de Data Binding, e veremos alguns conceitos que se aplicam a todo

controle que pode ser associado a uma fonte de dados. Os conceitos vistos nesta parte inicial são a base para que possamos estudar os Data-bound controls.

## Um “tal de” Data Binding

Data Binding é, sem dúvidas, uma das características de ASP.NET que mais pouparam tempo do programador. Tarefas que exigiam dezenas de linhas de código com as versões anteriores de ASP, agora podem ser feitas configurando-se umas poucas propriedades dos Data-bound controls.

Com o ASP.NET, o processamento e as funções de Data Binding são executados no Servidor. O resultado que retorna é HTML compatível com qualquer navegador, o que elimina a limitação das técnicas de Data Binding no lado cliente.

Com a manutenção de estado automática e as operações de PostBack e round-trip (descritas anteriormente), o funcionamento da página é transparente para o usuário, ou seja, do ponto de vista de quem está utilizando a página pouca importância tem onde está ocorrendo o processamento; desde que o desempenho seja satisfatório e o acesso possa ser feito de qualquer navegador disponível. Para que tudo isso seja possível, as novas características do Framework .NET, como PostBack e round-trip de páginas são fundamentais.

Antes de explicarmos em detalhe a sintaxe para Data Binding, vamos ver um exemplo prático onde fazemos uma comparação entre a maneira de construir um controle do tipo Lista de Opções com ASP 3.0, onde as opções são criadas, automaticamente, a partir de um banco de dados; e a maneira de construir a mesma lista, utilizando Data Binding com ASP.NET.

### À Maneira Antiga: Criando uma Lista Dinâmica com ASP 3.0

Neste exemplo construiremos uma lista com os nomes de todos os países para os quais existem Pedidos. A lista será construída a partir da tabela Pedidos do banco de dados C:\Meus documentos\NorthWind.mdb.

Na Listagem 11.1 temos o código para o exemplo proposto.

#### Listagem 11.1 – Uma lista dinâmica com ASP 3.0 – Chap11ex1.asp.

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
<TITLE>Lista dinâmica de Países.</TITLE>
```

**IMPORTANTE:** Os controles que utilizaremos neste capítulo fazem parte dos chamados Web Server Controls. Na prática isso significa que estes controles são processados no servidor e não no Cliente. Esta é uma grande vantagem, pois as operações de Data Binding irão funcionar corretamente, independente do navegador que estiver sendo utilizado. Antes do Framework .NET, utilizávamos algumas técnicas de Data Binding no cliente. O problema desta abordagem é que estas técnicas são dependentes do navegador que está sendo utilizado. Por exemplo, algumas técnicas de Data Binding utilizando DHTML que funcionam no Internet Explorer podem não funcionar (e provavelmente não funcionarão) no Netscape e vice-versa.

```
</HEAD>
```

```
<BODY>
```

```
<%
```

```
'O Primeiro passo é criar a conexão com o Banco de dados.
```

```
'Para isto crio um objeto do tipo Connection.
```

```
'Cria um Objeto do Tipo ADODB.Connection
```

```
Set conn=Server.CreateObject("ADODB.Connection")
```

```
'Agora abro uma conexão com o arquivo nwind.mdb
```

```
'utilizando OLE DB.
```

```
'O próximo comando deve estar todo em uma única linha.
```

```
conn.ConnectionString = "PROVIDER=MICROSOFT.JET.OLEDB.4.0;DATA SOURCE=c:\Meus
documentos\NorthWind.mdb"
```

```
conn.Open
```

```
'Agora criamos um Objeto RecordSet.
```

```
'Este Objeto irá acessar o campo PaísDeDestino
```

```
'da tabela Pedidos.
```

```
Set Paises = Server.CreateObject("ADODB.Recordset")
```

```
'O próximo comando deve estar todo em uma única linha.
```

```
Paises.Open "Select PaísDeDestino from Pedidos Group By PaísDeDestino Order By
PaísDeDestino", conn, 3, 3
```

```
'Neste ponto tenho o objeto Paises ligado com a tabela
```

```
'Pedidos do banco de dados NorthWind.mdb
```

```
%>
```

```
<P>Paises para o quais existem pedidos!!!</P>
```

```

<FORM action=Chap11ex1.asp method=post>

<SELECT id=listapaises name=listpaises>

<%
'Agora construo a lista de opções a partir dos
'dados obtidos da tabela Pedidos.
'Para cada produto obtido, crio uma nova opção
'na lista.

Do While Not Paises.EOF

 'O próximo comando deve estar todo em uma única linha.
Response.Write "<OPTION value=" & Chr(34) & Paises.Fields("PaísDeDestino") &
Chr(34) & ">" & Paises.Fields("PaísDeDestino")& "</OPTION>"

 Paises.MoveNext
Loop

%>

</SELECT>

<HR>

<INPUT type="submit" value="Paises" id=Localizar name=Localizar>

</FORM>

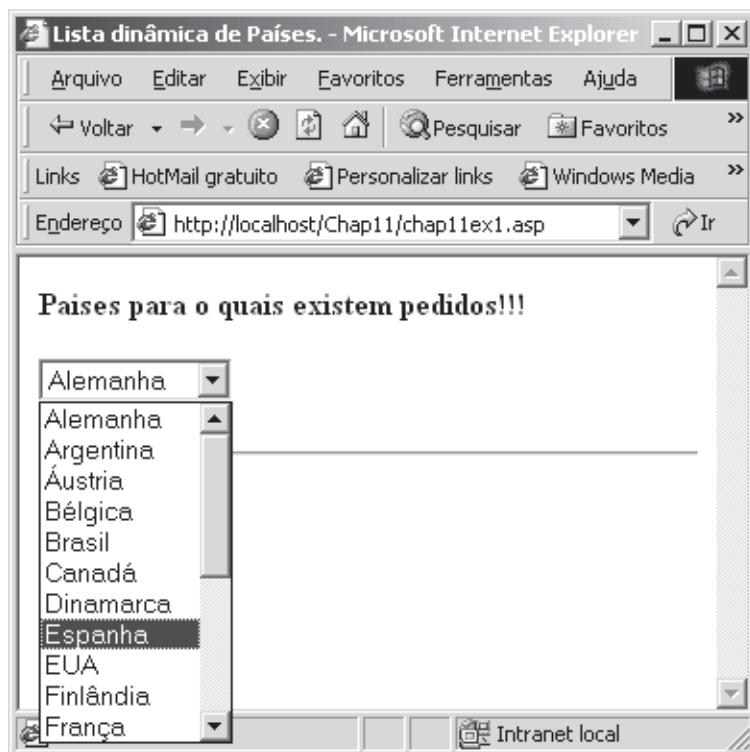
</BODY>
</HTML>

```

Digite o código da Listagem 11.1 e salve o mesmo em um arquivo chamado chap11ex1.asp, na pasta chap11, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6. É importante que a extensão seja .asp e não .aspx, pois trata-se de uma página com código ASP 3.0 e não ASP.NET.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap10/chap11ex1.asp>

Ao carregar a página você obtém o resultado indicado na Figura 11.1.



**Figura 11.1: Uma lista dinâmica com ASP 3.0.**

Além da conexão, criamos um laço Do While para percorrer todos os registros do RecordSet e construir uma opção da lista para cada registro. Com ASP 3.0 não temos muitas alternativas. Agora vamos demonstrar como a utilização de Data Binding torna as coisas bem mais fáceis.

## A Evolução: Data Binding com ASP.NET.

Neste exemplo construiremos uma lista com os nomes de todos os países para os quais existem Pedidos. A lista será construída utilizando um Web Server Control do tipo DropDownList. Para ligar a fonte de dados com o controle, simplesmente faremos uso do método DataBind do controle.

### Listagem 11.2 – Uma lista dinâmica com ASP.NET – Chap11ex2.aspx.

Na Listagem 11.2 temos o código para o exemplo proposto.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>

<html>

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

```
protected void Page_Load(Object Src, EventArgs E)

{

 // Crio uma conexão com o banco de dados do Microsoft Access.
 // C:\Meus documentos\NorthWind.mdb.

 String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
 "DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";

 OleDbConnection MinhaConexão = new OleDbConnection(DefineConexão);

 // Utilizamos um objeto DataAdapter para executar um comando SQL,
 // o qual retorna todos os dados da tabela "Clientes".

 OleDbDataAdapter MeuComando = new OleDbDataAdapter("SELECT PaísDeDestino" +
 " FROM Pedidos Group By PaísDeDestino Order By PaísDeDestino", MinhaConexão);

 // Criamos e preenchemos um objeto DataSet.
 // Observe que não temos mais o objeto Recordset,
 // como era de praxe com o ASP 3.0.

 DataSet ds = new DataSet();

 // Utilizo o método Fill do objeto DataAdapter, para preencher
 // o objeto DataSet, com os dados retornados pelo comando SQL.

 MeuComando.Fill(ds);

 // Conectamos um controle DropDownList com o DataSet criado anteriormente.
 // MinhaLista é o id (nome) de um controle do tipo
 // DropDownList que está na seção de apresentação da página.

 DataView source = new DataView(ds.Tables[0]);

 // Ligo o objeto DataView a um controle do tipo
```

```
// DropDownList - MinhaLista.

MinhaLista.DataSource = source ;
MinhaLista.DataBind();
}

</script>

<body>

<h3>Lista Dinâmica com ASP.NET!!!</h3>

<form runat=server>

 <asp:DropDownList
 id="MinhaLista"
 runat="server"
 DataTextField="PaísDeDestino"
 >

 </asp:DropDownList>

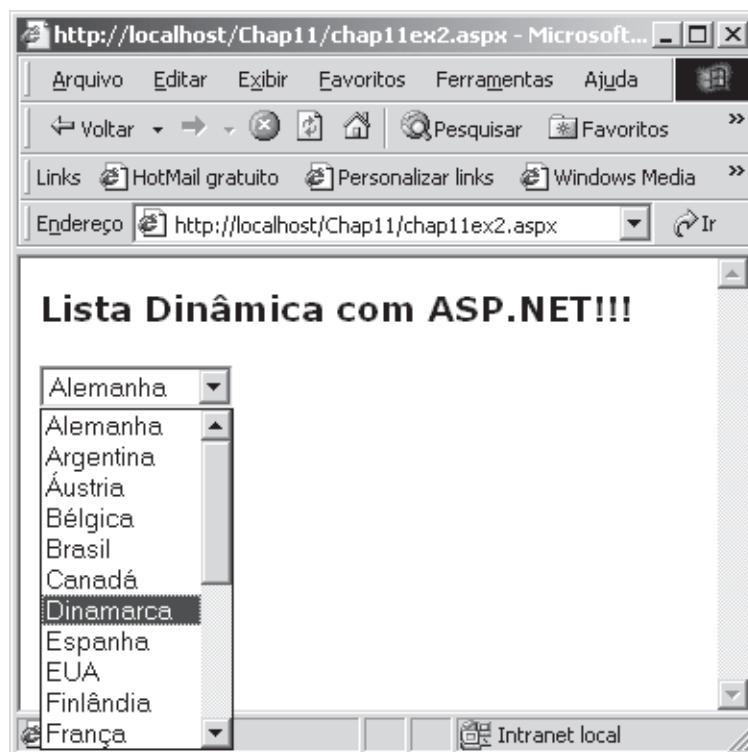
</form>

</body>
</html>
```

Digite o código da Listagem 11.2 e salve o mesmo em um arquivo chamado chap11ex2.aspx, na pasta chap11, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap10/chap11ex2.aspx>

Ao carregar a página você obtém o resultado indicado na Figura 11.2.



**Figura 11.2: Uma lista dinâmica com ASP.NET.**

Observe a simplicidade para criar a lista. Na seção de código definimos a propriedade DataSource do controle MinhaLista, que é um controle do tipo DropDownList. Depois chamamos o método DataBind deste controle:

```
MinhaLista.DataSource = source ;
MinhaLista.DataBind();
```

Na seção de apresentação basta criar o controle MinhaLista e definir a sua propriedade DataTextField como sendo igual ao nome do campo a ser exibido na lista:

```
<asp:DropDownList
 id="MinhaLista"
 runat="server"
 DataTextField="PaísDeDestino"
 >
</asp:DropDownList>
```

Rápido, simples, intuitivo e sem a necessidade de criar um laço para percorrer todos os registros do DataView. Este pequeno exemplo ilustra o quanto pode ser melhorada a produtividade do programador com a utilização dos controles avançados do ASP.NET.

Agora que já vimos um exemplo em ação, vamos estudar um pouco mais detalhadamente os conceitos de DataBinding, para depois estudarmos os demais controles Data-bound controls.

## Data Binding de Valores Simples

A idéia básica do Data Binding é fazer com que, ao ser processada a página ASP.NET, um ou mais valores sejam retornados em posições específicas. Pode ser um conjunto de valores retornados para um controle como o DropDownList que utilizamos no exemplo da Listagem 11.2, ou pode ser um único valor. Por exemplo, podemos fazer com que o rótulo de um controle seja baseado no valor contido em outro controle. Toda vez que a página for carregada e o método DataBind for chamado, o valor do rótulo será atualizado.

O exemplo do DropDownList, da Listagem 11.2, onde são exibidos diversos valores, é conhecido como “repeated-value-binding”. Quando a ligação é feita com um único valor, temos o exemplo de um “single-value-binding”. Vamos analisar a sintaxe e alguns exemplos para “single-value-binding.”

### Sintaxe Para o Data Binding

Podemos fazer o Data Binding com qualquer propriedade de um controle, utilizando a seguinte sintaxe:

```
<%# fonte de dados %>
```

Onde a fonte de dados pode ser de três tipos diferentes:

- ◆ O nome de uma propriedade: <%# nome de propriedade %>
- ◆ Uma chamada de método: <%# método(param1,param2,..., paramn) %>
- ◆ Uma expressão: <%# expressão %>

Qualquer uma das situações acima descritas deve retornar um único valor, o qual é ligado a uma propriedade do controle.

Vamos a um exemplo simples. Criaremos uma página ASP.NET com dois controles. Um do tipo TextBox e outro do tipo Label. O Texto contido no controle Label será definido pelo valor digitado no controle TextBox. O valor padrão inicial do controle TextBox é: Valor Inicial. Vamos ao exemplo, depois às explicações.

Na Listagem 11.3 temos o código para o exemplo proposto.

#### Listagem 11.3 – Um exemplo de single-value-binding.

```
<html>
<script language="C#" runat="server">
 void Page_Load(Object Src, EventArgs E)
 {
 Page.DataBind();
 }
</script>
<body>
 <form runat="server">
```

```

<h3>Exemplo de single-value-binding!!! </H3>

<HR>

<asp:TextBox
 id="CaixaTexto"
 Text="Valor Inicial"
 runat="server"
 />

<HR>

Rótulo:
<asp:Label
 id="RotuloTexto"
 Text="<%# CaixaTexto.Text %>"
 runat="server"
 />

</form>
</body>
</html>

```

Digite o código da Listagem 11.3 e salve o mesmo em um arquivo chamado chap11ex3.aspx, na pasta chap11, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap10/chap11ex3.aspx>

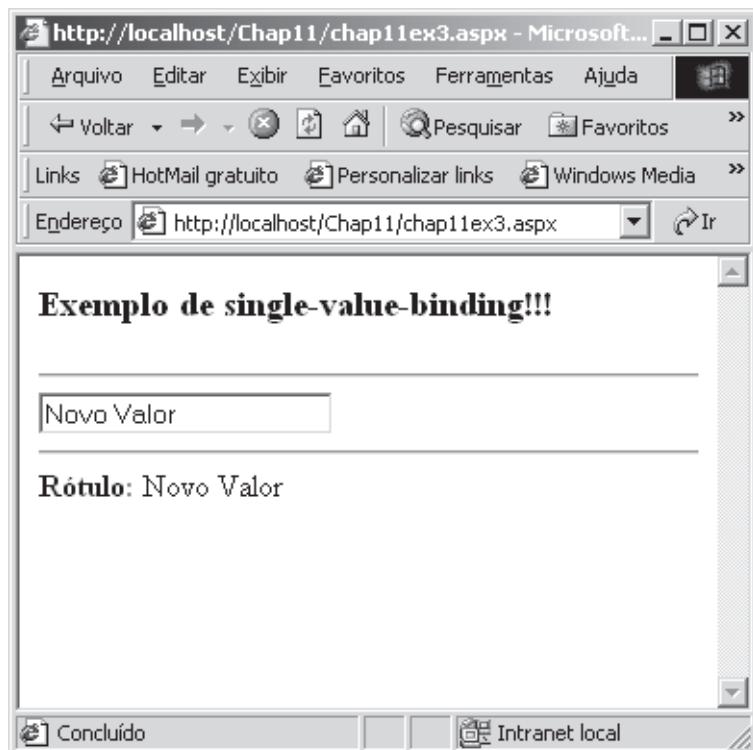
Observe que o rótulo vem preenchido com o valor inicial da caixa de texto: Valor Inicial. Clique na caixa de texto e digite: “Novo Valor” e pressione Enter. Observe que o rótulo é alterado para refletir o valor digitado na caixa de texto, conforme indicado na Figura 11.3.

A ligação do valor do rótulo, com a propriedade Text, da caixa de texto é feita com o seguinte código:

```

<asp:Label
 id="RotuloTexto"
 Text="<%# CaixaTexto.Text %>"
 runat="server"
 />

```



**Figura 11.3: Um exemplo de single-value-binding.**

Mais especificamente com a definição da propriedade Text, onde colocamos uma expressão de ligação que aponta para a propriedade Text do controle CaixaTexto:

```
Text="<%# CaixaTexto.Text %>"
```

Podemos fazer esta ligação para qualquer propriedade, desde que o valor retornado seja compatível com o tipo esperado pela propriedade.

Também é de fundamental importância observar a chamada do método DataBind do objeto Page, o qual é feito no evento Load da Página:

```
void Page_Load(Object Src, EventArgs E)
{
 Page.DataBind();
}
```

Sem esta chamada, a ligação não seria feita. O método DataBind, da classe Page, faz a ligação para a página ASP.NET e para todos os controles contidos na página. No nosso exemplo, o efeito prático é o mesmo que se tivéssemos chamado o método DataBind, do controle RotuloTexto – RotuloTexto.DataBind( ).

Ao invés da propriedade de um controle, poderíamos fazer a ligação com uma propriedade da página. Por exemplo, se quisermos que o controle RotuloTexto exiba o valor da propriedade EnableViewState da página, podemos fazer a seguinte ligação:

```
<asp:Label
```

```

 id="RotuloTexto"
 Text="<%# Page.EnableViewState %>"
 runat="server"
/>

```

## Data Binding de Múltiplos Valores. “repeated-value-binding”

Quando um controle é ligado a uma fonte de dados como um DataView, temos um exemplo de repeated-value-binding. No exemplo da Listagem 11.2, apresentamos o exemplo onde os valores de um controle do tipo DropDownList são obtidos a partir de um objeto DataView. Existem diversos controles capazes de receber dados de um objeto DataView ou DataReader, se for o caso. Neste capítulo estudaremos os seguintes controles:

- ◆ CheckBoxList
- ◆ DataList
- ◆ DropDownList
- ◆ RadioButtonList
- ◆ Repeater

O controle DataGrid também recebe dados de um DataView, conforme já vimos nos exemplos do Capítulo 10. No final deste capítulo estudaremos mais algumas características do controle DataGrid.

### O Controle CheckBoxList

Este controle permite que sejam exibidos diversos checkbox, com base em dados retornados por um objeto DataView, um objeto DataReader, um objeto DataSet, um ArrayList ou uma coleção. Por exemplo, se fizermos o controle baseado em um DataView que retorna o nome de 10 produtos, serão criados 10 CheckBox, uma com o nome de cada produto. Para dar um melhor alinhamento aos checkbox que são construídos dinamicamente, o controle CheckBoxList utiliza uma tabela. Podemos controlar a aparência desta tabela, utilizando algumas propriedades do controle CheckBoxList.

Sintaxe para o controle CheckBoxList:

```

<asp:CheckBoxList
 id="identificação_no_código"
 AutoPostBack="True|False"
 CellPadding="Distância entre as bordas e o conteúdo das células. "
 DataSource='<% Fonte dos dados, normalmente definida no código da página. %>'
 DataTextField="Campo ou coluna a partir do qual é obtido o texto de cada
 CheckBox"
 DataValueField="Campo ou coluna a partir do qual é obtido o valor de cada
 CheckBox"
 RepeatColumns="Define o número controles por linha"
 RepeatDirection="Vertical|Horizontal"
 RepeatLayout="Flow|Table"

```

```
 TextAlign="Right | Left"
 OnSelectedIndexChanged="Evento que executa quando um checkbox é alterado"
 runat="server"
 >

 <asp:ListItem
 value="value"
 selected="True | False">
 Items adicionados estaticamente.
 </asp:ListItem>

</asp:CheckBoxList>
```

O controle CheckBoxList possui uma coleção chamada Items. Esta coleção contém os elementos individuais do controle, os quais podem ser acessados através de código de programação. Para determinar quais itens foram selecionados, podemos fazer um loop através dos elementos da coleção Items.

O valor padrão da propriedade RepeatLayout é Table, o que faz com que os diversos Check Box sejam arranjados na forma de uma tabela.

Exemplo: Vamos utilizar um controle CheckBoxList que exibe todas as cidades do Brasil, para as quais já foram enviados Pedidos. Obteremos esta informação a partir da tabela Pedidos, do banco de dados NorthWind.mdb. Utilizaremos o layout no formato de uma tabela, para o controle CheckBoxList. Também utilizamos algumas propriedades herdadas da classe CheckBoxList Class.

**NOTA:** Existem diversas outras propriedades para este controle. Para maiores informações consulte a classe **CheckBoxList Class**, do namespace **System.Web.UI.WebControls**.

Na Listagem 11.4 temos o código para o exemplo proposto.

#### Listagem 11.4 – Um exemplo do controle CheckBoxList.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>

<html>

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

protected void Page_Load(Object Src, EventArgs E)
{

```

```
// Crio uma conexão com o banco de dados do Microsoft Access.
// C:\Meus documentos\NorthWind.mdb.

String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
 "DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";

OleDbConnection MinhaConexão = new OleDbConnection(DefineConexão);

// Utilizamos um objeto DataAdapter para executar um comando SQL,
// o qual retorna todos os dados da tabela "Clientes".

OleDbDataAdapter MeuComando = new OleDbDataAdapter("SELECT CidadeDeDestino"
+ " FROM Pedidos Group By CidadeDeDestino,PaísDeDestino HAVING PaísDeDestino='Brasil'
"
+ "Order By CidadeDeDestino", MinhaConexão);

// Criamos e preenchemos um objeto DataSet.

DataSet ds = new DataSet();

// Utilizo o método Fill do objeto DataAdapter, para preencher
// o objeto DataSet, com os dados retornados pelo comando SQL.

MeuComando.Fill(ds);

// Criamos um objeto DataView ligado com a primeira
// tabela, da coleção de tabelas, do objeto ds.

DataView source = new DataView(ds.Tables[0]);

// Para podermos comparar os resultados, vou utilizar um controle
// DropDownList e um controle CheckBoxList.
// *****
// Ligo o objeto DataView a um controle do tipo
// DropDownList.

MinhaLista.DataSource = source ;
```

```
MinhaLista.DataBind();

// Ligo o objeto DataView a um controle do tipo
// CheckBoxList.

MeusCheckBox.DataSource = source;
MeusCheckBox.DataBind();

}

</script>

<body>

<h3>Lista e CheckBox dinâmicos com ASP.NET!!!</h3>

<form runat=server>

 <asp:DropDownList
 id="MinhaLista"
 runat="server"
 DataTextField="CidadeDeDestino"
 >

 </asp:DropDownList>

 <asp:CheckBoxList
 id="MeusCheckBox"
 CellPadding="2"
 DataTextField="CidadeDeDestino"
 DataValueField="CidadeDeDestino"
```

```

RepeatColumns="2"
RepeatDirection="Vertical"
RepeatLayout="Table"
TextAlign="Right"
BackColor="#c0c0c0"
BorderWidth="2"
Font-Bold="True"
BorderColor="Blue"
runat="server"

>

</asp:CheckBoxList>

</form>

</body>
</html>

```

Digite o código da Listagem 11.4 e salve o mesmo em um arquivo chamado chap11ex4.aspx, na pasta chap11, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap11/chap11ex4.aspx>

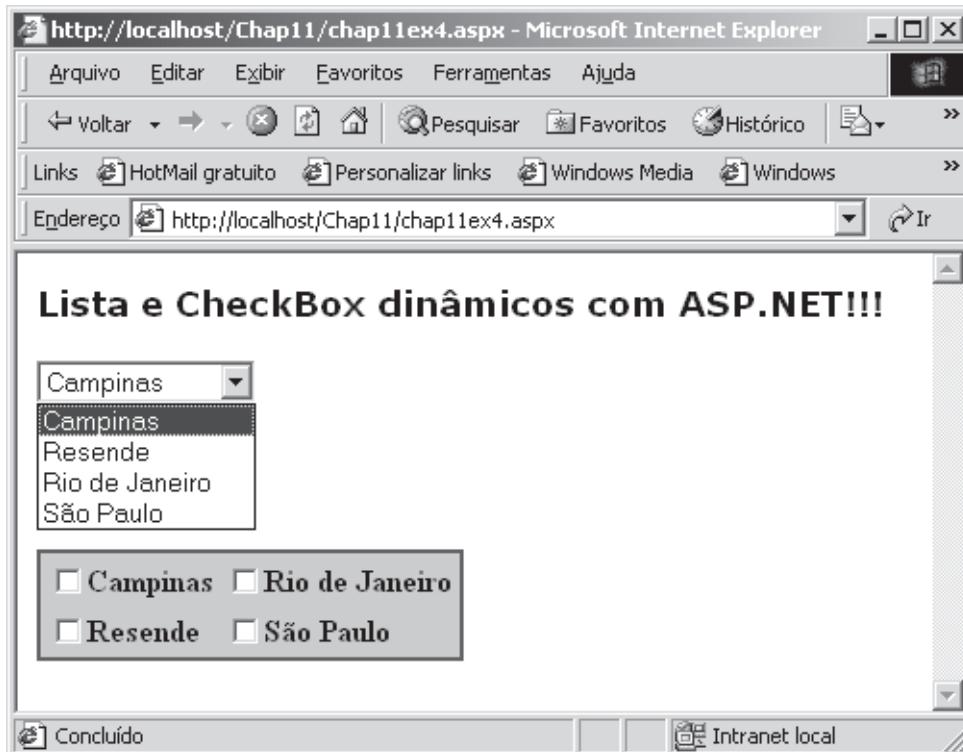


Figura 11.4: O controle CheckBoxList.

Você obtém uma lista com o nome das cidades do Brasil para as quais existem pedidos e um grupo de controles do tipo CheckBox, com um controle para cada cidade, conforme indicado na Figura 11.4.

Comentários sobre o código do exemplo:

- ◆ Para retornar apenas o nome das cidades do Brasil, para as quais houve pedidos, tivemos que lançar mão de um comando SQL um pouco sofisticado:

```
OleDbDataAdapter MeuComando = new OleDbDataAdapter("SELECT CidadeDeDestino"
+ " FROM Pedidos Group By CidadeDeDestino, PaísDeDestino HAVING
PaísDeDestino='Brasil'"
+ "Order By CidadeDeDestino", MinhaConexão);
```

Sem dúvidas, o ASP.NET é uma evolução em relação ao ASP 3.0, mas o “bom e velho SQL” está sempre presente e necessário. Precisamos conhecer os comandos básicos e avançados da linguagem SQL. Um detalhe importante é que existem pequenas diferenças em relação aos comandos SQL para o Microsoft Access e para o SQL Server. No Anexo III veremos mais detalhes sobre a linguagem SQL e as diferenças entre o SQL para o Microsoft Access e para o SQL Server.

- ◆ Na seção de código definimos a propriedade DataSource do controle MeusCheckBox e chamamos o método DataBind do mesmo:

```
MeusCheckBox.DataSource = source;
MeusCheckBox.DataBind();
```

Este procedimento é o mesmo que utilizamos para o controle DropDownList.

- ◆ Na seção de apresentação da página utilizamos um controle CheckBoxList, onde fizemos uso de diversas propriedades deste controle.

- ◆ **DataTextField="CidadeDeDestino"**: Define o nome do campo do DataView, que fornecerá o texto para cada CheckBox.
- ◆ **DataValueField="CidadeDeDestino"**: Define o nome do campo do DataView, que fornecerá o valor relacionado com cada CheckBox, quando este for selecionado.
- ◆ **RepeatColumns="2"**: Estamos utilizando um Layout de Tabela (RepeatLayout="Table"). Esta opção define o número de colunas.

Em seguida definimos negrito para a fonte (Font-Bold="True"), definimos a cor de segundo plano das células como cinza (BackColor="#c0c0c0"), o tamanho da borda em 2 pixels (BorderWidth="2") e a cor da borda azul (BorderColor="Blue").

Como detectar as opções que foram selecionadas em um controle CheckBoxList?

Para determinar as opções que foram selecionados em um controle CheckBoxList, podemos percorrer a coleção Items do controle e testar se determinado item foi selecionado. Por exemplo, se tivermos um controle chamado OpcoesDeCartao e quisermos determinar se o primeiro elemento foi selecionado, podemos fazer o seguinte teste:

```

if (OpcoesDeCartao.Items[0].Selected)
{
 Comando 1
 Comando 2
 ...
 Comando n
}

```

Exemplo: Vamos criar um exemplo, onde temos um controle CheckBoxList, onde as opções foram criadas de uma maneira estática. Cada vez que o usuário clica em uma opção é disparado o evento OnSelectedIndexChanged. Criaremos código para este evento, para atualizar o controle que exibe as opções selecionadas.

Na Listagem 11.5 temos o código para o exemplo proposto.

#### Listagem 11.5 – Detectando as opções selecionadas.

```

<%@ Page Language="C#" %>

<html>
 <head>

 </head>
 <body>

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

 void Verifica_Selecionados(Object sender, EventArgs e)

 {
 Mensagem.Text = "Selected Item(s):" + "
" + "
";

 for (int i=0; i<MinhasOpcoes.Items.Count; i++)
 {

 if (MinhasOpcoes.Items[i].Selected)
 Mensagem.Text = Mensagem.Text + MinhasOpcoes.Items[i].Text + "
";
 }
 }
 </script>
 </body>
 </head>
</html>

```

```
<form runat="server">

 <asp:CheckBoxList
 id="MinhasOpcoes"
 runat="server"
 AutoPostBack="True"
 CellPadding="5"
 CellSpacing="5"
 RepeatColumns="2"
 RepeatDirection="Vertical"
 RepeatLayout="Flow"
 TextAlign="Right"
 OnSelectedIndexChanged="Verifica_Selecionados">

 <asp:ListItem>Item 1</asp:ListItem>
 <asp:ListItem>Item 2</asp:ListItem>
 <asp:ListItem>Item 3</asp:ListItem>
 <asp:ListItem>Item 4</asp:ListItem>
 <asp:ListItem>Item 5</asp:ListItem>
 <asp:ListItem>Item 6</asp:ListItem>

 </asp:CheckBoxList>

 <asp:label
 id="Mensagem"
 runat="server"
 />

</form>

</body>
</html>
```

Digite o código da Listagem 11.5 e salve o mesmo em um arquivo chamado chap11ex5.aspx, na pasta chap11, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap10/chap11ex5.aspx>

Clique no Item 1 e observe que, automaticamente o Item selecionado é informado no controle Label. Selecione o Item 5. Novamente o Item selecionado é informado. Marque o Item 4, mesma coisa. Agora desmarque o Item 4, o Label é atualizado, conforme indicado na Figura 11.5.

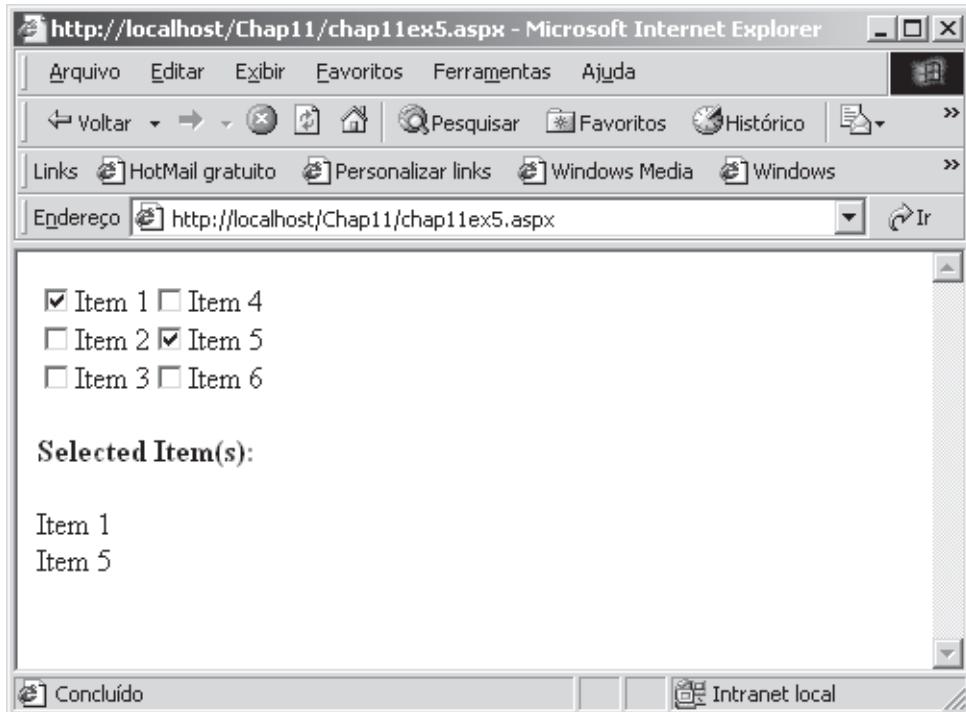


Figura 11.5: Exemplo da coleção **Items**.

Comentários sobre o código do exemplo:

- ♦ Na definição do controle, utilizamos o evento **OnSelectedIndexChanged**. Este evento é disparado toda vez que clicamos em uma das opções do controle **CheckBoxList**. Em resposta a este evento, criamos o procedimento **Verifica\_Selecionados**, na seção de código da página.
- ♦ O procedimento **Verifica\_Selecionados** define a propriedade **Text** do label **Mensagem**, para exibir quais opções estão atualmente selecionadas. Utilizamos um laço **For** que varia de 0 até o número de opções do controle. Para obter o número de opções do controle, utilizamos a propriedade **Count** da coleção **Items**:

#### **MinhasOpcoes.Items.Count**

No interior do laço for, se uma determinada opção estiver selecionada:

```
if (MinhasOpcoes.Items[i].Selected)
```

concatenamos o valor desta opção à propriedade **Text** do label **Mensagem**, mais uma tag **<BR>** para uma quebra de linha. A tag **<BR>** faz com que cada opção selecionada seja exibida em uma linha diferente.

No final do laço for, o Label **Mensagem** exibe as opções selecionadas, uma em cada linha. Este é exatamente o resultado desejado.

## O Controle DropDownList

Utilizamos este controle nos exemplos Chap11ex2.aspx e Chap11ex4.aspx. O controle DropDownList é utilizado para a criação de uma lista de opções. A lista pode ser definida de uma maneira estática, utilizando uma série de controles ListItem ou pode ser associada a uma fonte de dados (Data Binding). Ao associarmos o controle DropDownList a uma fonte de dados, as opções da lista serão criadas, automaticamente, a partir dos dados retornados por um objeto DataView ou DataReader.

A sintaxe para o controle DropDownList:

```
<asp:DropDownList
 id="identificação_no_código"
 runat="server"

 DataSource=<% fonte de dados %>
 DataTextField="Campo da fonte de dados com os rótulos para os itens da lista."
 DataValueField="Campo da fonte de dados com os valores para os itens da lista."
 AutoPostBack="True|False"
 OnSelectedIndexChanged="Método que executa quando um elemento da lista é
 selecionado">

 <asp:ListItem
 value="valor do item da lista."
 selected="True|False">
 Texto do item da lista.
 </asp:ListItem>

</asp:DropDownList>
```

O controle DropDownList é derivado da classe DropDownList, do namespace System.Web.UI.Controls. Esta classe possui uma série de propriedades que permitem a definição dos aspectos visuais do controle. Na Tabela 11.1, temos a definição das principais propriedades para o controle DropDownList.

**Tabela 11.1** Principais propriedades do controle DropDownList.

Propriedade	Descrição
BackColor	Utilizada para definir ou retornar a cor de segundo plano do controle.
BorderColor	Utilizada para definir ou retornar a cor de borda do controle.
BorderWidth	Utilizada para definir ou retornar o tamanho, em pixels, das bordas do controle.
DataSource	Define a fonte de dados a partir da qual são gerados os itens do controle.
DataTextField	Define o campo da fonte de dados, que define o texto que será exibido para cada item.
Font	Define ou retorna informações sobre as características da fonte do controle.
AccessKey	Define um atalho de teclado para colocar o foco no controle.

O principal método deste controle é DataBind(), o qual faz a ligação do controle com uma fonte de dados. Poderíamos redefinir o controle DropDownList do exemplo Chap11ex2.aspx, da seguinte maneira:

```
<asp:DropDownList
 id="MinhaLista"
 runat="server"
 DataTextField="PaísDeDestino"
 BackColor="#c0c0c0"
 ForeColor="Blue"
 Font-Bold="True"
 Font-Italic="True"
>
```

Com estas alterações a aparência do controle fica conforme indicado na Figura 11.6.

## O Controle RadioButtonList

Com este controle podemos criar um grupo de Radio Buttons, dinamicamente, a partir de uma fonte de dados. Por exemplo, podemos gerar um grupo com um Radio Button para cada cidade da Alemanha, para a qual temos pedidos na tabela Pedidos. Somente um RadioButton do controle pode estar selecionado ao mesmo tempo. O comportamento é o mesmo do controle RadioButton, com a diferença de que o controle RadioButtonList suporta Data Binding para a geração dinâmica dos seus itens, a partir de uma fonte de dados.

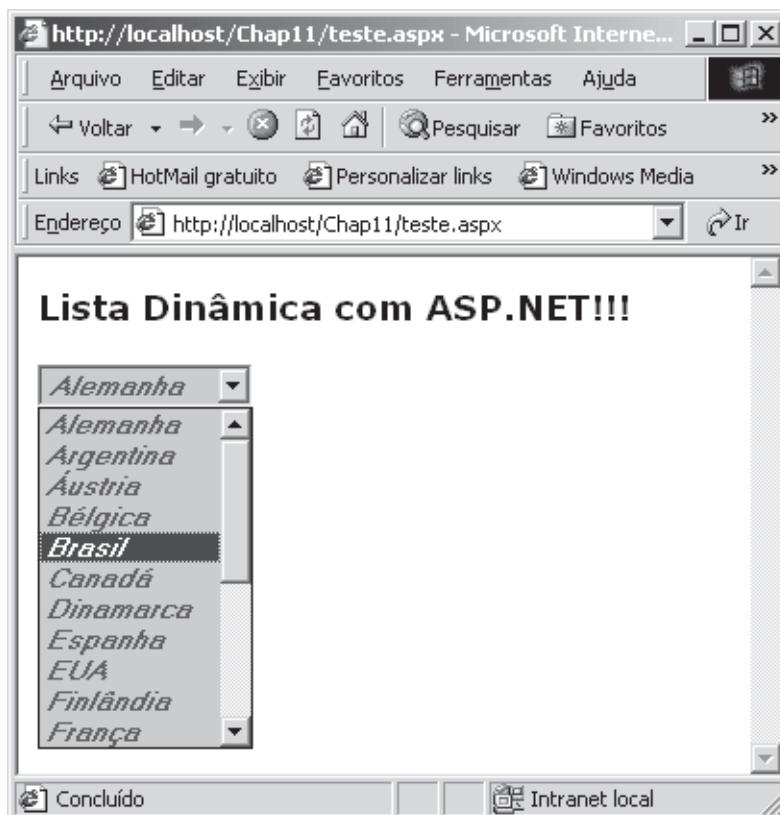


Figura 11.6: Propriedades do controle DropDownList.

A sintaxe para este controle:

```
<asp:RadioButtonList
 id="identificação_no_código"
 AutoPostBack="True|False"
 CellPadding="Distância entre as bordas e o conteúdo das células. "
 DataTextField="Campo a partir do qual é obtido o texto de cada RadioButton"
 DataValueField="Campo a partir do qual é obtido o valor de cada RadioButton"
 RepeatColumns="Define o número controles por linha"
 RepeatDirection="Vertical|Horizontal"
 RepeatLayout="Flow|Table"
 TextAlign="Right|Left"
 OnSelectedIndexChanged="Evento que executa quando clicamos em um RadioButton"
 runat="server">

 <asp:ListItem
 Text="Texto do item."
 Value="Valor do item."
 Selected="True|False"
 />

</asp:RadioButtonList>
```

O controle CheckBoxList possui uma coleção chamada Items. Esta coleção contém os elementos individuais do controle, os quais podem ser acessados através de código de programação. Para determinar quais items foram selecionados, podemos fazer um loop através dos elementos da coleção Items.

O valor padrão da propriedade RepeatLayout é Table, o que faz com que os diversos Check Box sejam arranjados na forma de uma tabela. Utilizamos a propriedade RepeatColumns para definir o número de controles por linha da tabela.

Exemplo: Vamos utilizar um controle RadioButtonList que exibe todas as cidades da Alemanha, para as quais já foram enviados Pedidos. Obteremos esta informação a partir da tabela Pedidos, do banco de dados NorthWind.mdb. Utilizaremos o layout no formato de uma tabela com três colunas, para o controle RadioButtonList. Também utilizamos algumas propriedades herdadas da classe RadioButtonList Class.

**NOTA:** Existem diversas outras propriedades para este controle. Para maiores informações consulte a classe **RadioButtonList Class**, do namespace **System.Web.UI.WebControls**.

Na Listagem 11.6 temos o código para o exemplo proposto.

## Listagem 11.6 – Um exemplo do controle RadioButtonList.

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

```
<%@ Import Namespace="System.Data.OleDb" %>

<html>

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

 void Page_Load(Object Src, EventArgs E)
 {

 // Crio uma conexão com o banco de dados do Microsoft Access.
 // C:\Meus documentos\NorthWind.mdb.

 String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
 "DATA SOURCE=c:\meus documentos\NorthWind.mdb";

 OleDbConnection MinhaConexão = new OleDbConnection(DefineConexão);

 // Utilizamos um objeto DataAdapter para executar um comando SQL,
 // o qual retorna as cidades da Alemanha.
 // Para maiores informações sobre a linguagem SQL, consulte o Anexo III.

 OleDbDataAdapter MeuComando = new OleDbDataAdapter("SELECT CidadeDeDestino" +
 " FROM Pedidos Group By CidadeDeDestino,PaísDeDestino HAVING PaísDeDestino='Alemanha'" +
 " + "Order By CidadeDeDestino", MinhaConexão);

 // Criamos e preenchemos um objeto DataSet.

 DataSet ds = new DataSet();

 // Utilizo o método Fill do objeto DataAdapter, para preencher
 // o objeto DataSet, com os dados retornados pelo comando SQL.

 MeuComando.Fill(ds);

 // Criamos um objeto DataView ligado com a primeira
 // tabela, da coleção de tabelas, do objeto ds.

 }
}
```

```
 DataView source = new DataView(ds.Tables[0]);

 // Ligo o objeto DataView a um controle do tipo
 // RadioButtonList - MeusBotoes.

 MeusBotoes.DataSource = source;
 MeusBotoes.DataBind();

 }

</script>

<body>

<h3>Radio Button dinâmicos com ASP.NET!!!</h3>

<form runat=server>

<HR>

<asp:RadioButtonList
 id="MeusBotoes"
 CellPadding="2"
 DataTextField="CidadeDeDestino"
 DataValueField="CidadeDeDestino"
 RepeatColumns="3"
 RepeatDirection="Vertical"
 RepeatLayout="Table"
 TextAlign="Right"
 BackColor="#c0c000"
 BorderWidth="2"
 Font-Bold="True"
 BorderColor="Blue"
 runat="server"
 >

</asp:RadioButtonList>
```

```
<HR>
</form>

</body>
</html>
```

Digite o código da Listagem 11.6 e salve o mesmo em um arquivo chamado chap11ex6.aspx, na pasta chap11, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap10/chap11ex6.aspx>

Você obtém o resultado indicado na Figura 11.7.

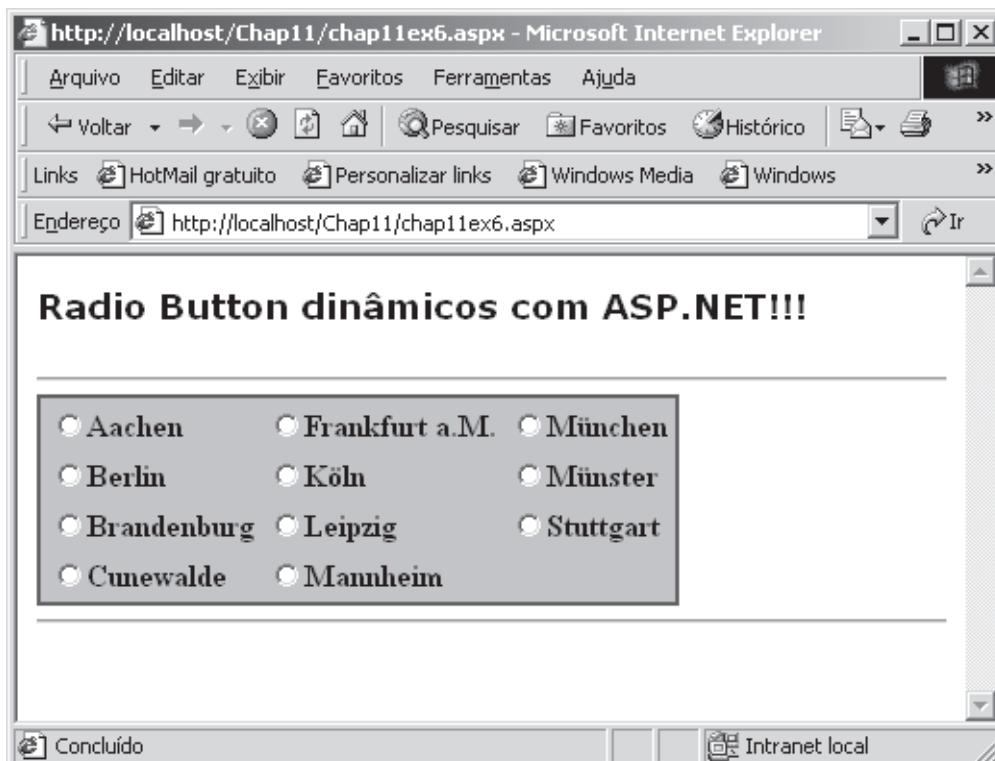


Figura 11.7: O controle RadioButtonList – opções geradas dinamicamente.

Observe que utilizamos algumas propriedades para definir os aspectos visuais do controle:

- ◆ **RepeatColumns="3"**: Define que os controles serão exibidos em uma tabela com três colunas, ou seja, três controles por linha.
- ◆ **RepeatLayout="Table"**: Os controles serão organizados no formato de uma tabela.
- ◆  **TextAlign="Right"**: Alinhamento de texto à direita.
- ◆  **BackColor="#c0c000"**: Cor de segundo plano da tabela, na qual são colocados os controles.
- ◆  **BorderWidth="2"**: Tamanho da borda externa em pixels.
- ◆  **Font-Bold="True"**: Exibe a fonte em negrito.
- ◆  **BorderColor="Blue"**: Define a cor da borda externa.

## Mais um Pouco Sobre o Controle DataGrid

No Capítulo 10 mostramos algumas das capacidades do controle DataGrid, ao mesmo tempo que chamamos a atenção para o fato de ser este um controle bastante poderoso, que nos oferece um grande número de funcionalidades. Vamos aprender, em detalhes, mais duas funcionalidades importantes do controle DataGrid:

- ◆ Ordenar dados.
- ◆ Filtrar dados.

Nos exemplos que iremos apresentar, vamos utilizar, como fonte de dados para o controle DataGrid, um objeto DataView. Desta maneira poderemos utilizar as facilidades de ordenação e filtragem do objeto DataView, para definir quais dados e de que maneira estes dados serão exibidos no DataGrid.

### Ordenação com o Controle DataGrid

O controle DataGrid possui uma propriedade chamada AllowSorting (já descrita no Capítulo 10). Quando esta propriedade é definida em True, os títulos de coluna, do controle DataGrid, são transformados em Hyperlinks. Quando clicamos em um destes links, ocorre um PostBack e o código definido para o evento SortCommand é executado.

Vamos construir um exemplo onde são exibidos todos os pedidos para o Brasil, a partir da tabela Pedidos do Banco de dados NorthWind.mdb. Definiremos a propriedade AllowSorting do controle DataGrid para True. O passo final é criar o código que executa quando o usuário clica no título de uma das colunas. Por exemplo, se o usuário clicar na coluna CidadeDeDestino, devemos ordenar os dados pela CidadeDeDestino.

Na Listagem 11.7 temos o código para o exemplo proposto.

#### Listagem 11.7 – Ordenação com o controle DataGrid.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>

<html>

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

 // Declaro uma variável do tipo String.
 // Esta variável contém o nome da coluna pela qual
 // faremos a ordenação.
 // da página.

 String OrdenaPor;
```

**NOTA:** Para a criação deste exemplo, utilizaremos algumas técnicas avançadas. Todas as técnicas utilizadas serão descritas após o código do exemplo.

```
// No evento Page_Load definimos a ordenação padrão, pelo campo
// NúmeroDoPedido e chamamos o procedimento - BuscaDados().
// A definição do campo padrão de ordenação somente ocorre quando
// a página é carregada pela primeira vez, isto é, quando não for um PostBack.

void Page_Load(Object Src, EventArgs E)
{

 if (Page.IsPostBack)
 {
 // Não faz nada.
 }
 else
 // Define, por padrão, a ordenação por NúmeroDoPedido.
 {

 OrdenaPor = "NúmeroDoPedido";
 }

 // Chamo a rotina que acessa o Banco de dados e preenche o DataGrid.

 BuscaDados();
}

// Rotina que faz a conexão com o banco de dados Northwind.mdb.
// Ordena os dados de acordo com o campo definido na variável OrdenaPor.

void BuscaDados()
{
 // Crio uma conexão com o banco de dados do Microsoft Access.
 // C:\Meus documentos\NorthWind.mdb.

String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
 "DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";
```

```
OleDbConnection MinhaConexão = new OleDbConnection(DefineConexão);

 // Utilizamos um objeto DataAdapter para executar um comando SQL,
 // o qual retorna os pedidos para o Brasil.

OleDbDataAdapter MeuComando = new
OleDbDataAdapter("SELECT NúmeroDoPedido,EndereçoDoDestinatário,"
+ "Frete,CidadeDeDestino,PáisDeDestino FROM Pedidos WHERE PaísDeDestino='Brasil'"
+ "Order By NúmeroDoPedido", MinhaConexão);

 // Criamos e preenchemos um objeto DataSet.

DataSet ds = new DataSet();

 // Utilizo o método Fill do objeto DataAdapter, para preencher
 // o objeto DataSet, com os dados retornados pelo comando SQL.

MeuComando.Fill(ds);

 // Criamos um objeto DataView ligado com a primeira
 // tabela, da coleção de tabelas, do objeto ds.

DataView source = new DataView(ds.Tables[0]);

 // Defino a propriedade Sort, do objeto source, como sendo
 // igual ao valor da String OrdenaPor.

source.Sort=OrdenaPor;

 // Ligo o objeto DataView a um controle do tipo
 // DataGrid - MinhaGrade.

MinhaGrade.DataSource = source;
MinhaGrade.DataBind();

}
```

```

// Rotina que executa em resposta ao evento OnSortCommando
// do controle DataGrid.
// Esta rotina recebe dois argumentos, um do tipo Object e outro
// e outro do tipo DataGridSortCommandEventArgs.

void OrdenaDados(Object sender, DataGridSortCommandEventArgs e)
{
 // Defino o valor da String OrdenaPor, com base na propriedade
 // SortExpression, do argumento "e", o qual é do tipo
 // DataGridSortCommandEventArgs, passado como parâmetro.

 OrdenaPor = e.SortExpression.ToString();

 // Chamo a rotina que faz a conexão para reordenar os dados,
 // com base no novo valor da String OrdenaPor.

 BuscaDados();
}

</script>

<body>

<h3>Ordenação com o controle DataGrid!!!</h3>

<form runat=server>

<H4> Clique no título da coluna para ordenar pelo campo respectivo.</H4>
<HR>

 <ASP:DataGrid
 id="MinhaGrade" runat="server"
 Width="700"
 BackColor="#ccccff"
 BorderColor="black"
 ShowFooter="false"

```

```
 CellPadding=3
 CellSpacing="0"
 Font-Name="Verdana"
 Font-Size="8pt"
 HeaderStyle-BackColor="#aaaadd"
 MaintainState="false"
 AllowSorting="True"
 OnSortCommand="OrdenaDados"

/>
</form>

</body>
</html>
```

Digite o código da Listagem 11.7 e salve o mesmo em um arquivo chamado chap11ex7.aspx, na pasta chap11, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap10/chap11ex7.aspx>

Você obtém uma página com todos os pedidos para o Brasil. Observe que o título das colunas é um link. Clique no título da coluna CidadeDeDestino. Observe que os dados são classificados, em ordem Crescente, pela coluna CidadeDeDestino, conforme indicado na Figura 11.8.

Agora dê um clique no cabeçalho da Coluna Frete e observe que os dados são classificados, em ordem crescente do valor do frete.

Comentários sobre o código do exemplo:

- ◆ A primeira grande diferença que você deve ter notado em relação aos exemplos anteriores é a utilização de diversas rotinas:
  1. Page\_Load
  2. Busca\_Dados
  3. OrdenaDados

Uma página ASP.NET é uma instância da classe Page. Nós vimos nos Capítulos 3, 4 e 5 que um Classe em C# pode ser composta de diversos procedimentos. Neste exemplo estamos criando três procedimentos. Não existe a definição explícita da classe, através da palavra Class. Esta é a única diferença em relação aos exemplos dos Capítulos 3, 4 e 5.

Também criamos uma variável do tipo String chamada OrdenaPor. Como esta variável foi declarada fora de qualquer procedimento, a mesma pode ser acessada em qualquer local da seção de código da página, ou seja, a variável OrdenaPor tem escopo de página:

```
String OrdenaPor;
```

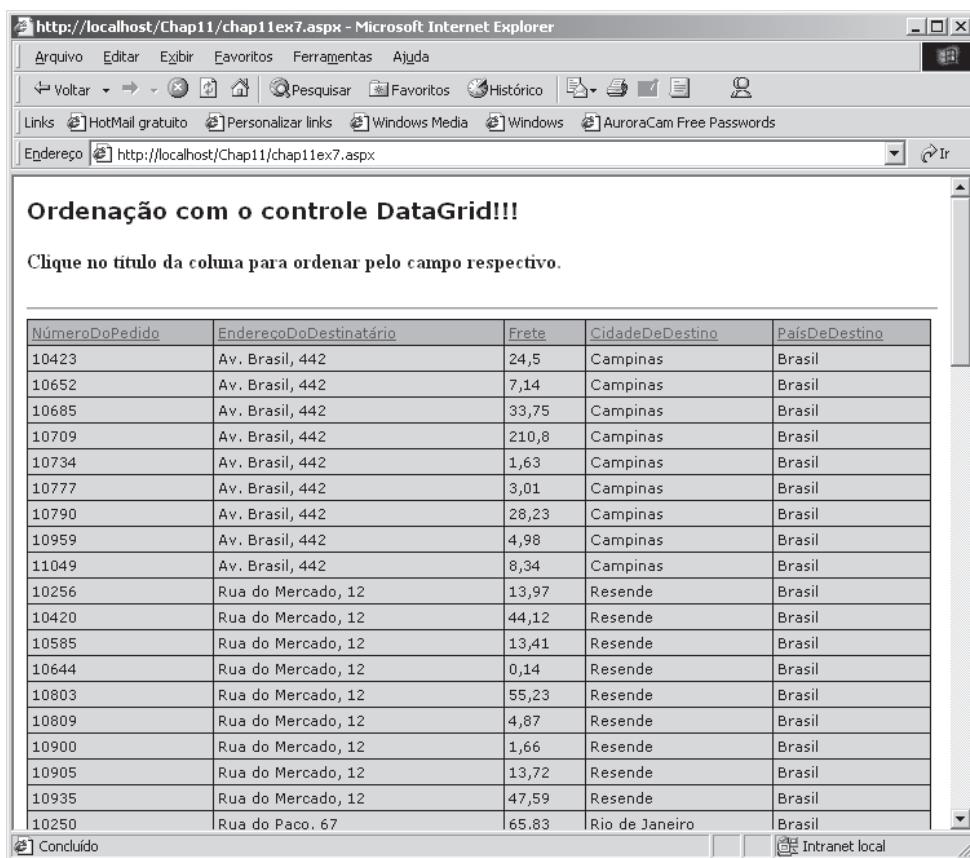


Figura 11.8: Dados ordenados pela coluna CidadeDeDestino.

- ◆ Na seção de apresentação da página, inserimos um controle do tipo DataGrid. Definimos a propriedade AllowSorting=True. Isso faz com que o cabeçalho das colunas seja exibido como um link. Na propriedade OnSortCommand, definimos o nome do procedimento que será executado quando o usuário clicar no link referente ao cabeçalho de uma coluna:

```
AllowSorting="True"
```

```
OnSortCommand="OrdenaDados"
```

- ◆ Vamos entender o procedimento Page\_Load: Este evento, conforme já descrito anteriormente, é executado toda vez que a página for carregada. Quando a página for carregada pela primeira vez, a propriedade Page.IsPostBack é igual a False e, neste caso, a variável OrdenaPor é definida como sendo igual a NúmeroDoPedido.

Fora da estrutura if/else, é chamado o procedimento BuscaDados( ), ou seja, este procedimento será chamado sempre que a página for carregada, independente de ser um PostBack ou não.

- ◆ Vamos entender o procedimento BuscaDados( ): Grande parte do código deste procedimento já é de nosso conhecimento. São comandos para conectar com o banco de dados NorthWind.mdb, executar um comando SQL e retornar os dados em um objeto DataSet, a partir do qual criamos um objeto DataView. O único detalhe adicional que temos neste procedimento é a definição da propriedade Sort, do objeto source, que é um objeto do tipo DataView:

```
source.Sort=OrdenaPor;
```

Neste momento o objeto DataView é ordenado pela coluna, cujo nome está contido na String OrdenaPor.

- ◆ **O elemento principal:** o procedimento OrdenaDados. Este procedimento é executado em resposta ao evento OnSortCommand, o qual acontece toda vez que o usuário clica em um link, de uma das colunas do DataGrid. Vamos acompanhar o que acontece quando, por exemplo, o usuário clica no link da coluna CidadeDeDestino.

1. O evento OnSortCommand é disparado.
2. Em resposta ao evento OnSortCommand, o procedimento OrdenaDados é executado.
3. O procedimento OrdenaDados recebe dois argumentos. O primeiro é um argumento do tipo Object e o segundo do tipo DataGridSortCommandEventEventArgs. Este segundo evento tem uma propriedade chamada SortExpression, a qual retorna o nome da coluna correspondente ao link clicado pelo usuário. No nosso exemplo a expressão:

```
OrdenaPor = e.SortExpression.ToString();
```

retorna CidadeDeDestino, uma vez que o usuário clicou no link da coluna CidadeDeDestino. Com isso conseguimos detectar qual o link clicado e, consequentemente, por qual campo devemos ordenar. Uma vez definido o valor da String OrdenaPor, chamamos o método BuscaDados( ), o qual retornará os dados e irá ordená-los com base no valor definido na String OrdenaPor.

Em resumo, o procedimento OrdenaDados faz o seguinte: define o valor da String OrdenaPor e chama o procedimento BuscaDados.

- ◆ É importante salientar que a possibilidade de criar diferentes procedimentos e a possibilidade de chamarmos um procedimento, dentro do outro, nos dá uma flexibilidade muito grande. Além disso, ao dividirmos o código de nossas páginas ASP.NET em procedimentos, podemos organizá-lo de uma maneira mais intuitiva, o que torna fácil a compreensão e, principalmente, a manutenção do mesmo.
- ◆ Observe que, conhecendo os elementos (eventos, argumentos, etc.) corretos, com poucas linhas de código é possível criar funcionalidades que, com ASP 3.0, exigem uma grande quantidade de código. Mais uma vez conseguimos comprovar o quanto mais produtivo, do ponto de vista do desenvolvedor, é o ASP.NET em relação às versões anteriores.

**NOTA:** Para maiores informações sobre a classe **DataGridSortCommandEventEventArgs**, consulte a documentação do Framework .NET. Esta classe faz parte do namespace **System.Web.UI.WebControls**.

## Filtrando Dados com o Controle DataGrid

Para filtrar os dados de um objeto DataView, utilizamos a propriedade RowFilter. Atribuímos, a esta propriedade, uma expressão de filtragem. Por exemplo, se quisermos que sejam exibidos apenas os pedidos em que o campo PaísDeDestino seja igual a Brasil, utilizamos o seguinte comando:

```
"PaísDeDestino = 'Brasil'"
```

A expressão é a mesma que utilizariamos em um cláusula WHERE da linguagem SQL. Podemos utilizar todos os operadores, funções e critérios válidos como critérios de filtragem.

Exemplo: Vamos apresentar um exemplo, onde destacaremos as seguintes técnicas:

- ◆ Utilização de um comando SQL para retornar dados de duas tabelas.
- ◆ Utilização da propriedade RowFilter, do objeto DataView, para filtrar os dados exibidos em um DataGrid.

**NOTA:** Para maiores detalhes sobre a linguagem SQL, consulte o Anexo III.

O nosso exemplo será composto de um formulário onde temos um controle DataGrid que exibe os campos indicados na Tabela 11.2.

**Tabela 11.2 Campos para o exemplo Chap11Ex8.aspx.**

Campo	De qual tabela?
NúmeroDoPedido	Pedidos
NomeDaEmpresa	Clientes
CidadeDeDestino	Pedidos
PaísDeDestino	Pedidos

Observe que estamos retornando dados de duas tabelas diferentes: Pedidos e Clientes. Neste caso teremos que utilizar uma cláusula JOIN, no comando SQL. As tabelas Pedidos e Clientes são relacionadas através do campo CódigoDoCliente. Este é um relacionamento do tipo Um (tabela Clientes) para vários (tabela Pedidos), o que significa que cada Cliente é cadastrado uma única vez, porém pode fazer vários pedidos. Para maiores informações sobre relacionamentos, consulte o Anexo II. Para maiores informações sobre o SQL consulte o Anexo III.

Além do controle DataGrid, teremos um campo do tipo TextBox, onde o usuário pode digitar o nome ou parte do nome de um Cliente. Ao clicar no botão Pesquisar, o DataGrid passará a exibir somente os pedidos para os clientes cujo nome, atende ao critério digitado.

Na Listagem 11.8 temos o código para o exemplo proposto.

**Listagem 11.8 – Filtragem com o controle DataGrid.**

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>

<html>

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

// Declaro uma variável do tipo String.
// Esta variável contém o nome da coluna pela qual
```

**NOTA:** Neste exemplo vamos manter o código que faz a classificação dos dados quando o usuário clica no título de uma das colunas do DataGrid. Desta forma poderemos filtrar e classificar os dados.

```
// faremos a ordenação.
// Esta variável pode ser acessada em qualquer procedimento da página.

String OrdenaPor;
String FiltraDados;

void Page_Load(Object Src, EventArgs E)
{

 if (Page.IsPostBack)
 {
 // Define o valor da String FiltraDados que,
 // na prática, é a expressão de filtragem dos dados,
 // que será atribuída à propriedade RowFilter.

 FiltraDados="NomeDaEmpresa LIKE '*" + DigitaCriterio.Text + "*'";
 }
 else

 // Define, por padrão, a ordenação por NúmeroDoPedido.
 // Atribui um valor padrão para o campo de pesquisa.
 {
 OrdenaPor = "NúmeroDoPedido";
 DigitaCriterio.Text="a";
 }

 // Chamo a rotina que acessa o Banco de dados e preenche o DataGrid.

 BuscaDados();

}

// Rotina que faz a conexão com o banco de dados Northwind.mdb.
// Ordena os dados de acordo com o campo definido na variável OrdenaPor.
// Filtra de acordo com o critério especificado na String FiltraDados.
```

```

void BuscaDados()
{
 // Crio uma conexão com o banco de dados do Microsoft Access.
 // C:\Meus documentos\NorthWind.mdb.

 String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
 "DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";

 OleDbConnection MinhaConexão = new OleDbConnection(DefineConexão);

 // Utilizamos um objeto DataAdapter para executar um comando SQL,
 // o qual retorna as cidades da Alemanha.

 String ComandoSQL;
 String aux1SQL;
 String aux2SQL;
 String aux3SQL;
 String aux4SQL;

 aux1SQL= "SELECT Pedidos.NúmeroDoPedido, Clientes.NomeDaEmpresa,
Pedidos.CidadeDeDestino";
 aux2SQL= ", Pedidos.PaísDeDestino FROM Clientes";
 aux3SQL= " INNER JOIN Pedidos ON Clientes.CódigoDoCliente = Pedidos.CódigoDoCliente";
 aux4SQL= " ORDER BY Pedidos.NúmeroDoPedido";

 ComandoSQL = aux1SQL + aux2SQL +aux3SQL +aux4SQL;

 OleDbDataAdapter MeuComando = new OleDbDataAdapter(ComandoSQL, MinhaConexão);

 // Criamos e preenchemos um objeto DataSet.
 // Observe que não temos mais o objeto Recordset,
 // como era de praxe com o ASP 3.0.

 DataSet ds = new DataSet();

 // Utilizo o método Fill do objeto DataAdapter, para preencher

```

```
// o objeto DataSet, com os dados retornados pelo comando SQL.

MeuComando.Fill(ds);

// Criamos um objeto DataView ligado com a primeira
// tabela, da coleção de tabelas, do objeto ds.

DataView source = new DataView(ds.Tables[0]);

// Defino a propriedade Sort, do objeto source, como sendo
// igual ao valor da String OrdenaPor.

source.Sort=OrdenaPor;

// Defino a propriedade RowFilter, para aplicar o critério de filtragem.
// definido na variável

source.RowFilter=FiltrarDados;

// Ligo o objeto DataView a um controle do tipo
// DataGrid.

MinhaGrade.DataSource = source;
MinhaGrade.DataBind();

}

// Rotina que executa em resposta ao evento OnSortCommando
// do controle DataGrid.

void OrdenaDados(Object sender, DataGridSortCommandEventArgs e)
{

 // Defino o valor da String OrdenPor, com base na propriedade
 // SortExpression, do argumento e, do tipo
 // DataGridSortCommandEventArgs, passado como parâmetro.
```

```
 OrdenaPor = e.SortExpression.ToString();

 // Chamo a rotina que faz a conexão para reordenar os dados,
 // com base no novo valor da String OrdenaPor.

 BuscaDados();

}

</script>

<body>

<h3>Ordenação com o controle DataGrid!!!</h3>

<form runat=server>

 <H4> Clique no título da coluna para ordenar pelo campo respectivo.</H4>
 <H4> Digite o nome ou parte do nome da empresa no campo Critério e dê
um</H4>
 <H4> clique no botão Pesquisar.</H4>
 <HR>

<Table>

<TR>
 <TD> Critério --> </TD>

 <TD>
 <asp:TextBox
 runat=server
 id="DigitaCriterio"
 Text=>>>
 Font_Face=>>Arial<>
 Font_Size="3"
 Font-Bold="True"
 BackColor="lightblue"
 </asp:TextBox>
 </TD>
</TR>
```

```
 Height="20"

 />

</TD>

<TD>

 <asp:Button

 id="Pesquisar"
 Text="Pesquisar"
 runat="server"
 />

</TD>

</TR>

</Table>

<ASP:DataGrid

 id="MinhaGrade" runat="server"
 Width="700"
 BackColor="#ccccff"
 BorderColor="black"
 ShowFooter="false"
 CellPadding=3
 CellSpacing="0"
 Font-Name="Verdana"
 Font-Size="8pt"
 HeaderStyle-BackColor="#aaaadd"
 MaintainState="false"
 AllowSorting="True"
 OnSortCommand="OrdenaDados"
 />

</form>
</body>
</html>
```

Digite o código da Listagem 11.8 e salve o mesmo em um arquivo chamado chap11ex8.aspx, na pasta chap11, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap10/chap11ex8.aspx>

Será exibida uma página com todos os pedidos. Observe que o título das colunas é um link. Clique no título da coluna CidadeDeDestino. Observe que os dados são classificados, em ordem Crescente, pela coluna CidadeDeDestino. Clique no título da coluna NúmeroDoPedido para classificar a listagem pelo NúmeroDoPedido, conforme indicado na Figura 11.9.

NúmeroDoPedido	NomeDaEmpresa	CidadeDeDestino	PaísDeDestino
10248	Vins et alcools Chevalier	Reims	França
10249	Toms Spezialitäten	Münster	Alemanha
10250	Hanari Carnes	Rio de Janeiro	Brasil
10251	Victuailles en stock	Lyon	França
10252	Suprêmes délices	Charleroi	Bélgica
10253	Hanari Carnes	Rio de Janeiro	Brasil
10254	Chop-suey Chinese	Bern	Suíça
10255	Richter Supermarkt	Genève	Suíça
10256	Wellington Importadora	Resende	Brasil
10257	HILARIOŃ-Abastos	San Cristóbal	Venezuela
10258	Ernst Handel	Graz	Áustria
10259	Centro comercial Moctezuma	México D.F.	México
10260	Ottilie's Käseladen	Köln	Alemanha
10261	Quantiacs.com	Praga	Checoslováquia

Figura 11.9: Todos os pedidos sendo exibidos.

Agora vamos definir um critério de Filtragem. No campo Critério digite a palavra Moreno e dê um clique no botão Pesquisar. Serão exibidos apenas os pedidos para os clientes que contêm a palavra Moreno em qualquer parte do campo NomeDaEmpresa, conforme indicado na Figura 11.10. Para voltar a exibir todos os registros, apague o conteúdo do campo Critério e dê um clique no botão Pesquisar.

Comentários sobre o código do exemplo Chap11ex8.aspx:

- ◆ Novamente utilizamos três procedimentos.
  1. Page\_Load
  2. Busca\_Dados
  3. OrdenaDados

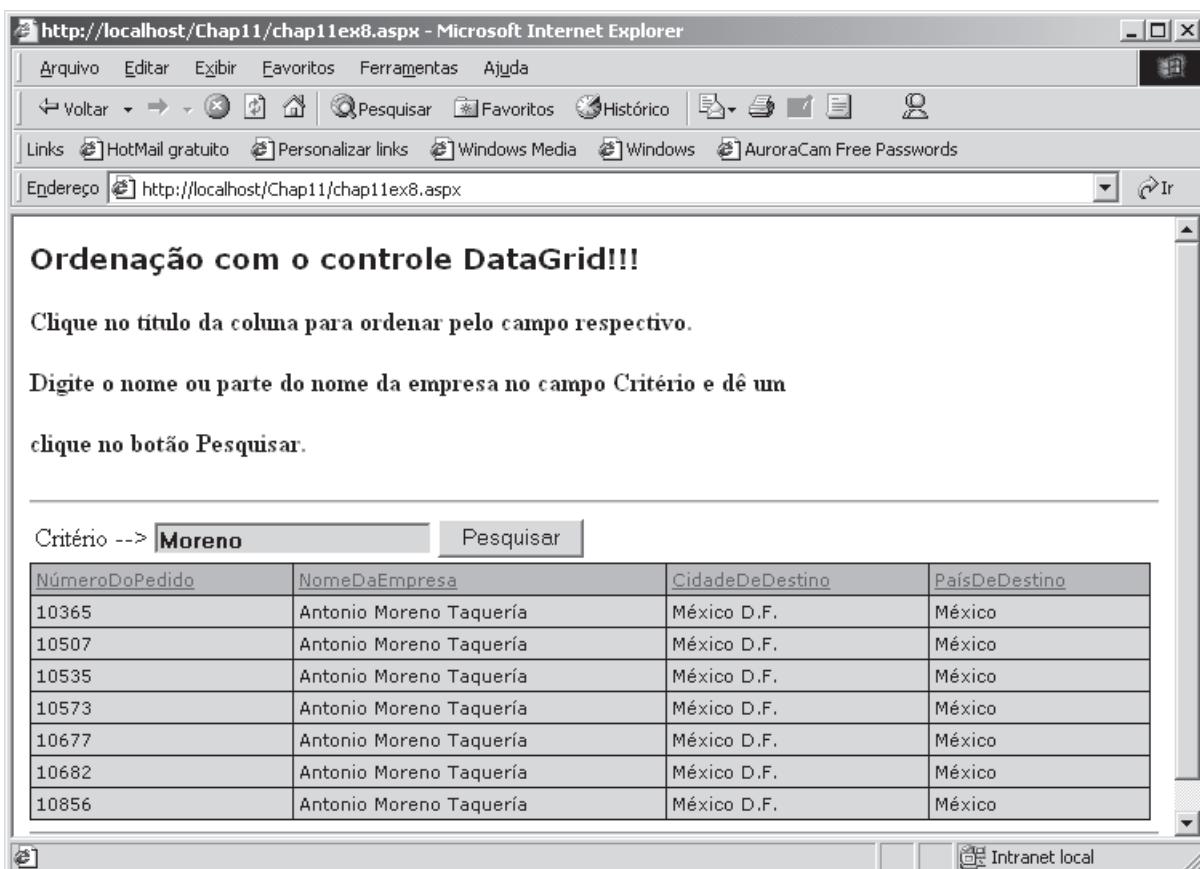


Figura 11.10: Aplicando um critério de filtragem.

Criamos duas variáveis do tipo String; uma para conter o nome do campo pelo qual ordenamos a listagem – OrdenaPor e outra para conter a expressão referente ao critério de Filtragem – FiltraDados:

```
String OrdenaPor;
```

```
String FiltraDados;
```

- ◆ O controle DataGrid, na seção de apresentação, é idêntico ao controle utilizado no exemplo da Listagem 11.7.
- ◆ Ao montar o comando SQL, utilizamos quatro variáveis do tipo String, depois concatenamos as quatro variáveis em uma variável chamada ComandoSQL. Esta variável foi passada como parâmetro para a criação do objeto DataAdapter:

```
String ComandoSQL;

 String aux1SQL;
 String aux2SQL;
 String aux3SQL;
 String aux4SQL;

aux1SQL= "SELECT Pedidos.NúmeroDoPedido, Clientes.NomeDaEmpresa,
Pedidos.CidadeDeDestino";
aux2SQL= ", Pedidos.PaísDeDestino FROM Clientes";
aux3SQL= " INNER JOIN Pedidos ON Clientes.CódigoDoCliente =
Pedidos.CódigoDoCliente";
```

```

aux4SQL= " ORDER BY Pedidos.NúmeroDoPedido";

ComandoSQL = aux1SQL + aux2SQL +aux3SQL +aux4SQL;
OleDbDataAdapter MeuComando = new OleDbDataAdapter(ComandoSQL, MinhaConexão);

```

Observe que fizemos uso da cláusula JOIN, pois estamos retornando dados de duas tabelas: Pedidos e Clientes. O resultado final para a string ComandoSQL é o seguinte:

```

SELECT Pedidos.NúmeroDoPedido, Clientes.NomeDaEmpresa, Pedidos.CidadeDeDestino,
Pedidos.PaísDeDestino
FROM Clientes
INNER JOIN Pedidos
ON
Clientes.CódigoDoCliente = Pedidos.CódigoDoCliente
ORDER BY Pedidos.NúmeroDoPedido;

```

Observe que estamos utilizando a nomenclatura NomeDoCampo.NomeDaTabela. Esta nomenclatura é necessária quando estamos acessando dados de duas ou mais tabelas. Com esta nomenclatura indicamos de qual tabela determinado campo deve ser acessado. Esta nomenclatura é particularmente útil, para situações em que o mesmo nome de campo existe em duas ou mais tabelas. Por exemplo, o campo CódigoDoCliente existe tanto na tabela Pedidos quanto na tabela Clientes.

- ◆ Vamos entender o procedimento Page\_Load: Este evento, conforme já descrito anteriormente, é executado toda vez que a página for carregada. Quando a página for carregada pela primeira vez, a propriedade Page.IsPostBack é igual a False e, neste caso, a variável OrdenaPor é definida como sendo igual a NúmeroDoPedido e o conteúdo do controle DigitaCriterio é definido como “a”.

**NOTA:** Maiores informações sobre a linguagem SQL podem ser encontradas no Anexo III.

Fora da estrutura if/else, é chamado o procedimento BuscaDados( ), ou seja, este procedimento será chamado sempre que a página for carregada, independente de ser um PostBack ou não.

- ◆ Vamos entender o procedimento BuscaDados( ): Grande parte do código deste procedimento já é de nosso conhecimento. São comandos para conectar com o banco de dados NorthWind.mdb, executar um comando SQL e retornar os dados em um objeto DataSet, a partir do qual criamos um objeto DataView. O único detalhe adicional que temos neste procedimento é a definição da propriedade RowFilter, do objeto source, que é um objeto do tipo DataView:

```
source.RowFilter=FiltrarDados;
```

Neste momento o objeto DataView é filtrado de acordo com a expressão de filtragem contida na String FiltraDados.

- ◆ O funcionamento do procedimento OrdenaDados é o mesmo do exemplo da Listagem 11.7. Para maiores detalhes sobre o funcionamento deste procedimento, consulte os comentários para o exemplo da Listagem 11.7.

## O Controle DataGrid em Detalhes

Nos exemplos que utilizamos até o momento, definimos a propriedade DataSource de um controle DataGrid como sendo um objeto DataView e chamamos o método DataBind do DataGrid. Isso faz com que seja gerado um DataGrid com tantas colunas quantos forem os campos do objeto DataView, sendo que as colunas do DataGrid são geradas automaticamente.

Podemos ter controle sobre a maneira como as colunas de um DataGrid são geradas. Por exemplo, podemos definir diferentes aspectos visuais para cada coluna, diferentes tipos, etc. Cada coluna de um DataGrid é um controle de um dos seguintes tipos:

- ◆ BoundColumn
- ◆ ButtonColumn
- ◆ EditCommandColumn
- ◆ HyperLinkColumn
- ◆ TemplateColumn

Por padrão a propriedade AutoGenerateColumns é definida em True, o que faz com que as colunas do DataGrid sejam geradas, automaticamente, e sejam do tipo BoundColumn. Para cada campo da fonte de dados, quer seja um objeto DataView ou um objeto DataReader, será gerada uma coluna do tipo BoundColumn, na ordem em que os campos aparecem na fonte de dados.

A primeira coluna gerada exibirá os dados do campo NúmeroDoPedido; a segunda exibirá os dados do campo CidadeDeDestino e assim por diante.

Para controlar, manualmente, quais colunas farão parte do DataGrid e qual o formato das colunas, devemos definir a propriedade AutoGenerateColumns="False". Uma vez definida esta propriedade como False, podemos adicionar as colunas desejadas. Para controlar a aparência de cada coluna, definimos uma série de propriedades para cada coluna.

Além de definir cada coluna manualmente, também podemos definir as características da primeira linha do DataGrid, normalmente a linha de títulos; dos itens de cada linha; dos itens de cada linha quando em modo de edição e da última linha do DataGrid. Estes elementos são definidos dentro de uma coluna do tipo TemplateColumn.

**NOTA:** A ordem em que os campos aparecem em uma fonte de dados é definida pelo comando SQL. Considere o seguinte comando: Select NúmeroDoPedido, CidadeDeDestino, PaísDeDestino From Pedidos.

A seguir temos a sintaxe completa para o controle DataGrid, quando definimos as colunas do mesmo manualmente:

```
<asp:DataGrid
 id="identificação_no_código"
 runat=server>
```

```
DataSource='<%# Fonte de dados. %>'
AutoGenerateColumns="False"
(outras propriedades)
>

<AlternatingItemStyle property="value"/>
<EditItemStyle property="value"/>
<FooterStyle property="value"/>
<HeaderStyle property="value"/>
<ItemStyle property="value"/>
<PagerStyle property="value"/>
<SelectedItemStyle property="value"/>

<Columns>
 <asp:BoundColumn
 DataField="DataSourceField"
 DataFormatString="FormatString"
 FooterText="FooterText"
 HeaderImageUrl="url"
 HeaderText="HeaderText"
 ReadOnly="True | False"
 SortField="DataSourceFieldToSortBy"
 Visible="True | False"
 FooterStyle-property="value"
 HeaderStyle-property="value"
 ItemStyle-property="value"/>

 <asp:ButtonColumn
 ButtonType="LinkButton | PushButton"
 Command="BubbleText"
 DataTextField="DataSourceField"
 DataTextFormatString="FormatString"
 FooterText="FooterText"
 HeaderImageUrl="url"
 HeaderText="HeaderText"
 ReadOnly="True | False"
 SortField="DataSourceFieldToSortBy"
```

```
 Text="ButtonCaption"
 Visible="True | False"/>

<asp:EditCommandColumn
 ButtonType="LinkButton | PushButton"
 CancelText="CancelButtonCaption"
 EditText="EditButtonCaption"
 FooterText="FooterText"
 HeaderImageUrl="url"
 HeaderText="HeaderText"
 ReadOnly="True | False"
 SortField="DataSourceFieldToSortBy"
 UpdateText="UpdateButtonCaption"
 Visible="True | False"/>

<asp:HyperLinkColumn
 DataNavigateUrlField="DataSourceField"
 DataNavigateUrlFormatString="FormatExpression"
 DataTextField="DataSourceField"
 DataTextFormatString="FormatExpression"
 FooterText="FooterText"
 HeaderImageUrl="url"
 HeaderText="HeaderText"
 NavigateUrl="url"
 ReadOnly="True | False"
 SortField="DataSourceFieldToSortBy"
 Target="window"
 Text="HyperLinkText"
 Visible="True | False"/>

<asp:TemplateColumn
 FooterText="FooterText"
 HeaderImageUrl="url"
 HeaderText="HeaderText"
 ReadOnly="True | False"
 SortField="DataSourceFieldToSortBy"
 Visible="True | False">
```

```

<HeaderTemplate>
 Header template HTML
</HeaderTemplate >

<ItemTemplate>
 ItemTemplate HTML
</ItemTemplate>

<EditItemTemplate>
 EditItem template HTML
</EditItemTemplate>

<FooterTemplate>
 Footer template HTML
</FooterTemplate>

</asp:TemplateColumn>
</Columns>

</asp:DataGrid>

```

Uma vez definida a propriedade AutoGenerateColumns para False, podemos definir manualmente as colunas. Isto é feito entre as tags <Columns> </Columns>. Conforme citado anteriormente, temos diferentes tipos de colunas, tipos estes que estão descritos na Tabela 11.3.

**Tabela 11.3** Tipos de colunas para o controle DataGrid.

Tipo	Descrição
BoundColumn	É uma coluna associada com um campo da fonte de dados (um objeto DataView ou DataReader). Podemos controlar os aspectos visuais utilizando as diversas propriedades da coluna. É o tipo de coluna padrão, isto é, se optarmos pela geração automática de colunas (AutoGenerateColumns="True"), será gerada uma coluna do tipo BoundColumn para cada campo da fonte de dados.
ButtonColumn	Cria uma coluna com botões de comando. Podemos definir o texto dos botões utilizando a propriedade Text ou podemos fazer o DataBind da coluna com uma fonte de dados, de tal maneira que o rótulo do botão de comando seja diferente para cada linha do DataGrid.
HyperLinkColumn	Cria uma coluna com links que são associados com um campo da fonte de dados. Por exemplo, uma coluna que exibe o nome dos funcionários pode ter o nome como um link. Ao clicarmos no Link será aberta uma página com todas as informações do funcionário.

Tipo	Descrição
EditCommandColumn	Cria uma coluna onde podemos definir os comandos para Edição (Update), inclusão (Add) ou exclusão (Remove) de registros.
TemplateColumn	Utilizada para definir o layout dos controles que compõem o DataGrid, a partir de modelos para determinados elementos do HTML. Veremos exemplos mais adiante.

As colunas de um DataGrid, adicionadas manualmente, fazem parte da coleção Columns. A ordem em que as colunas são exibidas no DataGrid, da esquerda para a direita, é definida pela ordem em que as colunas estão definidas na coleção Columns. Podemos alterar esta ordem, utilizando programação.

## Um Exemplo de Criação Manual de Colunas – BoundColumn

Cada tipo de coluna tem uma série de propriedades que podemos utilizar para controlar os aspectos visuais, funcionais e de formatação da coluna. Vamos apresentar um exemplo onde criamos manualmente as colunas de um DataGrid, utilizando colunas do tipo BoundColumn. Utilizaremos algumas propriedades para definir as características de cada coluna, e depois estudaremos mais detalhes sobre as propriedades disponíveis.

Na Listagem 11.9 temos o código para o exemplo proposto.

### Listagem 11.9 – Criando colunas manualmente em um DataGrid.

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

<html>

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

void BuscaDados()
{
 // Crio uma conexão com o banco de dados pubs localizado no servidor local.
 // Vamos acessar a instância SERVIDOR\NETSDK.

 SqlConnection myConnection = new
SqlConnection("server=SERVIDOR\\NETSDK;" +
" uid=sa;pwd=;database=pubs");

 // Conectamos com o banco de dados utilizando um comando SQL,
 // o qual retorna todos os dados da tabela "Authors", do banco de
 // dados pubs, do SQL Server 2000 - SERVIDOR\NETSDK.
}
```

**NOTA:** É possível utilizar uma mescla entre colunas geradas automaticamente e colunas definidas manualmente. Quando isto acontece as colunas criadas manualmente são processadas em primeiro lugar. As colunas geradas automaticamente não farão parte da coleção Columns.

```

 SqlDataAdapter myCommand = new SqlDataAdapter("SELECT " +
" au_id,au_fname,phone FROM Authors Order By au_fname", myConnection);

// Criamos e preenchemos um objeto DataSet.

 DataSet ds = new DataSet();
myCommand.Fill(ds);

// Conectamos um controle DataGrid com o DataSet criado anteriormente.
// MinhaGrade é o id (nome) de um controle do tipo
// DataGrid que está na seção de apresentação da página.

 DataView source = new DataView(ds.Tables[0]);
MinhaGrade.DataSource = source ;
MinhaGrade.DataBind();
}

void Page_Load(Object sender, EventArgs e)
{
 BuscaDados();
}

</script>

<body>

<form runat=server>

<h3>Criando colunas manualmente.</h3>
<HR>
<H4>Observe que temos diferentes formatações para cada coluna>/H4>

Product List

<asp:DataGrid
 id="MinhaGrade"
 BorderColor="black"
 BorderWidth="1"
 CellPadding="3"
 AutoGenerateColumns="false"
 runat="server">

 <HeaderStyle

```

```
 BackColor="#00aaaa"
 Font-Bold="True"
 Font-Name="Courier New"
 ForeColor="White"
 >

</HeaderStyle>

<Columns>

 <asp:BoundColumn
 HeaderText="Código"
 DataField="au_id"
 ItemStyle-BackColor="#c0c0c0"
 >

 </asp:BoundColumn>

 <asp:BoundColumn
 HeaderText="Nome"
 DataField="au_fname"
 ItemStyle-Font-Italic="True"
 ItemStyle-Font-Name="Courier New"
 >

 </asp:BoundColumn>

 <asp:BoundColumn
 HeaderText="Telefone"
 DataField="phone"
 ItemStyle-BackColor="#c0ffff"
 ItemStyle-Font-Bold="True"
 >

 </asp:BoundColumn>
</Columns>

</asp:DataGrid>

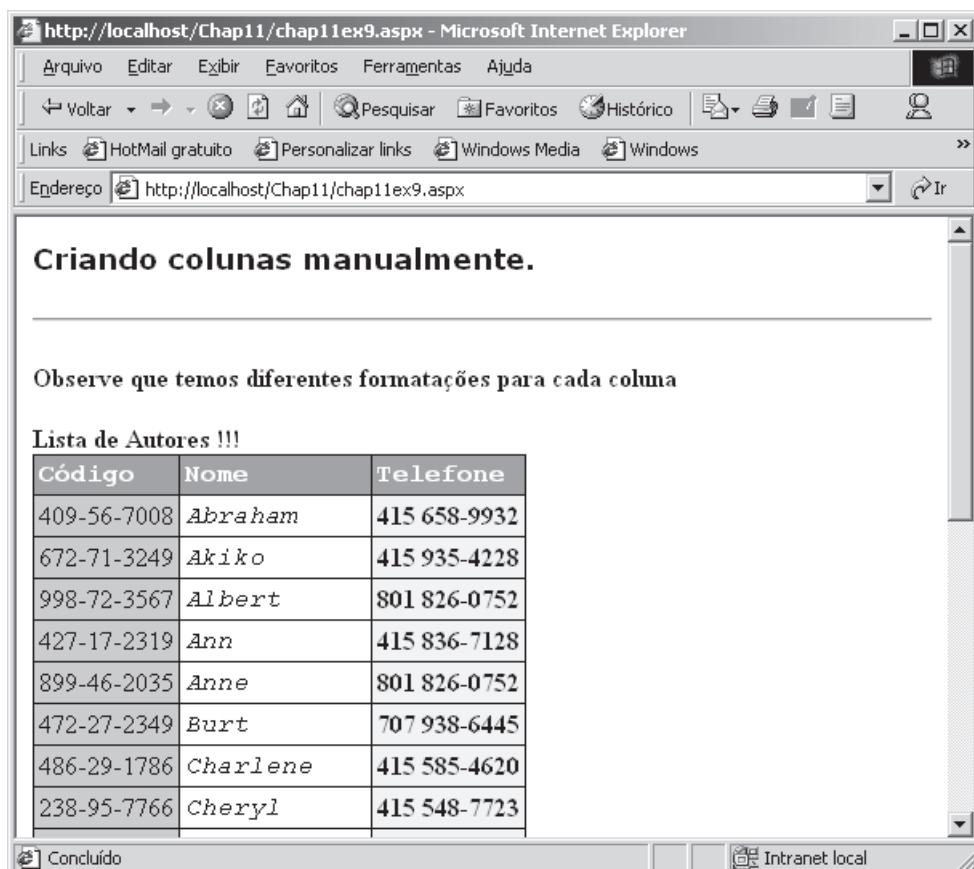
</form>

</body>
</html>
```

Digite o código da Listagem 11.9 e salve o mesmo em um arquivo chamado chap11ex9.aspx, na pasta chap11, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap10/chap11ex9.aspx>

Será exibida uma listagem com o código, nome e telefone dos autores, da tabela authors, do Banco de dados pubs, conforme indicado na Figura 11.11.



**Figura 11.11: Colunas com diferentes formatações.**

Observe que as colunas apresentam diferentes formatações.

Comentários sobre o código do exemplo Chap11ex8.aspx:

- ◆ Em primeiro lugar observe que aplicamos diversas formatações diferentes às diversas partes do controle DataGrid.
1. Para a primeira linha dos títulos das colunas, aplicamos as seguintes formatações: Negrito, fonte Courier New e cor Branca.
  2. Para a primeira coluna aplicamos cor cinza para o segundo plano.
  3. Para a segunda coluna aplicamos uma fonte Courier New e Itálico.
  4. Para a terceira coluna aplicamos negrito e uma tonalidade de verde para a cor de segundo plano.

- ◆ Para criar as colunas manualmente, em primeiro lugar definimos a propriedade AutoGenerateColumns do DataGrid como False:

```
AutoGenerateColumns="False"
```

A definição das colunas é feita dentro das tags <Columns> </Columns>. Neste exemplo utilizamos apenas colunas do tipo BoundColumn.

1. A primeira coluna exibe dados do campo au\_id:

```
<asp:BoundColumn
 HeaderText="Código"
 DataField="au_id"
 ItemStyle-BackColor="#c0c0c0"
 >
</asp:BoundColumn>
```

A propriedade HeaderText define o título da coluna.

A propriedade DataField define com qual campo do objeto DataView a coluna é ligada. Como definimos esta propriedade com au\_id, esta coluna irá exibir o código do autor. Esta propriedade é que faz a ligação da coluna com um campo da fonte de dados.

A propriedade ItemStyle-BackColor define a cor de segundo plano.

2. A segunda coluna exibe dados do campo au\_fname:

```
<asp:BoundColumn
 HeaderText="Nome"
 DataField="au_fname"
 ItemStyle-Font-Italic="True"
 ItemStyle-Font-Name="Courier New"
 >
</asp:BoundColumn>
```

A propriedade HeaderText define o título da coluna.

A propriedade DataField liga a coluna com o campo au\_fname da fonte de dados. Esta coluna exibe o primeiro nome do autor.

A propriedade ItemStyle-Font-Italic define Itálico para a fonte.

A propriedade ItemStyle-Font-Name define o tipo de fonte a ser utilizado. Poderíamos ter utilizado Arial, Times New Roman, etc.

3. A terceira coluna exibe dados do campo phone:

```
<asp:BoundColumn
 HeaderText="Telefone"
 DataField="phone"
 ItemStyle-BackColor="#c0ffff"
 ItemStyle-Font-Bold="True"
>
</asp:BoundColumn>
```

A propriedade HeaderText define o título da coluna.

A propriedade DataField liga a coluna com o campo phone da fonte de dados.

A propriedade ItemStyle-Font-Bold define Negrito para a fonte.

- ◆ Também incluímos um elemento HeaderStyle. Este elemento define a formatação para a linha de títulos das colunas. Definimos negrito (Font-Bold="True"), fonte CourierNew (Font-Name="Courier New") e cor de fonte branca (ForeColor="White").

Observe que as formatações para o elemento HeaderStyle são diferentes das formatações para o elemento BoundColumn. Por exemplo, no elemento BoundColumn, utilizamos ItemStyle-Font-PropriedadeDaFonte, para definir uma característica da fonte. Por exemplo, utilizamos ItemStyle-Font-Bold="True", para definir Negrito. Já no elemento HeaderStyle utilizamos apenas Font-PropriedadeDaFonte. Por exemplo, utilizamos Font-Bold="True", para definir Negrito.

```
<HeaderStyle
 BackColor="#00aaaa"
 Font-Bold="True"
 Font-Name="Courier New"
 ForeColor="White"
>
</HeaderStyle>
```

- ◆ Na seção de código da página, simplesmente estabelecemos uma conexão com o servidor SQL Server e utilizamos um objeto Data Repeater para executar um comando SQL que preenche um objeto DataSet. A partir do objeto DataSet criamos um objeto DataView, objeto este que é associado com o DataGrid. Criamos um procedimento BuscaDados( ), o qual é chamado toda vez que a página é carregada – Page\_Load.

**NOTA:** Ainda não estudamos todos os detalhes do controle DataGrid. No próximo tópico, veremos como implementar a paginação, utilizando um controle DataGrid. No Capítulo 12 aprenderemos a utilizar o controle DataGrid para Editar, Alterar e Excluir dados.

## Implementando Paginação com o Controle DataGrid

No Capítulo 10 descrevemos o conceito de paginação e vimos que, para habilitar a paginação, devemos definir a propriedade AllowPaging como True. Neste tópico veremos

como fazer com que a paginação funcione, ou seja, quando o usuário clica no link Próxima Página ou no número da próxima página, faremos com que seja exibido o próximo conjunto de registros.

Ao tornarmos a propriedade AllowPaging igual a True, será criada mais uma linha, no final do DataGrid. Nesta linha podemos exibir um link para a próxima página e para a página anterior ou a numeração de página: 1 2 3 4 5 6 7 8 9 10 ...; onde cada número é um link para a página respectiva. Podemos personalizar o que é exibido nesta linha que chamaremos de Barra de Navegação. Por padrão os links de navegação são alinhados à esquerda, no DataGrid.

Cada vez que o usuário clica em um dos links da Barra de Navegação, é disparado o eventoPageIndexChanged. Ao criarmos o DataGrid definimos a propriedadePageIndexChanged com o nome do procedimento que será executado em resposta ao evento. Neste procedimento colocaremos o código para navegar para a página relacionada ao link clicado pelo usuário.

Em resumo, para fazer paginação com o DataGrid, devemos fazer o seguinte:

- ◆ Definir a propriedade AllowPaging como True.
- ◆ Definir o nome do procedimento que será executado em resposta ao eventoPageIndexChanged.
- ◆ Criar o código do procedimento que executa em resposta ao evento OnIndexChanged.

Para definir as características da linha onde são exibidos os links de navegação, nós fazemos uso das propriedades PagerStyle-Definição, do controle DataGrid. Por exemplo, para alinhar os links de navegação à direita, nós definimos a seguinte propriedade:

**PagerStyle-HorizontalAlignment="Right"**

Na Tabela 11.4 temos a definição das principais características da Barra de Navegação, que podem ser configuradas com a propriedade PagerStyle.

**Tabela 11.4** Configurações da propriedade PagerStyle.

Propriedade	Utilizada para definir
PagerStyle-BackColor	A cor de segundo plano.
PagerStyle-BorderColor	A cor das bordas.
PagerStyle-BorderWidth	O tamanho das bordas.
PagerStyle-Font	Características da fonte (Negrito, Itálico, etc.).
PagerStyle-ForeColor	A cor da fonte.
PagerStyle-Mode	Define se serão exibidos links “Próximo” e “Anterior”, ou se um link para o número de cada página. O valor padrão é NextPrev, o que faz com que sejam exibidos os botões “Próximo” e “Anterior”. Se definirmos o valor desta propriedade como NumericPages, serão exibidos links para o número de cada página.
PagerStyle-NextPageText	O texto para o link que navega para a próxima página.

Propriedade	Utilizada para definir
PagerStyle-PrevPageText	O texto para o link que navega para a página anterior.
PagerStyle-Position	Define a posição da Barra de Navegação. Pode ser Bottom – barra no final do DataGrid, que é o valor padrão, Top – barra no início do DataGrid ou TopAndBottom – barra no início e no final do DataGrid.

Para definir o número de linhas por página, utilizamos a propriedade PageSize. Por exemplo, o seguinte comando define que sejam exibidos quinze registros por página:

```
PageSize="15"
```

Amos apresentar um exemplo.

Exemplo: Vamos criar um exemplo, onde retornamos o nome de todos os clientes da tabela Clientes, do Banco de dados Northwind.mdb. Dividiremos os registros em páginas de dez registros. Utilizaremos o modo padrão, onde é exibido um link “Próxima Página ->>” (com exceção da última página, quando este link estará desabilitado) e um link “<<- Página Anterior” (com exceção da primeira página, quando este link estará desabilitado). Utilizaremos algumas propriedades da Tabela 11.4 para personalizar a aparência da Barra de Navegação.

Na Listagem 11.10 temos o código para o exemplo proposto.

#### Listagem 11.10 – Paginação com o controle DataGrid.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>

<html>

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

void Page_Load(Object Src, EventArgs E)
{
 if (!Page.IsPostBack)
 {
 MinhaGrade.CurrentPageIndex = 0;
 BuscaDados();
 }
}

void BuscaDados()
{
```

```
// Crio uma conexão com o banco de dados do Microsoft Access.
// C:\Meus documentos\NorthWind.mdb.

String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
 "DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";

OleDbConnection MinhaConexão = new OleDbConnection(DefineConexão);

// Utilizamos um objeto DataAdapter para executar um comando SQL,
// o qual retorna as cidades da Alemanha.

String aux1SQL;
String aux2SQL;
String ComandoSQL;

aux1SQL= "SELECT CódigoDoCliente, NomeDaEmpresa, Cidade, País, Telefone ";
aux2SQL= "FROM Clientes Order By CódigoDoCliente";

ComandoSQL= aux1SQL + aux2SQL;

OleDbDataAdapter MeuComando = new OleDbDataAdapter(ComandoSQL, MinhaConexão);

// Criamos e preenchemos um objeto DataSet.

DataSet ds = new DataSet();

// Utilizo o método Fill do objeto DataAdapter, para preencher
// o objeto DataSet, com os dados retornados pelo comando SQL.

MeuComando.Fill(ds);

// Criamos um objeto DataView ligado com a primeira
// tabela, da coleção de tabelas, do objeto ds.

DataView source = new DataView(ds.Tables[0]);

// Ligo o objeto DataView a um controle do tipo
```

```
// DataGrid - MinhaGrade.

MinhaGrade.DataSource = source;
MinhaGrade.DataBind();

}

// Procedimento que faz a navegação para a próxima página
// ou para a página anterior, dependendo do link clicado
// pelo usuário.

void TrocaPagina(Object sender, DataGridPageChangedEventArgs e)
{
 MinhaGrade.CurrentPageIndex=e.NewPageIndex;
 BuscaDados();
}

</script>

<body>

<h3>Paginação com ASP.NET!!!</h3>

<form runat=server>

<HR>

<ASP:DataGrid
 id="MinhaGrade"
 runat="server"
 CellPadding=3
 CellSpacing="0"
 Font-Name="CourierNew"
 Font-Size="10pt"
 HeaderStyle-BackColor="#aaaadd"
 MaintainState="false"
 AllowPaging="True"
 OnPageIndexChanged="TrocaPagina"
 PageSize="10"
 PagerStyle-Position="TopAndBottom"
 PagerStyle-NextPageText="Próxima Página ->>">
```

```

 PagerStyle-PrevPageText="<<- Página Anterior"
 PagerStyle-Font-Bold="True"
 PagerStyle-Font-Size="12pt"
 PagerStyle-BackColor="#c0c0c0"

 />

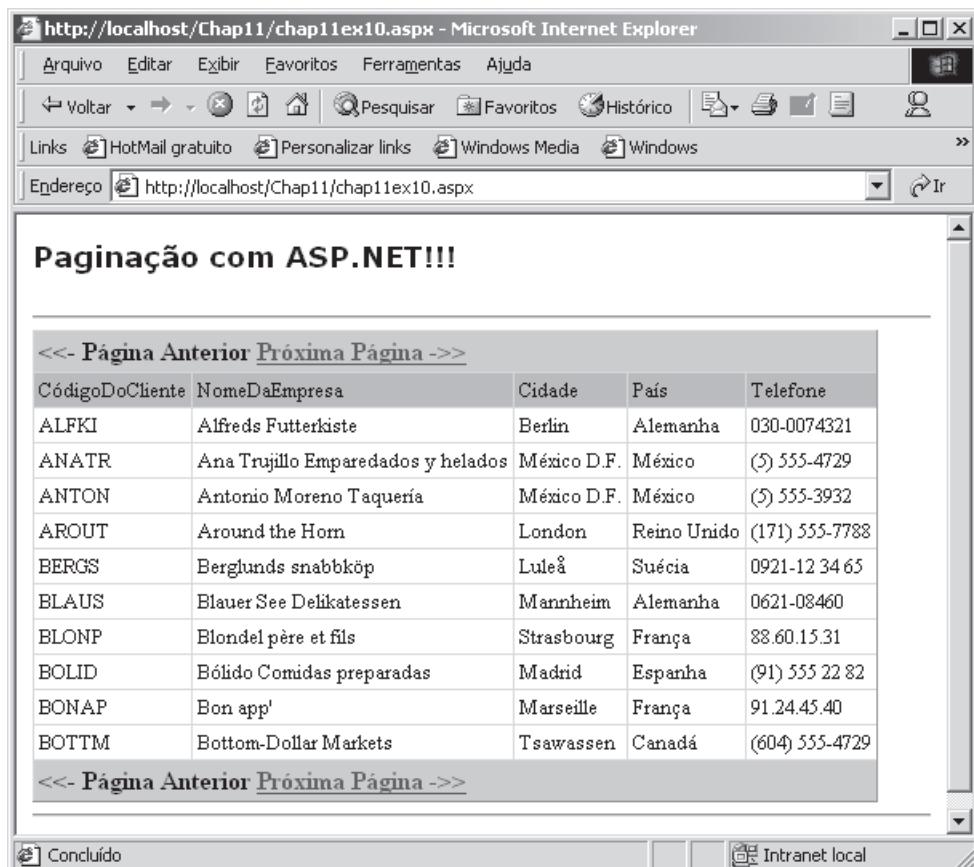
<HR>
</form>
</body>
</html>

```

Digite o código da Listagem 11.10 e salve o mesmo em um arquivo chamado chap11ex10.aspx, na pasta chap11, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap10/chap11ex10.aspx>

Será exibida uma listagem com os 10 primeiros clientes. Observe que temos links para a próxima página e para a página anterior, no início e no final do DataGrid, conforme indicado na Figura 11.12.



The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost/Chap11/chap11ex10.aspx - Microsoft Internet Explorer". The menu bar includes Arquivo, Editar, Exibir, Favoritos, Ferramentas, and Ajuda. The toolbar includes links for Voltar, Avançar, Pesquisar, Favoritos, Histórico, and others. The address bar shows the URL "http://localhost/Chap11/chap11ex10.aspx". The main content area displays a DataGrid with the following header row:

CódigoDoCliente	NomeDaEmpresa	Cidade	País	Telefone
-----------------	---------------	--------	------	----------

The DataGrid contains 10 rows of client information:

ALFKI	Alfreds Futterkiste	Berlin	Alemanha	030-0074321
ANATR	Ana Trujillo Emparedados y helados	México D.F.	México	(5) 555-4729
ANTON	Antonio Moreno Taquería	México D.F.	México	(5) 555-3932
AROUT	Around the Horn	London	Reino Unido	(171) 555-7788
BERGS	Berglunds snabbköp	Luleå	Suécia	0921-12 34 65
BLAUS	Blauer See Delikatessen	Mannheim	Alemanha	0621-08460
BLONP	Blondel père et fils	Strasbourg	França	88.60.15.31
BOLID	Bólido Comidas preparadas	Madrid	Espanha	(91) 555 22 82
BONAP	Bon app'	Marseille	França	91.24.45.40
BOTTM	Bottom-Dollar Markets	Tsawassen	Canadá	(604) 555-4729

Below the DataGrid, there are two sets of navigation links:

- Top:** "<<- Página Anterior" and "Próxima Página ->>"
- Bottom:** "<<- Página Anterior" and "Próxima Página ->>"

The status bar at the bottom of the browser window shows "Concluído" and "Intranet local".

Figura 11.12: Listagem de Clientes dividida em páginas.

Clique no link “Próxima Página ->” e observe que são exibidos os próximos dez clientes. Também observe que o link “<<- Página Anterior” foi habilitado, conforme indicado na Figura 11.13.

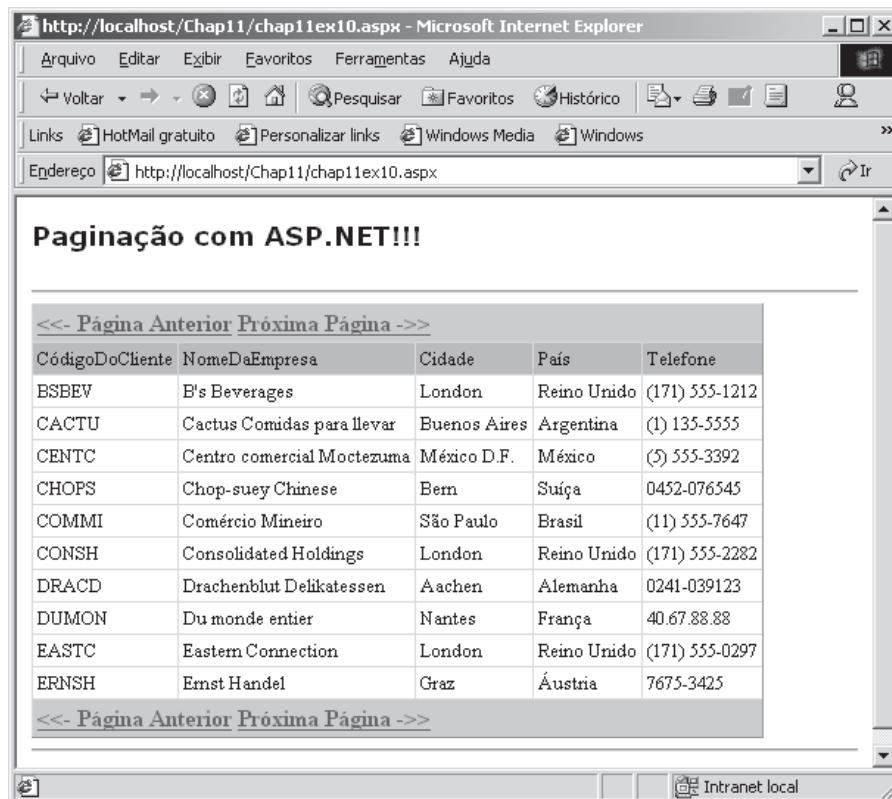


Figura 11.13: Navegando com o controle DataGrid.

Comentários sobre o código do exemplo Chap11ex8.aspx:

- ♦ Na definição do DataGrid utilizamos uma série de propriedades para a definição dos aspectos visuais da barra de navegação. Estas propriedades estão descritas na Tabela 11.4. Uma propriedade interessante é a propriedade Mode, que por padrão tem o valor NextPrev. Se alterarmos o seu valor para NumericPages, ao invés do link Próximo e do link Anterior, teremos diversos links, um para cada página do DataGrid. Vamos alterar a definição do DataGrid, para que a mesma fique da seguinte maneira:

```
<ASP:DataGrid
 id="MinhaGrade"
 runat="server"
 CellPadding=3
 CellSpacing="0"
 Font-Name="CourierNew"
 Font-Size="10pt"
 HeaderStyle-BackColor="#aaaadd"
 MaintainState="false"
```

```

AllowPaging="True"
OnPageIndexChanged="TrocaPagina"
PageSize="10"
PagerStyle-Position="TopAndBottom"
PagerStyle-Mode="NumericPages"
PagerStyle-Font-Bold="True"
PagerStyle-Font-Size="12pt"
PagerStyle-BackColor="#c0c0c0"
/>

```

Se fizermos estas alterações e recarregarmos a página, obteremos o resultado indicado na Figura 11.14.

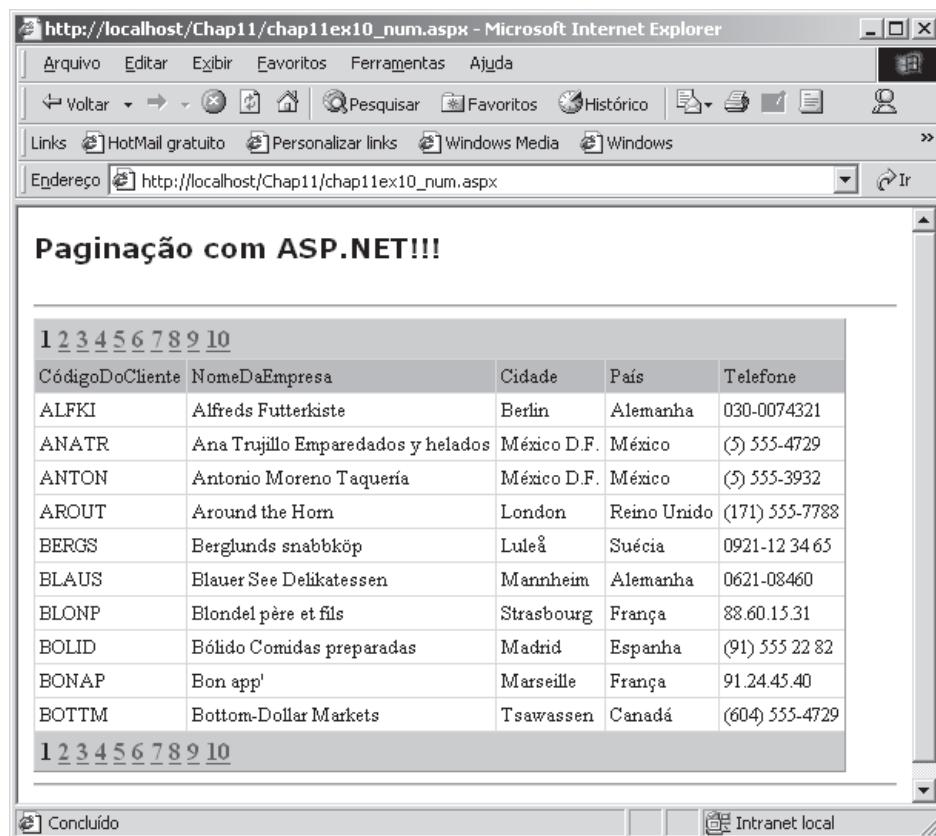


Figura 11.14: **PagerStyle-Mode="NumericPages"**.

- ◆ No evento Page\_Load detectamos se não é um PostBack (isto acontece quando a página está sendo carregada pela primeira vez); caso não seja um PostBack, definimos a página atual como sendo a primeira página e executamos o procedimento BuscaDados:

```
MinhaGrade.CurrentPageIndex = 0;
```

```
BuscaDados();
```

A definição da página atual é feita com o uso da propriedade CurrentPageIndex, do DataGrid. A primeira página é a página zero, a segunda é a página um e assim por diante.

- ◆ O procedimento BuscaDados é idêntico ao utilizado em exemplos anteriores e, portanto, dispensa maiores comentários.
- ◆ O procedimento TrocaPagina é definido na propriedade OnPageIndexChanged, do DataGrid:  
**OnPageIndexChanged="TrocaPagina"**

O evento OnPageIndexChanged é disparado toda vez que o usuário clica em um dos links de navegação. O procedimento TrocaPagina recebe um argumento do tipo DataGridPageChangedEventArgs, do namespace System.Web.UI.WebControls. Quando clicamos em um dos links de navegação, o índice da página para a qual aponta o link pode ser acessado através da NewPageIndex deste argumento. Atribuímos o valor contido nesta propriedade para a propriedade CurrentPageIndex do DataGrid. O efeito prático é carregar a página do DataGrid, cujo índice é o mesmo definido na propriedade NewPageIndex.

- ◆ Observe que, simplesmente, com a definição de uma propriedade e um procedimento de duas linhas, implementamos paginação no ASP.NET, o que com ASP 3.0 demandaria uma boa quantidade de código.

## Conclusão

Iniciamos o capítulo apresentando o conceito de DataBinding e aprendendo a utilizar os seguintes Web Server Controls:

- ◆ CheckBoxList
- ◆ DropDownList
- ◆ RadioButtonList

Em seguida começamos a estudar mais alguns detalhes sobre o controle DataGrid. Aprendemos as seguintes operações:

- ◆ Filtrar dados.
- ◆ Criar as colunas do DataGrid manualmente.
- ◆ Fazer paginação.

No Capítulo 12 continuaremos trabalhando com o acesso a Banco de dados. Aprenderemos a criar páginas ASP.NET que permitem a edição, exclusão e inclusão de registros no Banco de dados. Aprenderemos a utilizar mais algumas classes dos namespaces System.Data, System.Data.SqlClient e System.Data.OleDb.

Como não poderia deixar de ser, aprenderemos a realizar mais algumas operações e a utilizar mais algumas opções do controle DataGrid.

# Introdução

Nos capítulos 10 e 11 aprendemos a acessar dados do Microsoft Access e do SQL Server 2000. Utilizamos diversas classes do ADO.NET para fazer o acesso aos dados. Comentamos que com ASP.NET temos um “modelo desconectado”, ou seja, retornamos uma ou mais tabelas através de um objeto DataSet e encerramos a conexão com o banco de dados.

Nos exemplos vistos até o momento apenas exibimos, pesquisamos e classificamos os dados obtidos. Neste capítulo aprenderemos a alterar, adicionar e excluir dados. Estas operações serão feitas no conjunto de dados do objeto DataSet, ou seja, no conjunto de dados Desconectados. Em seguida aprenderemos a passar as alterações que foram feitas no conjunto de dados desconectados, para a fonte original de dados; em outras palavras, aprenderemos a sincronizar as alterações feitas nos dados do DataSet com a fonte original de dados.

Iniciaremos o capítulo aprendendo a utilizar o objeto DataTable para realizar as seguintes operações básicas:

- ◆ Adicionar um registro.
- ◆ Excluir um registro.
- ◆ Alterar um registro.

Para realizar estas operações aprenderemos novas propriedades e métodos do objeto DataTable. Em seguida aprenderemos a enviar as alterações feitas no conjunto de dados do objeto DataTable, de volta para o banco de dados.

Neste capítulo apresentamos tópicos fundamentais para a construção de qualquer aplicação Web que acesse dados. Por exemplo, se você constrói um site, onde o internauta pode cadastrar-se para participar de promoções, você precisará construir um formulário onde o internauta possa digitar seus dados e enviar os mesmos para o banco de dados. Você também terá que fornecer um formulário para que o usuário possa alterar seus dados e, finalmente, uma opção para que o usuário possa excluir seu cadastro.

Conforme destacaremos neste capítulo, as operações de manipulação de dados com o ASP.NET são bem diferentes das operações que utilizávamos com ASP 3.0. De início, a maior dificuldade será entender este novo mecanismo.

Vamos apresentar exemplos detalhados, onde explicaremos, em detalhes, toda a funcionalidade de cada exemplo. Desta maneira, o leitor poderá entender os passos necessários e quais objetos utilizar para implementar as operações de manipulação de dados.

# CAPÍTULO 12

## Acessando Bancos de Dados com ASP.NET – Parte 3

Nos diversos exemplos deste capítulo, utilizaremos os objetos do namespace System.Data.OleDb, para fazer conexão com um banco de dados do Microsoft Access – C:\Meus documentos\ NorthWind.mdb. Você pode, facilmente, adaptar os exemplos para utilizar um banco de dados do SQL Server ou do ORACLE.

Para utilizar um banco de dados do SQL Server, basta alterar a string de conexão, de tal maneira que, ao invés de utilizar o banco de dados NorthWind.mdb, você passe a utilizar o banco de dados NorthWind que é instalado com o SQL Server 2000. Porém, para obter um melhor desempenho, sugiro que você utilize, ao invés dos objetos do namespace System.Data.OleDb, os objetos do namespace System.Data.SqlClient, os quais são específicos e otimizados para o acesso a bancos de dados do SQL Server.

Para conexão com o ORACLE você apenas precisa alterar a string de conexão, para fazer a conexão com um servidor/banco de dados onde o ORACLE esteja rodando.

## Não Esqueça, Estamos em um Modelo Desconectado

Vamos recordar os passos que utilizamos nos Capítulos 10 e 11, para conectar e retornar dados a partir de uma fonte de dados:

1. Criamos um objeto OleDbConnection ou SqlConnection.
2. Criamos um objeto OleDbDataAdapter ou SqlDataAdapter. Passamos como parâmetros para este objeto uma string SQL e uma referência para o objeto Connection, criado no item 1.
3. Criamos um objeto do tipo DataSet.
4. Utilizamos o método Fill, do objeto DataAdapter, para executar o comando SQL passado como parâmetro, na criação do objeto DataAdapter e retornar os dados para o objeto DataSet.
5. Neste momento temos um objeto DataSet com dados. Podemos criar um objeto DataView para ligar estes dados (Data Bound) a um controle, como por exemplo DataGrid ou DropDownList.

É importante salientar que o objeto DataSet não mantém nenhuma conexão com o banco de dados, ou seja, o mesmo está desconectado do banco de dados. Para fazermos alterações nos dados, e enviar estas alterações de volta para o banco de dados, utilizaremos a seguinte abordagem:

- ◆ Criaremos um objeto do tipo DataTable, ligado a uma das tabelas do DataSet.
- ◆ Faremos as alterações, inclusões e exclusões necessárias, utilizando os métodos respectivos do objeto DataTable.
- ◆ Utilizaremos métodos do objeto DataAdapter para enviar as alterações de volta para o banco de dados. Se não utilizarmos estes métodos, as alterações não serão enviadas para o banco de dados e serão perdidas quando a página for encerrada.

## O Objeto DataTable – Alterações nos Dados Desconectados

O objeto DataTable faz parte do namespace System.Data. Este objeto é utilizado para representar uma tabela de dados na memória.

Podemos criar um objeto DataTable completamente independente, isto é, podemos definir através de código as colunas do objeto e adicionar registros. Outra maneira é criar o objeto associado a uma tabela de um objeto DataSet.

Uma das maneiras de criar um objeto DataSet é utilizando o próprio construtor da classe DataTable, conforme o exemplo a seguir:

```
DataTable MinhaTabela = new DataTable("Pedidos");
```

No exemplo a seguir, criamos um objeto DataTable, utilizando o método Add, da coleção Tables, de um objeto DataSet:

```
DataSet MeuDataSet = new DataSet();
DataTable MinhaTabela = MeuDataSet.Tables.Add("Pedidos");
```

O número máximo de linhas que um objeto DataTable pode conter é: 16.777.216. O objeto DataTable também possui uma coleção de objetos do tipo Constraint. Podemos utilizar esta coleção para implementar a “Integridade dos dados”. Por exemplo, podemos utilizar uma Constraint para definir que a coluna Salário não pode conter valores maiores do que R\$ 10.000,00 e que a coluna DiasDeFérias deve conter um valor entre 1 e 30.

**IMPORTANTE:** No momento da criação do objeto DataTable, o construtor da classe DataTable faz distinção entre maiúsculas e minúsculas, para o nome da tabela. Por exemplo, os dois comandos a seguir criam dois objetos DataTable distintos:

```
DataTable MinhaTabela = new DataTable("Pedidos");
DataTable MinhaOutraTabela = new DataTable("pedidos");
```

Na Tabela 12.1 temos a descrição das principais propriedades da classe DataTable.

**Tabela 12.1** Principais propriedades da classe DataTable.

Propriedade	Descrição
CaseSensitive	Propriedade booleana que indica se, para comparação de Strings, será feita distinção entre maiúsculas e minúsculas.
Columns	Retorna a coleção de colunas da tabela.
Constraints	Retorna a coleção de Constraints da tabela.
DataSet	Retorna o objeto DataSet a partir do qual foi criado o objeto DataTable.
PrimaryKey	Utilizada para retornar ou definir um Array de colunas que atuam como Chave Primária da tabela.
Rows	Retorna a coleção de linhas da tabela.
TableName	Utilizada para retornar ou definir o nome associado ao objeto DataTable.

Na Tabela 12.2 temos a descrição dos principais métodos da classe DataTable.

**Tabela 12.2** Principais métodos da classe DataTable.

Método	Descrição
AcceptChanges	Torna definitivas todas as alterações que foram feitas, nos dados do DataTable, desde a última chamada do método AcceptChanges. Quando este método é chamado, qualquer objeto DataRow (que representa uma linha da tabela), que ainda estiver em modo de edição, terá sua edição encerrada. Todas as linhas adicionadas ou modificadas, desde a última chamada do método, terão o seu estado alterado para Unchanged. As linhas marcadas como Deletadas serão removidas. É importante que você somente chame o método AcceptChanges, após ter atualizado a fonte de dados, utilizando um dos métodos que aprenderemos nos próximos tópicos.
Clear	Remove todas as linhas do DataTable.
Clone	Faz uma cópia da estrutura do DataTable, inclusive os relacionamentos e constraints.
Copy	Faz uma cópia da estrutura e também dos dados do DataTable.
LoadDataRow	Localiza e atualiza uma linha especificada. Se não for encontrada nenhuma linha coincidente, uma nova linha será criada, com os valores passados para o método. Passamos um Array de valores, como parâmetro para este método.
NewRow	Este método é utilizado para criar uma nova linha.
RejectChanges	Desfaz todas as alterações que foram feitas desde que a tabela foi inicializada ou desde a última chamada do método AcceptChanges.

## Inserindo, Excluindo e Adicionando Dados com o Objeto DataTable

Vamos apresentar alguns exemplos onde faremos as seguintes operações:

- ◆ Adição de uma nova linha.
- ◆ Exclusão de uma linha.
- ◆ Alteração de uma linha.

Neste tópico aprenderemos a fazer as alterações no conjunto de dados da tabela representada pelo objeto DataTable. Porém ainda não faremos a sincronização de dados com o banco de dados. Iremos comprovar este fato exibindo os dados do DataTable, após as alterações e conferindo que as mesmas não foram enviadas para o banco de dados. Isso comprova o modelo desconectado do ASP.NET. Nos próximos tópicos aprenderemos a fazer a sincronização com o banco de dados.

Exemplo 1: Vamos criar um exemplo onde acessamos os dados da tabela Clientes, do banco de dados NorthWind.mdb. Retornaremos apenas os clientes para o Brasil. Utilizando código, no evento Page\_Load, adicionaremos dois novos registros ao objeto DataTable. Exibiremos a tabela original em um DataGrid e a tabela com os novos registros em um

segundo DataGrid. Em seguida abriremos o banco de dados NorthWind.mdb e comprovaremos que as adições não foram enviadas para o banco de dados.

Na Listagem 12.1 temos o código para o exemplo proposto.

Listagem 12.1 – O objeto DataTable – Ch1ap12Ex1.aspx.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>

<html>

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

protected void Page_Load(Object Src, EventArgs E)
{
 // Crio uma conexão com o banco de dados NorthWind.mdb
 // que está na pasta C:\Meus documentos.

 String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
 "DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";

 OleDbConnection MinhaConexão = new OleDbConnection(DefineConexão);

 // Utilizamos um objeto DataAdapter para executar um comando SQL,
 // o qual retorna todos os dados da tabela "Clientes".

 String auxSQL1;
 String auxSQL2;
 String auxSQL3;
 String ComandoSQL;

 auxSQL1= "SELECT CódigoDoCliente,NomeDaEmpresa,Endereço,Telefone,Páis";
 auxSQL2= " FROM Clientes Where País='Brasil'";
 auxSQL3= " Order By NomeDaEmpresa";

 ComandoSQL= auxSQL1+auxSQL2+auxSQL3;
}
```

```
OleDbDataAdapter MeuComando = new OleDbDataAdapter(ComandoSQL, MinhaConexão);

// Criamos e preenchemos um objeto DataSet.

DataSet ds = new DataSet();

// Utilizo o método Fill do objeto DataAdapter, para preencher
// o objeto DataSet, com os dados retornados pelo comando SQL.

MeuComando.Fill(ds,"Clientes2");

// Conectamos um controle DataGridView com o DataSet criado anteriormente.
// MinhaGrade é o id (nome) de um controle do tipo
// DataGridView que está na seção de apresentação da página.

DataView source = new DataView(ds.Tables[0]);

MinhaGrade1.DataSource = source ;
MinhaGrade1.DataBind();

// Agora vamos criar um objeto DataTable.
// Chamaremos este objeto de Clientes2 e iremos
// associá-lo a primeira tabela do objeto ds.

DataTable Clientes2 = ds.Tables[0];

// Adicionaremos duas novas linhas à tabela Clientes2.

DataRow Linha = Clientes2.NewRow();

// Defino os valores para a linha a ser inserida.

Linha["CódigoDoCliente"]="XXYYK";
Linha["NomeDaEmpresa"]="ABC Ltda.";
Linha["Endereço"]="Rua das letras, 123";
Linha["Telefone"]="222-2222";
Linha["País"]="Brasil";
```

```
 Clientes2.Rows.Add(Linha);

 DataRow Linha2 = Clientes2.NewRow();
<1
// Defino os valores para a linha a ser inserida.

 Linha2["CódigoDoCliente"] = "WWMMK";
 Linha2["NomeDaEmpresa"] = "LMN Ltda.";
 Linha2["Endereço"] = "Rua dos números, 123";
 Linha2["Telefone"] = "555-5555";
 Linha2["País"] = "Brasil";

 Clientes2.Rows.Add(Linha2);

 MinhaGrade2.DataSource = Clientes2.DefaultView;
 MinhaGrade2.DataBind();

}

</script>

<body>
<%-- Exibe as informações do DataGrid no corpo da página. --%>

<h3>Exemplo do objeto DataSet!!!</h3>

<ASP:DataGrid
 id="MinhaGrade1"
 runat="server"
 Width="700"
 BackColor="#ccccff"
 BorderColor="black"
 ShowFooter="false"
 CellPadding=3
 CellSpacing="0"
 Font-Name="Verdana"
 Font-Size="8pt">
```

```

HeaderStyle-BackColor="#aaaaadd"
MaintainState="false"
/>>

<HR>
<ASP:DataGrid
 id="MinhaGrade2"
 runat="server"
 width="700"
 BackColor="#ccccff"
 BorderColor="black"
 ShowFooter="false"
 CellPadding=3
 CellSpacing="0"
 Font-Name="Verdana"
 Font-Size="8pt"
 HeaderStyle-BackColor="#aaaaadd"
 MaintainState="false"
/>>
</body>
</html>

```

Digite o código da Listagem 12.1 e salve o mesmo em um arquivo chamado chap12ex1.aspx, na pasta chap12, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap12/chap12ex1.aspx>

Ao carregar a página você obtém o resultado indicado na Figura 12.1.

Comentários sobre o código do exemplo – Chap12ex1.aspx.

- ◆ Na seção de apresentação da página, utilizamos dois controles DataGrid – MinhaGrade1 e MinhaGrade2. No controle MinhaGrade1 exibimos os registros retornados da tabela Clientes; no controle MinhaGrade2 exibimos os dados, após termos adicionados duas linhas ao objeto DataTable.
- ◆ Utilizamos o evento Page\_Load da página, para fazer a conexão com o banco de dados. Neste evento temos a seguinte seqüência de passos:
  1. Conectamos com o Banco de dados NortWhind.mdb e retornamos somente os Clientes do Brasil. Em seguida utilizamos um objeto DataView para ligar os dados retornados com o controle DataGrid1. Esta parte do código já foi utilizada em outros exemplos, dos capítulos anteriores.



Figura 12.1: Utilizando o objeto DataTable – Chap12ex1.aspx.

- Em seguida criamos um objeto DataTable, chamado Clientes2, objeto este ligado à primeira tabela do objeto DataSet, que no nosso caso é a tabela Clientes. Na prática estamos criando uma cópia da tabela Clientes, na tabela Clientes2.

```
DataTable Clientes2 = ds.Tables[0];
```

- Na seqüência declaro um objeto do tipo DataRow – Linha, defino o valor de cada coluna e utilizo o método Add, da coleção Rows da tabela Clientes2, para adicionar a linha à coleção de linhas da tabela.

```
// Declaro um objeto do tipo DataRow.
```

```
DataRow Linha = Clientes2.NewRow();
```

```
// Defino os valores para as colunas da linha a ser inserida.
```

```
Linha["CódigoDoCliente"] = "XXYYK";
```

```
Linha["NomeDaEmpresa"] = "ABC Ltda.,";
```

```
Linha["Endereço"] = "Rua das letras, 123";
```

```

Linha["Telefone"] = "222-2222";
Linha["País"] = "Brasil";

// Adiciona a linha à coleção de linhas da tabela Clientes2.

```

4. Repetimos as operações do item anterior, para adicionar mais uma linha:

```
// Declaro um objeto do tipo DataRow.
```

```
DataRow Linha2 = Clientes2.NewRow();
```

```
// Defino os valores para a linha a ser inserida.
```

```

Linha2["CódigoDoCliente"] = "WWMMK";
Linha2["NomeDaEmpresa"] = "LMN Ltda.";
Linha2["Endereço"] = "Rua dos números, 123";
Linha2["Telefone"] = "555-5555";
Linha2["País"] = "Brasil";

```

```
// Adiciona a linha à coleção de linhas da tabela Clientes2.
```

```
Clientes2.Rows.Add(Linha2);
```

5. Por último faço a ligação entre o objeto DataTable – Clientes2 e o controle DataGrid – MinhaGrade2:

```
MinhaGrade2.DataSource = Clientes2.DefaultView;
MinhaGrade2.DataBind();
```

6. Neste momento, conforme pode ser confirmado pela Figura 12.1, temos duas novas linhas adicionadas ao objeto Clientes2. Este objeto segue o modelo “desconectado”, do ADO.NET. Isto significa que as alterações feitas no objeto DataTable (no caso adição de duas novas linhas) ainda não foram enviadas para o Banco de dados NorthWind.mdb. Isto pode ser facilmente comprovado.

Abra o Microsoft Access e abra o Banco de dados NorthWind.mdb. Crie uma consulta baseada na tabela Clientes e defina como critério “Brasil”, para o campo País. Serão exibidos apenas os Clientes do Brasil. Observe que os dois novos clientes (códigos XXYYK e WWMMK) não foram adicionados à tabela Clientes, conforme indicado na Figura 12.2.

7. Para atualizar o banco de dados, isto é, enviar de volta as alterações feitas no objeto DataTable, temos que utilizar alguns métodos do objeto DataAdapter. Aprenderemos a utilizar estes métodos mais adiante, neste capítulo.

Vamos aprender a utilizar mais alguns métodos do objeto DataTable, antes de aprendermos a sincronizar as alterações deste com o banco de dados.

Código do Cliente	Nome da Empresa	Endereço	Telefone	País
COMM1	Comércio Mineiro	Av. dos Lusíadas, 23	(11) 555-7647	Brasil
FAMIL	Família Arquibaldo	Rua Orós, 92	(11) 555-9857	Brasil
GOURL	Gourmet Lanchonetes	Av. Brasil, 442	(11) 555-9482	Brasil
HANAR	Hanari Carnes	Rua do Paço, 67	(21) 555-0091	Brasil
QUEDE	Que Delícia	Rua da Panificadora, 12	(21) 555-4252	Brasil
QUEEN	Queen Cozinha	Alameda dos Canários, 891	(11) 555-1189	Brasil
RICAR	Ricardo Adocicados	Av. Copacabana, 267	(21) 555-3412	Brasil
TRADH	Tradição Hipermercados	Av. Inês de Castro, 414	(11) 555-2167	Brasil
WELLI	Wellington Importadora	Rua do Mercado, 12	(14) 555-8122	Brasil

Figura 12.2: O Banco de dados NorthWind.mdb ainda não foi atualizado.

Exemplo 2: Vamos modificar um pouco o exemplo anterior. Ao invés de adicionarmos duas novas linhas, vamos excluir as duas primeiras linhas do objeto DataTable.

Vamos exibir os dados originais, retornados do Banco de dados NorthWind.mdb e a tabela Clientes2, após as exclusões dos registros.

Na Listagem 12.2 temos o código para o exemplo proposto. Vou excluir o comentário da parte básica da listagem, parte esta que já explicamos em exemplos anteriores. Somente incluirei comentários nos pontos da listagem onde temos novidades.

#### Listagem 12.2 – O objeto DataTable – Excluindo linhas.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>

<html>

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

protected void Page_Load(Object Src, EventArgs E)
{
 String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
 "DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";

 OleDbConnection MinhaConexão = new OleDbConnection(DefineConexão);

 String auxSQL1;
```

```
String auxSQL2;
String auxSQL3;
String ComandoSQL;

auxSQL1= "SELECT CódigoDoCliente,NomeDaEmpresa,Endereço,Telefone,Páis";
auxSQL2= " FROM Clientes Where País='Brasil'";
auxSQL3= " Order By NomeDaEmpresa";

ComandoSQL= auxSQL1+auxSQL2+auxSQL3;

OleDbDataAdapter MeuComando = new OleDbDataAdapter(ComandoSQL, MinhaConexão);

DataSet ds = new DataSet();
MeuComando.Fill(ds,"Clientes2");

DataView source = new DataView(ds.Tables[0]);

MinhaGrade1.DataSource = source ;
MinhaGrade1.DataBind();

// Agora vamos criar um objeto DataTable.
// Chamaremos este objeto de Clientes2, para
// distinguir da tabela Clientes, da coleção de
// Tabelas do DataSet.

DataTable Clientes2 = ds.Tables[0];

// Agora excluo as duas primeiras linhas
// índice 0 e índice1.
// Para excluir uma linha, utilizo o método
// Delete, da coleção Rows, do objeto DataTable.

Clientes2.Rows[0].Delete();
Clientes2.Rows[1].Delete();

// Neste momento as linhas foram apenas marcadas para exclusão,
// porém continuam no objeto DataTable, conforme pode ser confirmado
// pelos dados exibidos no controle MinhaGrade2.
```

```
// ****
// Vamos descartar as exclusões. Para isto chamaremos o método
// RejectChanges(), do objeto DataTable.
// A chamada deste método fará com que as linhas marcadas para
// exclusão sejam restauradas ao seu estatus padrão e, portanto,
// exibidas no DataGrid MinhaGrade2.

Clientes2.RejectChanges();

MinhaGrade2.DataSource = Clientes2.DefaultView;
MinhaGrade2.DataBind();

// Agora excluo novamente as duas primeiras linhas
// índice 0 e índice1.
// Para excluir uma linha, utilizo o método
// Delete, da coleção Rows, do objeto DataTable.
// ****

Clientes2.Rows[0].Delete();
Clientes2.Rows[1].Delete();

// Para fazer a exclusão definitiva das linhas, vamos
// chamar o método AcceptChanges(), do objeto DataTable.
// Após a chamada deste método, as linhas são excluídas
// do objeto DataTable, conforme pode ser confirmado
// pelos dados exibidos no controle MinhaGrade3.

Clientes2.AcceptChanges();

MinhaGrade3.DataSource = Clientes2.DefaultView;
MinhaGrade3.DataBind();

}

</script>

<body>
```

```
<%-- Exibe as informações do DataGrid no corpo da página. --%>
```

```
<ASP:DataGrid
 id="MinhaGrade1"
 runat="server"
 Width="700"
 BackColor="#c0c0c0"
 BorderColor="black"
 ShowFooter="false"
 CellPadding=3
 CellSpacing="0"
 Font-Name="Verdana"
 Font-Bold="True"
 Font-Size="7pt"
 HeaderStyle-BackColor="#aaaaadd"
 MaintainState="false"
/>

<ASP:DataGrid
 id="MinhaGrade2"
 runat="server"
 Width="700"
 BorderColor="black"
 ShowFooter="false"
 CellPadding=3
 CellSpacing="0"
 Font-Name="Verdana"
 Font-Size="7pt"
 HeaderStyle-Font-Bold="True"
 MaintainState="false"
/>

<ASP:DataGrid
 id="MinhaGrade3"
 runat="server"
```

```
Width="700"
BackColor="#c0c0c0"
BorderColor="black"
ShowFooter="false"
CellPadding=3
CellSpacing="0"
Font-Name="Courier New"
Font-Size="8pt"
Font-Bold="True"
HeaderStyle-BackColor="#aaaadd"
MaintainState="false"
/>
</body>

</html>
```

Digite o código da Listagem 12.2 e salve o mesmo em um arquivo chamado chap12ex2.aspx, na pasta chap12, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap12/chap12ex2.aspx>

Ao carregar a página você obtém o resultado indicado na Figura 12.3.

Comentários sobre o código do exemplo – Chap12ex2.aspx.

- ◆ Para excluir uma linha de um objeto DataTable, utilizamos o método Delete, da coleção de linhas Rows. A sintaxe para este método é a seguinte:

```
MeuDataTable.Rows[número_da_linha].Delete();
```

O número da primeira linha é zero, o da segunda linha é 1 e assim por diante.

Após a chamada deste método, a linha é marcada para exclusão. Podemos retornar a linha ao seu estado normal, chamando o método RejectChanges. A chamada deste método faz com que sejam descartadas todas as alterações efetuadas, desde a última chamada do método AcceptChanges. No nosso exemplo, excluímos as duas primeiras linhas, e depois chamamos o método RejectChanges, para cancelar as exclusões. Em seguida exibimos os dados do objeto Clientes2, no controle MinhaGrade2:

The screenshot shows a Microsoft Internet Explorer browser window with three identical tables displayed vertically. Each table has five columns: CódigoDoCliente, NomeDaEmpresa, Endereço, Telefone, and País. The data in all three tables is identical, representing ten different companies with their addresses and phone numbers. The first table has all rows selected for deletion. The second table has the first two rows selected for deletion. The third table shows the original data without any changes.

CódigoDoCliente	NomeDaEmpresa	Endereço	Telefone	País
COMM1	Comércio Mineiro	Av. dos Lusíadas, 23	(11) 555-7647	Brasil
FAMIA	Família Arquibaldo	Rua Orós, 92	(11) 555-9857	Brasil
GOURL	Gourmet Lanchonetes	Av. Brasil, 442	(11) 555-9482	Brasil
HANAR	Hanari Carnes	Rua do Paço, 67	(21) 555-0091	Brasil
QUEDE	Que Delícia	Rua da Panificadora, 12	(21) 555-4252	Brasil
QUEEN	Queen Cozinha	Alameda dos Canários, 891	(11) 555-1189	Brasil
RICAR	Ricardo Adocicados	Av. Copacabana, 267	(21) 555-3412	Brasil
TRADH	Tradição Hipermercados	Av. Inês de Castro, 414	(11) 555-2167	Brasil
WELLI	Wellington Importadora	Rua do Mercado, 12	(14) 555-8122	Brasil

CódigoDoCliente	NomeDaEmpresa	Endereço	Telefone	País
COMM1	Comércio Mineiro	Av. dos Lusíadas, 23	(11) 555-7647	Brasil
FAMIA	Família Arquibaldo	Rua Orós, 92	(11) 555-9857	Brasil
GOURL	Gourmet Lanchonetes	Av. Brasil, 442	(11) 555-9482	Brasil
HANAR	Hanari Carnes	Rua do Paço, 67	(21) 555-0091	Brasil
QUEDE	Que Delícia	Rua da Panificadora, 12	(21) 555-4252	Brasil
QUEEN	Queen Cozinha	Alameda dos Canários, 891	(11) 555-1189	Brasil
RICAR	Ricardo Adocicados	Av. Copacabana, 267	(21) 555-3412	Brasil
TRADH	Tradição Hipermercados	Av. Inês de Castro, 414	(11) 555-2167	Brasil
WELLI	Wellington Importadora	Rua do Mercado, 12	(14) 555-8122	Brasil

CódigoDoCliente	NomeDaEmpresa	Endereço	Telefone	País
COMM1	Comércio Mineiro	Av. dos Lusíadas, 23	(11) 555-7647	Brasil
FAMIA	Família Arquibaldo	Rua Orós, 92	(11) 555-9857	Brasil
GOURL	Gourmet Lanchonetes	Av. Brasil, 442	(11) 555-9482	Brasil
HANAR	Hanari Carnes	Rua do Paço, 67	(21) 555-0091	Brasil
QUEDE	Que Delícia	Rua da Panificadora, 12	(21) 555-4252	Brasil
QUEEN	Queen Cozinha	Alameda dos Canários, 891	(11) 555-1189	Brasil
RICAR	Ricardo Adocicados	Av. Copacabana, 267	(21) 555-3412	Brasil
TRADH	Tradição Hipermercados	Av. Inês de Castro, 414	(11) 555-2167	Brasil
WELLI	Wellington Importadora	Rua do Mercado, 12	(14) 555-8122	Brasil

Figura 12.3: Utilizando o objeto DataTable – Chap12ex2.aspx.

```

Clientes2.Rows[0].Delete();
Clientes2.Rows[1].Delete();

// Neste momento as linhas foram apenas marcadas para exclusão,
// porém continuam no objeto DataTable, conforme pode ser confirmado
// pelos dados exibidos no controle MinhaGrade2.

// Vamos descartar as alterações. Para isto chamaremos o método
// RejectChanges(), do objeto DataTable.

// A chamada deste método fará com que as linhas marcadas para
// exclusão sejam restauradas ao seu estatus padrão e, portanto,
// exibidas no DataGrid MinhaGrade2.

Clientes2.RejectChanges();
MinhaGrade2.DataSource = Clientes2.DefaultView;
MinhaGrade2.DataBind();

```

Conforme podemos comprovar na Figura 12.3, as linhas não foram excluídas.

- ◆ Em seguida marcamos as duas primeiras linhas para exclusão, porém agora chamamos o método AcceptChanges, para confirmar as exclusões. Exibimos os resultados no controle DataGrid3:

```
// Agora excluo novamente as duas primeiras linhas
// índice 0 e índice1.

// Para excluir uma linha, utilizo o método
// Delete, da coleção Rows, do objeto DataTable.
// ****

Clientes2.Rows[0].Delete();
Clientes2.Rows[1].Delete();

// Para fazer a exclusão definitiva das linhas, vamos
// chamar o método AcceptChanges(), do objeto DataTable.
// Após a chamada deste método, as linhas são excluídas
// do objeto DataTable, conforme pode ser confirmado
// pelos dados exibidos no controle MinhaGrade3

Clientes2.AcceptChanges();

MinhaGrade3.DataSource = Clientes2.DefaultView;
MinhaGrade3.DataBind();
```

Conforme podemos comprovar na Figura 12.3, as linhas foram realmente excluídas.

- ◆ Mais uma vez cabe ressaltar que estas exclusões não foram enviadas para o banco de dados. Foram feitas no objeto DataTable que é um objeto que segue o modelo “desconectado” do ADO.NET. Aprenderemos a sincronizar as alterações com o banco de dados ainda neste capítulo.

Exemplo 3: Vamos aprender a alterar os dados de uma linha do objeto DataSet. Vamos apresentar um exemplo, onde temos uma Caixa de Combinação com o nome de todos os Clientes do Brasil. O usuário seleciona um nome na lista e clica no botão Editar. O respectivo registro será retornado e os valores exibidos em controles do tipo TextBox, para que o usuário possa alterá-los, com exceção do campo CódigoDoCliente. Uma vez feitas as alterações necessárias, o usuário clica no botão Salvar. Os novos valores são salvos e a listagem de Clientes é novamente exibida para que possamos confirmar se as alterações foram feitas.

Na Listagem 12.3 temos o código para o exemplo proposto. Vou excluir o comentário da parte básica da listagem, parte esta que já explicamos em exemplos anteriores. Somente incluirei comentários nos pontos da listagem onde temos novidades. O exemplo é um pouco longo pois, além das técnicas para edição de registros, vamos aprender a tratar eventos, localizar dados automaticamente, ocultar e exibir controles utilizando código e outras técnicas úteis.

Listagem 12.3 – O objeto DataTable – Editando linhas.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>

<html>

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

// Declaro variáveis que serão globais para a página.

OleDbDataAdapter MeuComando;
String auxSQL1;
String auxSQL2;
String comandoSQL;
DataSet ds = new DataSet();
OleDbConnection MinhaConexão;
String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
"DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";

protected void Page_Load(Object Src, EventArgs E)
{
 if (!Page.IsPostBack)
 {
 OcultaControles();

 // Defino a string para o comando SQL;

 String auxSQL1;
 String auxSQL2;
 String comandoSQL;

 auxSQL1 = "Select CódigoDoCliente,NomeDaEmpresa,Endereço,País,Telefone From
Clientes";
 auxSQL2 = " Where País='Brasil'" + "Order By NomeDaEmpresa";

 comandoSQL = auxSQL1+auxSQL2;
 MinhaConexão = new OleDbConnection(DefineConexão);
 }
}
```

```
MeuComando = new OleDbDataAdapter(comandoSQL, MinhaConexão);

// Utilizo o método Fill do objeto DataAdapter, para preencher
// o objeto DataSet, com os dados retornados pelo comando SQL.

MeuComando.Fill(ds);

// Conectamos um controle DropDownList com o DataSet criado anteriormente.
// MinhaLista é o id (nome) de um controle do tipo
// DropDownList que está na seção de apresentação da página.

DataView source = new DataView(ds.Tables[0]);

// Ligo o objeto DataView a um controle do tipo
// ListBox

MinhaLista.DataSource = source ;
MinhaLista.DataBind();

}

}

void OcultaControles()
{
 txtNovoCódigoDoCliente.Visible =false;
 txtNovoNomeDaEmpresa.Visible = false;
 txtNovoEndereço.Visible=false;
 txtNovoTelefone.Visible=false;
 txtNovoPaís.Visible=false;
 txtMostraAtualizado.Visible=false;
 txtMensagem.Visible=false;
}

void ClienteSelecionado(Object sender, EventArgs e)
{
}
```

```
OcultaControles();

// Localiza o registro, correspondente ao
// cliente selecionado na lista.

// Defino a string para o comando SQL;

String auxSQL1;
String auxSQL2;
String comandoSQL;

auxSQL1 = "Select CódigoDoCliente,NomeDaEmpresa,Endereço,País,Telefone
From Clientes";
auxSQL2 = " Where País='Brasil'" + "Order By NomeDaEmpresa";

comandoSQL = auxSQL1+auxSQL2;

MinhaConexão = new OleDbConnection(DefineConexão);
MeuComando = new OleDbDataAdapter(comandoSQL, MinhaConexão);

// Utilizo o método Fill do objeto DataAdapter, para preencher
// o objeto DataSet, com os dados retornados pelo comando SQL.

MeuComando.Fill(ds);

// Conectamos um controle DropDownList com o DataSet criado anteriormente.
// MinhaLista é o id (nome) de um controle do tipo
// DropDownList que está na seção de apresentação da página.

DataView source = new DataView(ds.Tables[0]);

DataTable Clientes = ds.Tables[0];

// Vamos definir o campo CódigoDoCliente como sendo a chave
// primária da tabela Clientes.

Clientes.PrimaryKey = new DataColumn[] {Clientes.Columns["CódigoDoCliente"]};
```

```
DataRow Linha = Clientes.Rows.Find(MinhaLista.SelectedItem.Value);

// Defino os valores dos controles TextBox da seção de apresentação.

txtCódigoDoCliente.Text = Linha["CódigoDoCliente"].ToString();
txtNomeDaEmpresa.Text = Linha["NomeDaEmpresa"].ToString();
txtEndereço.Text = Linha["Endereço"].ToString();
txtTelefone.Text = Linha["Telefone"].ToString();
txtPaís.Text = Linha["País"].ToString();

}

void AtualizarRegistro(Object sender, EventArgs e)
{
 // Localiza o registro, correspondente ao
 // cliente que está sendo atualizado.

 // Defino a string para o comando SQL;

 String auxSQL1;
 String auxSQL2;
 String comandoSQL;

auxSQL1 = "Select CódigoDoCliente,NomeDaEmpresa,Endereço,País,Telefone From
Clientes";
auxSQL2 = " Where País='Brasil'" + "Order By NomeDaEmpresa";

comandoSQL = auxSQL1+auxSQL2;

MinhaConexão = new OleDbConnection(DefineConexão);
MeuComando = new OleDbDataAdapter(comandoSQL, MinhaConexão);
// Utilizo o método Fill do objeto DataAdapter, para preencher
// o objeto DataSet, com os dados retornados pelo comando SQL.

MeuComando.Fill(ds);
DataView source = new DataView(ds.Tables[0]);
```

```
DataTable Clientes = ds.Tables[0];

// Vamos definir o campo CódigoDoCliente como sendo a chave
// primária da tabela Clientes.

Clientes.PrimaryKey = new DataColumn[] {Clientes.Columns["CódigoDoCliente"]};

DataRow Linha = Clientes.Rows.Find(MinhaLista.SelectedItem.Value);

// Declaro algumas variáveis auxiliares para
// armazenar os valores originais para os campos
// do registro que está sendo editado.

String auxCódigoDoCliente = Linha["CódigoDoCliente"].ToString();
String auxNomeDaEmpresa = Linha["NomeDaEmpresa"].ToString();
String auxEndereço = Linha["Endereço"].ToString();
String auxTelefone = Linha["Telefone"].ToString();
String auxPaís = Linha["País"].ToString();

// Agora vamos editar o valor da linha.

Linha.BeginEdit();

 Linha["CódigoDoCliente"] = txtCódigoDoCliente.Text;
 Linha["NomeDaEmpresa"] = txtNomeDaEmpresa.Text;
 Linha["Endereço"] = txtEndereço.Text;
 Linha["Telefone"] = txtTelefone.Text;
 Linha["País"] = txtPaís.Text;

Linha.EndEdit();

// Defino o valor dos controles da terceira coluna
// como sendo igual aos novos valores da linha

txtMostraAtualizado.Visible=true;

txtNovoCódigoDoCliente.Text = Linha["CódigoDoCliente"].ToString();
txtNovoCódigoDoCliente.Visible =true;
```

```
txtNovoNomeDaEmpresa.Text = Linha["NomeDaEmpresa"].ToString();
txtNovoNomeDaEmpresa.Visible = true;

txtNovoEndereço.Text = Linha["Endereço"].ToString();
txtNovoEndereço.Visible=true;

txtNovoTelefone.Text = Linha["Telefone"].ToString();
txtNovoTelefone.Visible = true;

txtNovoPaís.Text = Linha["País"].ToString();
txtNovoPaís.Visible=true;

txtMensagem.Visible=true;

// Exibo na segunda coluna os valores originais,
// obtidos a partir das variáveis auxiliares
// criadas anteriormente.

txtCódigoDoCliente.Text = auxCódigoDoCliente;
txtNomeDaEmpresa.Text = auxNomeDaEmpresa;
txtEndereço.Text = auxEndereço;
txtTelefone.Text = auxTelefone;
txtPaís.Text = auxPaís;

}

</script>

<body>

<h3>

 Seleccione um cliente na lista,

 para editar o registro do cliente.

</h3>
<HR>
```

```
<form runat=server>

<div align="left">
 <table border="0">
 <tr>

 <td>
 <p align="right">

 Seleccione um cliente ->>

 </p>
 </td>

 <td>
 <asp:DropDownList
 id="MinhaLista"
 runat="server"
 BackColor="#c0c0c0"
 Font-Bold="True"
 DataTextField = "NomeDaEmpresa"
 DataValueField = "CódigoDoCliente"
 AutoPostBack = "True"
 onSelectedIndexChanged = "ClienteSelecionado"
 >
 </asp:DropDownList>
 </td>

 <td>
 <asp:TextBox
 runat=server
 id="txtMostraAtualizado"
 Text="REGISTRO ATUALIZADO !!!"
 TextMode="SingleLine"
 Font_Face="Arial"
 Font_Size="3"
 </asp:TextBox>
 </td>
 </tr>
 </table>
</div>
```

```
Font-Bold="True"
BackColor="#c0c0c0"
Enabled="False"
Visible="False"
Height="20"
Width="200"
/>
</td>

</tr>

<tr>
<td>
<p align="right">
Código do Cliente:
</td>
<td>
<asp:TextBox
 runat=server
 id="txtCódigoDoCliente"
 Text=""
 TextMode="SingleLine"
 Font_Face="Arial"
 Font_Size="3"
 Height="20"
 Width="200"
 Enabled="False"
 />
</td>
<td>
<asp:TextBox
 runat=server
```

```
 id="txtNovoCódigoDoCliente"
 Text=""
 BackColor="#c0c0c0"
 TextMode="SingleLine"
 Font_Face="Arial"
 Font_Size="3"
 Height="20"
 Width="200"
 Enabled="False"
 Visible="False"
 />
</td>
</tr>

<tr>

 <td>
 <p align="right">
 Nome da Empresa:
 </td>
 <td>
 <asp:TextBox
 runat=server
 id="txtNomeDaEmpresa"
 Text=""
 TextMode="SingleLine"
 Font_Face="Arial"
 Font_Size="3"
 Height="20"
 Width="200"
 />
 </td>
 <td>
```

```
<asp:TextBox
 runat=server
 id="txtNovoNomeDaEmpresa"
 Text=""
 BackColor="#c0c0c0"
 TextMode="SingleLine"
 Font_Face="Arial"
 Font_Size="3"
 Height="20"
 Width="200"
 Visible="False"
/>

</td>
</tr>

<tr>
 <td>
 <p align="right">
 Endereço:
 </td>

 <td>

 <asp:TextBox
 runat=server
 id="txtEndereço"
 Text=""
 TextMode="SingleLine"
 Font_Face="Arial"
 Font_Size="3"
 Height="20"
 Width="200"
 />

</td>
```

```
<td>

<asp:TextBox
 runat=server
 id="txtNovoEndereço"
 Text=""
 BackColor="#c0c0c0"
 TextMode="SingleLine"
 Font_Face="Arial"
 Font_Size="3"
 Height="20"
 Width="200"
 Visible="False"
/>

</td>
</tr>

<tr>

<td>

Telefone:

</td>

<td>
<asp:TextBox
 runat=server
 id="txtTelefone"
 Text=""
 TextMode="SingleLine"
 Font_Face="Arial"
 Font_Size="3"
 Height="20"
 Width="200"
/>
```

```
</td>

<td>
<asp:TextBox
 runat=server
 id="txtNovoTelefone"
 Text=""
 BackColor="#c0c0c0"
 TextMode="SingleLine"
 Font_Face="Arial"
 Font_Size="3"
 Height="20"
 Width="200"
 Visible="False"
/>

</td>
</tr>

<tr>

<td>
<p align="right">
País:
</td>

<td>
<asp:TextBox
 runat=server
 id="txtPaís"
 Text=""
 TextMode="SingleLine"
 Font_Face="Arial"
 Font_Size="3"
 Height="20"
 Width="200"

```

```
>

</td>

<td>
<asp:TextBox
 runat="server"
 id="txtNovoPaís"
 Text=""
 BackColor="#c0c0c0"
 TextMode="SingleLine"
 Font_Face="Arial"
 Font_Size="3"
 Height="20"
 Width="200"
 Visible="False"
/>

</td>
</tr>
<tr>
<td>
<p align="right">
Clique no botão Atualizar
</td>

<td>
<asp:Button
 id="AtualizaDados"
 Text="Atualizar dados"
 runat="server"
 OnClick="AtualizarRegistro"
/>

</td>
</td>

```

```
<asp:TextBox
 runat="server"
 id="txtMensagem"
 Text="Atualizado com sucesso !!!"
 TextMode="SingleLine"
 Font_Face="Arial"
 Font_Size="3"
 Font-Bold="True"
 BackColor="#c0c0c0"
 Visible="False"
 Enabled="False"
 Height="20"
 Width="200"
 />
 </td>

</table>
</div>

<asp:Label id="Label1" runat="server"/>

</form>
</body>
</html>
```

Digite o código da Listagem 12.3 e salve o mesmo em um arquivo chamado chap12ex3.aspx, na pasta chap12, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap12/chap12ex3.aspx>

Ao carregar a página você obtém o resultado indicado na Figura 12.4, onde é apresentada uma lista com o nome dos clientes do Brasil e cinco controles do tipo TextBox, em branco.

Na lista de Clientes selecione “Tradição Hipermercados”. Observe que, automaticamente, as informações do registro referente ao cliente selecionado são exibidas nos controles do formulário, conforme indicado na Figura 12.5.

Clique no campo Nome da Empresa e altere o valor para: Tradição Hipermercados XYZ.

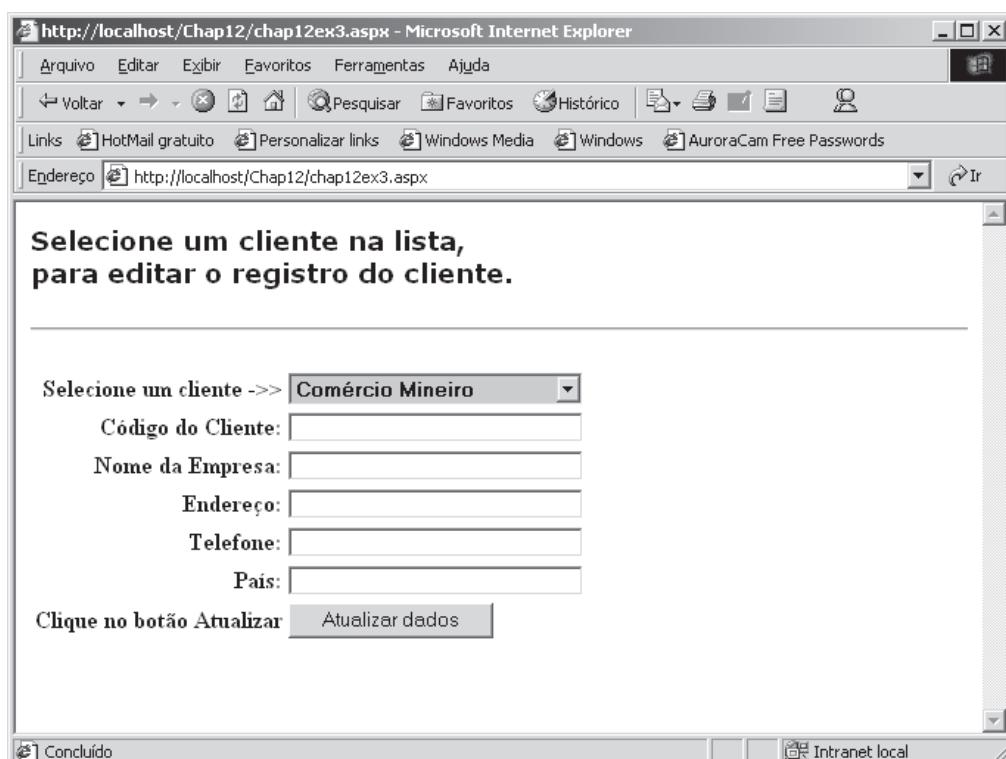


Figura 12.4: Uma lista com os clientes do Brasil – Chap12ex3.aspx.

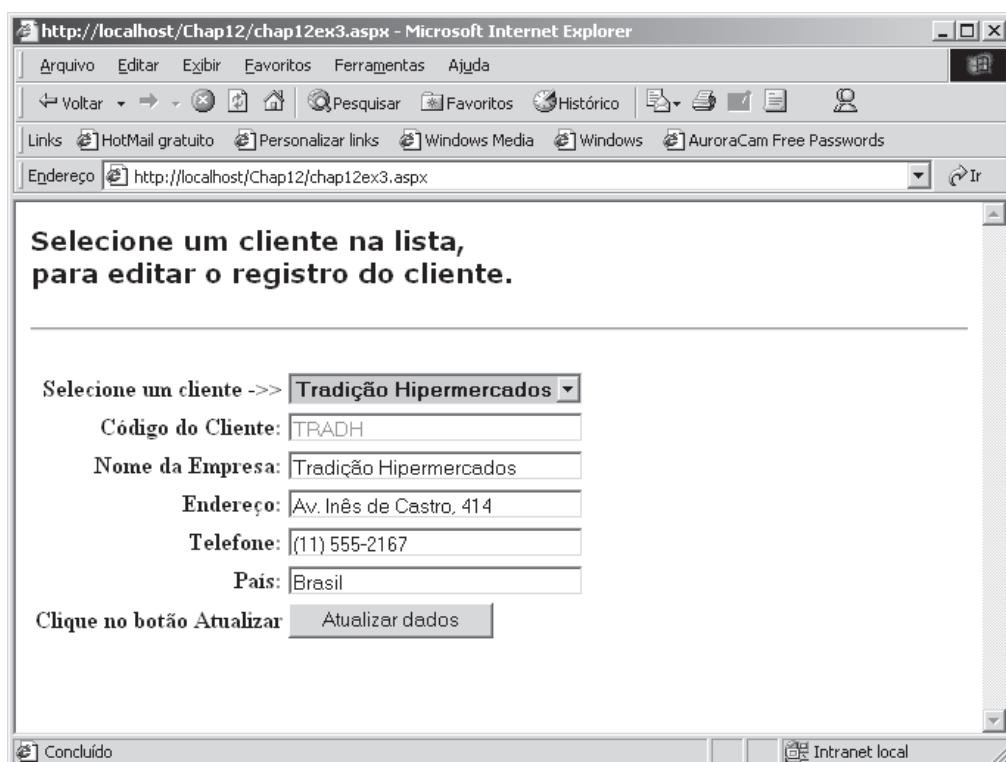
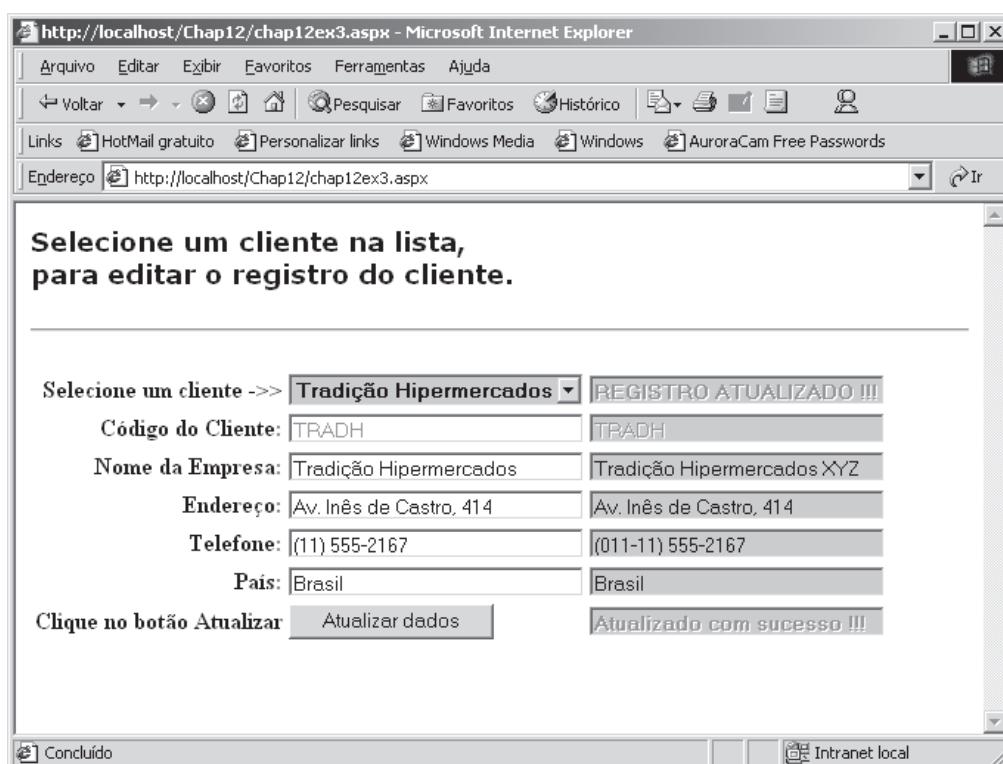


Figura 12.5: O registro do cliente é automaticamente localizado.

Clique no campo Telefone e altere o número do telefone para: (011-11) 555-2167.

Dê um clique no botão Atualizar dados. Na coluna do meio são exibidos os valores originais do registro para o cliente Tradição Hipermercados e, na coluna da direita, são exibidos os valores já alterados, conforme indicado na Figura 12.6.



**Figura 12.6: Exibindo os valores antes e depois da edição.**

Cabe, mais uma vez, ressaltar que os valores foram alterados no objeto DataTable, desconectado do banco de dados. As alterações ainda não foram enviadas para o Banco de dados NorthWind.mdb. Para comprovar isto, basta abrir o Banco de dados NorthWind.mdb e verificar que o registro para o cliente Tradição Hipermercados não foi alterado. No próximo tópico aprenderemos a sincronizar o objeto DataTable com o banco de dados.

Comentários sobre o código do exemplo – Chap12ex3.aspx.

- ◆ O exemplo é um pouco extenso, porém veremos que a sua funcionalidade é bastante simples.
  - ◆ Declaramos algumas variáveis com escopo de página, ou seja, fora de qualquer procedimento. Estas variáveis serão utilizadas em diversos procedimentos da página, e por isso foram declaradas com escopo de página:
- ```
// Declaro variáveis que serão globais para a página.
```

```
OleDbDataAdapter MeuComando;
String auxSQL1;
String auxSQL2;
String comandoSQL;
DataSet ds = new DataSet();
OleDbConnection MinhaConexão;
String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
"DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";
```

- ◆ No evento Page_Load da página, fazemos a conexão com o Banco de dados NorthWind.mdb, retornamos dados da tabela Clientes e fazemos a ligação com o controle MinhaLista. O controle MinhaLista exibe uma listagem com o nome dos clientes do Brasil. O código do evento Page_Load somente é executado quando a página é carregada pela primeira vez, ou seja, quando não for um PostBack. Observe que, no início do procedimento, testamos se é ou não um PostBack:

```
if (!Page.IsPostBack)
```

O operador “!” é o operador not. O teste é equivalente a dizer: “Se não for um PostBack, execute o código dentro do if”. O código para conexão com o banco de dados e ligação com o controle MinhaLista é semelhante ao código utilizado em exemplos anteriores.

No evento Page_Load, também fazemos uma chamada ao procedimento OcultaControles. Este procedimento oculta todos os controles da terceira coluna, os quais somente voltarão a ser exibidos quando o usuário editar um registro:

```
OcultaControles();
```

Na definição do controle MinhaLista, definimos diferentes valores para as propriedades DataTextField. A propriedade DataTextField define qual o campo que fornece os valores que serão exibidos para cada item da lista. No nosso exemplo, como queríamos que fossem exibidos os nomes dos clientes, definimos a propriedade DataTextField=“NomeDaEmpresa”. A propriedade DataValueField define qual o valor associado com cada item da lista. Em outras palavras, quando o usuário seleciona um cliente na lista, qual o valor que ficará associado com a propriedade Value da lista. Poderia ser o mesmo campo que define o texto de cada item. Porém, para o nosso exemplo, queremos que, ao selecionar um cliente, seja definido, como valor do elemento selecionado da lista, o respectivo CódigoDoCliente. Por isso definimos a propriedade DataValueField=“CódigoDoCliente”. Na prática, quando o usuário selecionar, por exemplo, o cliente “Tradição Hipermercados”, o valor associado será “TRADH”, que é o código do cliente “Tradição Hipermercados”. Mais adiante veremos que o código do cliente é utilizado para localizar o registro do cliente. A seguir temos a definição do controle MinhaLista:

```
<asp:DropDownList
    id="MinhaLista"
    runat="server"
    BackColor="#c0c0c0"
    Font-Bold="True"
    DataTextField = "NomeDaEmpresa"
    DataValueField = "CódigoDoCliente"
    AutoPostBack = "True"
    onSelectedIndexChanged = "ClienteSelecionado"
>
</asp:DropDownList>
```

- ◆ Para alinhar os diversos controles do formulário, utilizamos uma tabela de três colunas. Na primeira coluna são exibidos os rótulos para cada campo. Na segunda coluna são exibidos os valores originais do registro. Na terceira coluna são exibidos os valores após o usuário ter feito alterações.

- ◆ Quando é selecionado um cliente, na lista de clientes, é disparado o evento onSelectedIndexChanged. Para tratar este evento, definimos o procedimento ClienteSelecionado. Vamos analisar o funcionamento deste procedimento.

O procedimento ClienteSelecionado inicia ocultando os controles da terceira coluna:

```
OcultaControles();
```

Depois executamos uma série de comandos para conectar com o banco de dados. Estes comandos são idênticos aos utilizados no evento Page_Load.

Em seguida criamos um objeto DataTable, associado à primeira tabela do objeto DataSource – ds:

```
DataTable Clientes = ds.Tables[0];
```

Agora vamos definir a chave primária para a tabela Clientes:

```
Clientes.PrimaryKey = new DataColumn[] { Clientes.Columns["CódigoDoCliente"] };
```

Este comando precisa de mais alguns comentários. Para definir a chave primária, utilizamos a propriedade PrimaryKey, do objeto DataTable. Devemos passar um objeto do tipo DataColumn para a propriedade PrimaryKey. O objeto DataColumn é um array de objetos do tipo Column. O Array contém a referência a uma ou mais colunas, que formam a chave primária. No nosso exemplo, definimos apenas a coluna CódigoDoCliente, como sendo a chave primária.

Agora é hora de criar um objeto do tipo Linha, o qual contém os valores para o cliente selecionado. Para localizar o registro do cliente que foi selecionado na lista de clientes, utilizamos o método Find, da coleção Rows, da tabela Clientes:

```
DataRow Linha = Clientes.Rows.Find(MinhaLista.SelectedItem.Value);
```

Como parâmetro para o método Find, nós passamos o valor selecionado na lista MinhaLista. O método Find recebe um valor e pesquisa na coluna Chave Primária da tabela. Caso encontre o valor passado como parâmetro, o registro correspondente será retornado. Observe que definimos o campo CódigoDoCliente como Chave Primária da tabela Clientes e a propriedade DataValueField, da lista de clientes como sendo CódigoDoCliente. Agora ficou mais claro o porquê desta definição. Ao selecionarmos um cliente na lista, é associado o valor do CódigoDoCliente, com o item selecionado. Este valor é utilizado para localizar o registro do cliente.

Uma vez localizado o registro do cliente, vamos definir o conteúdo dos controles do tipo TextBox, da segunda coluna, como sendo igual aos valores dos campos do registro do cliente:

```
// Defino os valores dos controles TextBox da seção de apresentação.
```

```
txtCódigoDoCliente.Text = Linha["CódigoDoCliente"].ToString();
txtNomeDaEmpresa.Text = Linha["NomeDaEmpresa"].ToString();
txtEndereço.Text = Linha["Endereço"].ToString();
txtTelefone.Text = Linha["Telefone"].ToString();
txtPaís.Text = Linha["País"].ToString();
```

Em resumo, o procedimento ClienteSelecionado faz o seguinte:

1. Oculta os controles da terceira coluna.
 2. Conecta com o banco de dados.
 3. Cria um objeto DataTable – Clientes.
 4. Localiza o registro correspondente ao cliente selecionado na lista de clientes.
 5. Exibe os dados do cliente, nos controles TextBox da segunda coluna.
- ◆ Uma vez exibidos os dados do cliente, o usuário poderá alterar o conteúdo dos campos, com exceção do campo CódigoDoCliente, que está desabilitado para alterações (Enabled="False"). O usuário faz as alterações necessárias e clica no botão “Atualizar dados”. Ao clicar neste botão é disparado o procedimento “AtualizaRegistro”.

O procedimento AtualizaRegistro faz a conexão com o banco de dados e localiza o registro do cliente que está sendo editado. Neste caso são retornados os valores originais do registro, sem as alterações feitas pelo usuário. Esta parte do procedimento é idêntica à utilizada no procedimento ClienteSelecionado.

Uma vez localizado o registro, com os valores originais, utilizamos uma série de variáveis auxiliares, para armazenar os valores originais para o registro do cliente:

```
String auxCódigoDoCliente = Linha["CódigoDoCliente"].ToString();
String auxNomeDaEmpresa = Linha["NomeDaEmpresa"].ToString();
String auxEndereço = Linha["Endereço"].ToString();
String auxTelefone = Linha["Telefone"].ToString();
String auxPaís = Linha["País"].ToString();
```

Estas variáveis serão utilizadas para exibir os valores originais, na segunda coluna.

Finalmente chegamos ao momento de fazer a edição no registro do cliente selecionado. Para editar um registro, chamamos o método BeginEdit da linha, atribuímos os novos valores a cada campo da linha e chamamos o método EndEdit();

```
// Agora vamos editar o valor da linha.

Linha.BeginEdit();

    Linha["CódigoDoCliente"]      = txtCódigoDoCliente.Text;
    Linha["NomeDaEmpresa"]        = txtNomeDaEmpresa.Text;
    Linha["Endereço"]            = txtEndereço.Text;
    Linha["Telefone"]             = txtTelefone.Text;
    Linha["País"]                = txtPaís.Text;

Linha.EndEdit();
```

Em seguida tornamos os controles da terceira coluna visíveis e exibimos, nestes controles, o registro já alterado:

```
// Defino o valor dos controles da terceira coluna  
// como sendo igual aos novos valores da linha  
txtMostraAtualizado.Visible=true;  
  
txtNovoCódigoDoCliente.Text = Linha["CódigoDoCliente"].ToString();  
txtNovoCódigoDoCliente.Visible =true;  
  
txtNovoNomeDaEmpresa.Text = Linha["NomeDaEmpresa"].ToString();  
txtNovoNomeDaEmpresa.Visible = true;  
  
txtNovoEndereço.Text = Linha["Endereço"].ToString();  
txtNovoEndereço.Visible=true;  
  
txtNovoTelefone.Text = Linha["Telefone"].ToString();  
txtNovoTelefone.Visible = true;  
  
txtNovoPaís.Text = Linha["País"].ToString();  
txtNovoPaís.Visible=true;  
  
txtMensagem.Visible=true;
```

Para finalizar, exibimos, na segunda coluna, os valores originais do registro. Para isso fazemos uso das variáveis auxiliares criadas anteriormente, variáveis estas que contêm os valores originais do registro, isto é, antes da edição:

```
// Exibo na segunda coluna os valores originais,  
// obtidos a partir das variáveis auxiliares  
// criadas anteriormente.  
txtCódigoDoCliente.Text = auxCódigoDoCliente;  
txtNomeDaEmpresa.Text = auxNomeDaEmpresa;  
txtEndereço.Text = auxEndereço;  
txtTelefone.Text = auxTelefone;  
txtPaís.Text = auxPaís;
```

Em resumo, o procedimento AtualizaRegistro faz o seguinte:

1. Conecta com o banco de dados.
2. Cria um objeto DataTable – Clientes.
3. Localiza o registro correspondente ao cliente selecionado na lista de clientes.
4. Atribui os valores antes da edição a variáveis auxiliares.

5. Edita o registro.
 6. Exibe os novos valores na terceira coluna.
 7. Exibe os valores originais na segunda coluna.
- ♦ O exemplo é longo mas salienta uma série de técnicas úteis que podem ser, facilmente, adaptadas para as aplicações que o leitor venha a desenvolver com ASP.NET.

Agora é hora de aprendermos a sincronizar as alterações feitas em um objeto DataTable, com o banco de dados, de tal forma que estas alterações sejam enviadas de volta ao banco de dados.

Sincronizando Dados com o Banco de Dados

Para sincronizar as alterações feitas nos dados desconectados, com o banco de dados, utilizaremos algumas propriedades do objeto OleDbDataAdapter/SqlDataAdapter. Também utilizaremos o método Update, deste objeto.

O processo para enviar as alterações para o banco de dados é bastante simples:

1. Definimos uma chave primária para o objeto DataTable.
2. Criamos um objeto do tipo OleDbComandBuilder/SqlCommandBuilder. Na criação deste objeto passamos o nome do objeto DataAdapter como parâmetro.
3. Defino as seguintes propriedades do objeto DataAdapter:

InsertCommand: Um comando SQL que envia, para o banco de dados, as novas linhas do objeto DataTable.

DeleteCommand: Um comando SQL que exclui do banco de dados as linhas que foram excluídas no objeto DataTable.

UpdateCommand: Um comando SQL que envia, para o banco de dados, as alterações feitas nas linhas do objeto DataTable.

Estas propriedades podem ser definidas manualmente ou através da utilização de um objeto OleDbCommandBuilder/SqlCommandBuilder. O objeto CommandBuilder possui um método GetDeleteCommand para gerar, automaticamente, o comando para enviar as exclusões para o banco de dados; possui um método GetInsertCommand para gerar, automaticamente, o comando para inserir as novas linhas no banco de dados; e um método GetUpdateCommand para gerar, automaticamente, o comando para enviar as alterações para o banco de dados.

4. Uma vez definidos os comandos necessários, chamamos o método Update do objeto DataAdapter. Este método executa, no banco de dados, os comandos definidos nas propriedades InsertCommand, DeleteCommand e UpdateCommand. Caso tenhamos definido apenas uma destas propriedades, somente os respectivos comandos serão executados. Por exemplo, se construirmos um formulário para cadastrar clientes, apenas precisaremos definir a propriedade InsertCommand.

Vamos lembrar do exemplo – Chap12ex1.aspx. Neste exemplo inserimos duas linhas no objeto DataTable e comprovamos que estas adições não foram enviadas para o Banco de dados NorthWind.mdb. Se quisermos atualizar

o Banco de dados NorthWind.mdb, basta incluir as seguintes linhas de código, no final do procedimento Page_Load, do referido exemplo:

```
// Em primeiro lugar, defino uma chave primária para o objeto DataTable.  
  
Clientes2.PrimaryKey = new DataColumn[] {Clientes2.Columns["CódigoDoCliente"]};  
  
// Agora crio os comandos necessários para enviar  
// as alterações/inclusões/exclusões para o banco  
// de dados NorthWind.mdb.  
// No nosso exemplo, apenas o comando para adição das linhas.  
// Em primeiro lugar crio um objeto do tipo OleDbCommandBuilder  
  
OleDbCommandBuilder CriaComando = new OleDbCommandBuilder(MeuCommando);  
  
// Agora defino a propriedade InsertCommand do objeto MeuDataAdapter.  
  
MeuDataAdapter.InsertCommand= CriaComando.GetInsertCommand();  
MeuDataAdapter.Update(ds,"Clientes2");
```

Ao inserir estes comandos, no final do procedimento Page_Load e recarregar a página, dois novos registros serão inseridos no banco de dados. Você pode abrir o Microsoft Access, carregar o Banco de dados NorthWind.mdb e conferir. Os registros realmente foram inseridos.

Observe o seguinte trecho da mensagem de erro:

- ◆ **Exception Details:** System.Data.ConstraintException: Column ‘CódigoDoCliente’ is constrained to be unique. Value ‘XXYYK’ is already present.

Este trecho informa que já existe o cliente com CódigoDoCliente=”XXYYK”. Este cliente foi inserido na primeira vez que carregamos a página. Esta é mais uma prova de que o método Update realmente enviou as adições para o banco de dados.

Vamos apresentar um exemplo onde criamos um formulário para cadastro de clientes. O formulário apresenta diversos campos a serem preenchidos. O usuário preenche os campos e clica no botão Cadastrar. Os dados são enviados para o Banco de dados NorthWind.mdb. Faremos uso dos Controles de Validação, vistos no Capítulo 8. Utilizaremos controles de validação para garantir que os seguintes campos sejam preenchidos, ou seja, são campos obrigatórios:

- ◆ CódigoDoCliente
- ◆ NomeDaEmpresa

IMPORTANTE: Na segunda vez que a página for carregada, será gerado um erro. Isso acontece porque, na segunda vez, estamos tentando cadastrar um cliente, com um CódigoDoCliente que já existe. Como o campo CódigoDoCliente é do tipo Chave Primária, não podem existir dois clientes com o mesmo código. Este erro é mostrado na Figura 12.7.

- ◆ Endereço
- ◆ Telefone
- ◆ País

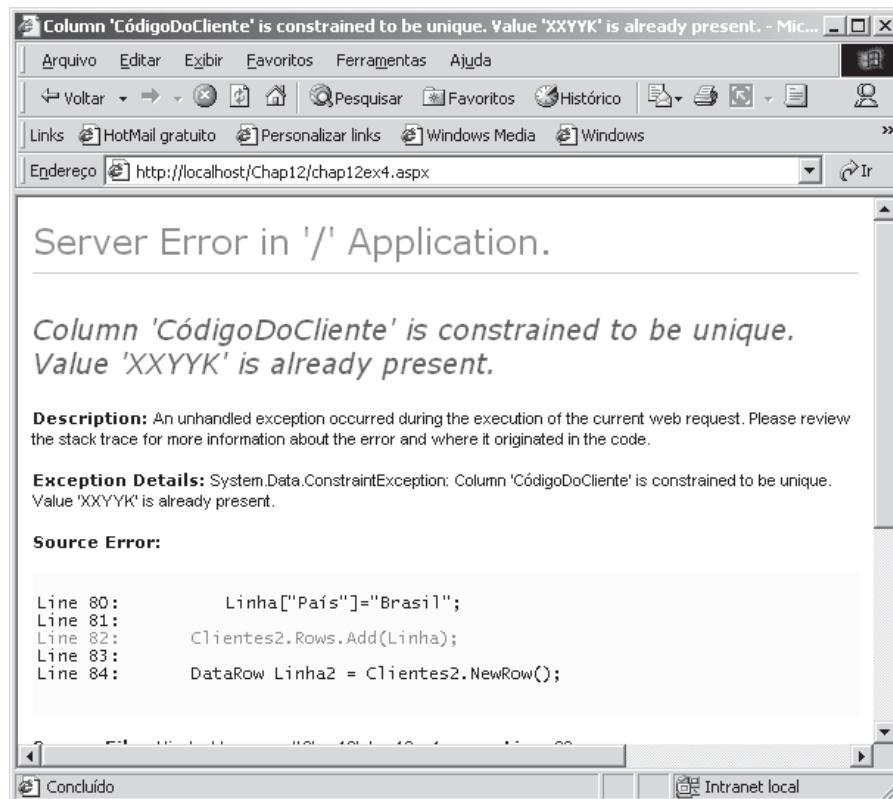


Figura 12.7: Erro – CódigoDoCliente repetido.

Os demais campos não são de preenchimento obrigatório.

Na Listagem 12.4 temos o código para o exemplo proposto. Vou excluir o comentário da parte básica da listagem, parte esta que já explicamos em exemplos anteriores. Somente incluirei comentários nos pontos da listagem em que temos novidades.

Listagem 12.4 – Formulário para Cadastro de Clientes – Chap12ex4.aspx.

```

<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>

<html>

<script language="C#" runat="server">
    // Declaro variáveis que serão globais para a página.

    OleDbDataAdapter MeuDataAdapter;

```

```
String auxSQL1;
String auxSQL2;
String comandoSQL;
DataSet ds = new DataSet();
OleDbConnection MinhaConexão;
String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
    "DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";

void InserirCliente(Object sender, EventArgs e)
{
    // Defino a string para o comando SQL;

    String comandoSQL;
    comandoSQL = "Select * From Clientes";

    MinhaConexão = new OleDbConnection(DefineConexão);
    MeuDataAdapter = new OleDbDataAdapter(comandoSQL, MinhaConexão);

    MeuDataAdapter.Fill(ds,"Clientes");

    DataView source = new DataView(ds.Tables[0]);

    DataTable Clientes = ds.Tables[0];

    // Vamos definir o campo CódigoDoCliente como sendo a chave
    // primária da tabela Clientes.

    Clientes.PrimaryKey = new DataColumn[] {Clientes.Columns["CódigoDoCliente"]};

    // Chamo o método NewRow da tabela Clientes.

    DataRow Linha = Clientes.NewRow();

    // Defino os valores para a linha a ser inserida.
```

```

Linha["CódigoDoCliente"]      = txtCódigoDoCliente.Text;
Linha["NomeDaEmpresa"]        = txtNomeDaEmpresa.Text;
Linha["NomeDoContato"]        = txtNomeDoContato.Text;
Linha["CargoDoContato"]       = txtCargoDoContato.Text;
Linha["Endereço"]             = txtEndereço.Text;
Linha["Cidade"]               = txtCidade.Text;
Linha["Região"]               = txtRegião.Text;
Linha["CEP"]                  = txtCEP.Text;
Linha["País"]                 = txtPaís.Text;
Linha["Telefone"]             = txtTelefone.Text;
Linha["Fax"]                  = txtFax.Text;

Clientes.Rows.Add(Linha);

// Agora crio os comandos necessários para enviar
// as alterações/inclusões/exclusões para o banco
// de dados NorthWind.mdb

// Em primeiro lugar crio um objeto do tipo OleDbCommandBuilder

OleDbCommandBuilder CriaComando = new OleDbCommandBuilder(MeuDataAdapter);

// Agora defino a propriedade InsertCommand do objeto MeuDataAdapter.

MeuDataAdapter.InsertCommand= CriaComando.GetInsertCommand();

// Chamo o método Update do objeto DataAdapter.

MeuDataAdapter.Update(ds,"Clientes");

// Informo que o registro foi inserido com sucesso.

Label1.Text= "Registro para o cliente: " + txtNomeDaEmpresa.Text + " Inserido com
sucesso";
Label2.Text="Preencha os campos abaixo para cadastrar outro Cliente";

// Limpo o valor dos campos TextBox.

```

```
txtCódigoDoCliente.Text ="";  
txtNomeDaEmpresa.Text ="";  
txtNomeDoContato.Text ="";  
txtCargoDoContato.Text ="";  
txtEndereço.Text ="";  
txtCidade.Text ="";  
txtRegião.Text ="";  
txtCEP.Text ="";  
txtPaís.Text ="";  
txtTelefone.Text ="";  
txtFax.Text ="";
```

```
}
```

```
</script>
```

```
<body>
```

```
<asp:Label  
    id="Label1"  
    runat="server"  
/>
```

```
<HR>
```

```
<asp:Label  
    id="Label2"  
    runat="server"  
/>
```

```
<h3><font face="Verdana">  
    CADASTRO DE CLIENTES  
<BR>  
    Empresa North Traders Ltda.  
</font>  
</h3>
```

<HR>

```
<form runat=server>

<asp:RequiredFieldValidator
    id="Requer_CódigoDoCliente"
    ControlToValidate="txtCódigoDoCliente"
    Type="String"
    ErrorMessage="O Código do Cliente é campo Obrigatório <BR>"
    Text="O Código do Cliente é campo Obrigatório."
    ForeColor="Red"
    runat="server"
/>

<asp:RequiredFieldValidator
    id="Requer_NomeDaEmpresa"
    ControlToValidate="txtNomeDaEmpresa"
    Type="String"
    ErrorMessage="O Nome da Empresa é campo Obrigatório."
    Text="O Nome da Empresa é campo Obrigatório."
    ForeColor="Red"
    runat="server"
/>

<asp:RequiredFieldValidator
    id="Requer_Endereço"
    ControlToValidate="txtEndereço"
    Type="String"
    ErrorMessage="O Endereço é campo Obrigatório."
    Text="O Endereço é campo Obrigatório."
    ForeColor="Red"
    runat="server"
/>

<asp:RequiredFieldValidator
    id="Requer_Telefone"
    ControlToValidate="txtTelefone"
```

```
Type="String"
ErrorMessage="O Telefone é campo Obrigatório."
Text="O Telefone é campo Obrigatório."
ForeColor="Red"
runat="server"
/>>

<asp:RequiredFieldValidator
    id="Requer_País"
    ControlToValidate="txtPaís"
    Type="String"
    ErrorMessage="O País é campo Obrigatório."
    Text="O País é campo Obrigatório."
    ForeColor="Red"
    runat="server"
/>

<div align="left">
<table border="0">

    <tr>
        <td>
            <p align="right">
                <B>Código do Cliente: (*)</B>
            </p>
            <td>
                <asp:TextBox
                    runat="server"
                    id="txtCódigoDoCliente"
                    Text=""
                    TextMode="SingleLine"
                    Font_Face="Arial"
                    Font_Size="3"
                    MaxLength="5"
                    Height="20"
                    Width="300"
                />
            
```

```
</td>

</tr>

<tr>
    <td>
        <p align="right">
            <B>Nome da Empresa: (*)</B>
        </td>
        <td>
            <asp:TextBox
                runat="server"
                id="txtNomeDaEmpresa"
                Text=""
                TextMode="SingleLine"
                Font_Face="Arial"
                Font_Size="3"
                MaxLength="40"
                Height="20"
                Width="250"
            />
        </td>
    </tr>
    <tr>
        <td>
            <p align="right">
                <B>Nome do Contato:</B>
            </td>
            <td>
                <asp:TextBox
                    runat="server"
                    id="txtNomeDoContato"
                    Text=""
                    TextMode="SingleLine"
                    Font_Face="Arial"
                    Font_Size="3"
                    MaxLength="30"
                />
            </td>
        </tr>
```

```
    Height="20"
    Width="250"
/>
</td>
</tr>
<tr>
<td>
<p align="right">
<B>Cargo do Contato:</B>
</td>
<td>
<asp:TextBox
    runat=server
    id="txtCargoDoContato"
    Text=""
    TextMode="SingleLine"
    Font_Face="Arial"
    Font_Size="3"
    MaxLength="30"
    Height="20"
    Width="250"
/>
</td>
</tr>
<tr>
<td>
<p align="right">
<B>Endereço: (*)</B>
</td>
<td>
<asp:TextBox
    runat=server
    id="txtEndereço"
    Text=""
    TextMode="SingleLine"
    Font_Face="Arial"
    Font_Size="3"
```

```
        MaxLength="60"

        Height="20"
        Width="200"

    />

</td>

</tr>

<tr>

    <td>

        <p align="right">
        <B>Cidade:</B>
        </td>

    <td>

        <asp:TextBox
            runat=server
            id="txtCidade"
            Text=""
            TextMode="SingleLine"
            Font_Face="Arial"
            Font_Size="3"
            MaxLength="15"
            Height="20"
            width="250"
        />
    </td>

</tr>

<tr>

    <td>

        <p align="right">
        <B>Região:</B>
        </td>

    <td>

        <asp:TextBox
            runat=server
            id="txtRegião"
            Text=""
            TextMode="SingleLine"
            Font_Face="Arial"

```

```
        Font_Size="3"
        MaxLength="15"
        Height="20"
        Width="250"
    />
</td>

</tr>
<tr>
<td>
<p align="right">
<B>CEP:</B>
</td>
<td>
<asp:TextBox
    runat=server
    id="txtCEP"
    Text=""
    TextMode="SingleLine"
    Font_Face="Arial"
    Font_Size="3"
    MaxLength="10"
    Height="20"
    Width="250"
    />
</td>
</tr>
<tr>
<td>
<p align="right">
<B>FAX:</B>
</td>
<td>
<asp:TextBox
    runat=server
    id="txtFax"
    Text=""
    TextMode="SingleLine"
```

```
Font_Face="Arial"
Font_Size="3"
MaxLength="24"

Height="20"
Width="250"

/>
</td>

</tr>
<tr>

<td>


Telefone: (*)


</td>
<td>
<asp:TextBox
    runat=server
    id="txtTelefone"
    Text=""
    TextMode="SingleLine"
    Font_Face="Arial"
    Font_Size="3"
    MaxLength="24"

    Height="20"
    Width="200"
/>
</td>
</tr>
<tr>

<td>


País: (*)


</td>
<td>
<asp:TextBox
    runat=server
    id="txtPaís"

```

```
        Text=""  
        TextMode="SingleLine"  
        Font_Face="Arial"  
        Font_Size="3"  
        MaxLength="15"  
  
        Height="20"  
        Width="200"  
    />  
  </td>  
  <td>  
    </td>  
  </tr>  
  <tr>  
    <td>  
      <p align="right">  
        <b>Clique no botão Cadastrar.</b>  
      </td>  
    <td>  
      <asp:Button  
        id="CadastraCliente"  
        Text="Cadastrar Cliente"  
        runat="server"  
        OnClick="InserirCliente"  
      />  
    </td>  
  </tr>  
  <tr>  
    <td> <B><p align="right"> *** </B> </td>  
    <td> <B> Campos de preenchimento obrigatório </B> </td>  
  </tr>  
  
</table>  
</div>  
</form>  
  
</body>  
</html>
```

Digite o código da Listagem 12.4 e salve o mesmo em um arquivo chamado chap12ex4.aspx, na pasta chap12, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço:

<http://localhost/chap12/chap12ex4.aspx>

Ao carregar a página você obtém o resultado indicado na Figura 12.8, onde é apresentado um formulário para que você digite as informações do cliente a ser Cadastrado.

The screenshot shows a Microsoft Internet Explorer window with the title bar 'http://localhost/Chap12/chap12ex4.aspx - Microsoft Internet Explorer'. The menu bar includes 'Arquivo', 'Editar', 'Exibir', 'Favoritos', 'Ferramentas', and 'Ajuda'. The toolbar includes standard buttons like Back, Forward, Stop, Home, Refresh, and Favorites. The address bar shows the URL 'http://localhost/Chap12/chap12ex4.aspx'. The main content area displays a form titled 'CADASTRO DE CLIENTES' for 'Empresa North Traders Ltda.'. The form fields are as follows:

- Código do Cliente: (*) [Text input field]
- Nome da Empresa: (*) [Text input field]
- Nome do Contato: [Text input field]
- Cargo do Contato: [Text input field]
- Endereço: (*) [Text input field]
- Cidade: [Text input field]
- Região: [Text input field]
- CEP: [Text input field]
- FAX: [Text input field]
- Telefone: (*) [Text input field]
- Pais: (*) [Text input field]

Below the fields, there is a button labeled 'Clique no botão Cadastrar' and a 'Cadastrar Cliente' button. A note at the bottom states '*** Campos de preenchimento obrigatório'.

Figura 12.8: Formulário para cadastro de Clientes.

Digite informações para um cliente fictício, porém deixe o campo Telefone, que é um campo obrigatório, em branco. Clique no botão Cadastrar Clientes. O cadastro do cliente não é feito e você é informado de que o campo Telefone é obrigatório, conforme indicado na Figura 12.9.

The screenshot shows a Microsoft Internet Explorer window displaying a web page titled "CADASTRO DE CLIENTES" for "Empresa North Traders Ltda.". The page contains several input fields and validation messages. At the top, there is a message: "O Telefone é campo Obrigatório." Below this, there are text input fields for "Código do Cliente" (containing "AABCD"), "Nome da Empresa" (containing "Maravilhas da Mata"), "Nome do Contato" (containing "José da Silva"), and "Cargo do Contato" (containing "Gerente"). There are also fields for "Endereço" (containing "Orlando Pereira - 805"), "Cidade" (containing "São Paulo"), "Região" (containing "Sudeste"), "CEP" (containing "99999-999"), and "FAX" (containing "011-11-777-8899"). Below these, there are fields for "Telefone" and "País" (both empty). At the bottom left is a button labeled "Clique no botão Cadastrar" and at the bottom right is a button labeled "Cadastrar Cliente". A note below the buttons says "*** Campos de preenchimento obrigatório". The status bar at the bottom shows "Concluído" and "Intranet local".

Figura 12.9: Os controles de validação em ação.

Digite um valor para o telefone e dê um clique no botão Cadastrar Cliente. Agora sim, o cliente é cadastrado e o formulário é reapresentado, para que você possa cadastrar outro cliente, conforme indicado na Figura 12.10.

Comentários sobre o código do exemplo – Chap12ex4.aspx.

- ◆ Para alinhar os controles, na seção de apresentação, utilizamos uma tabela com duas colunas. Na primeira coluna colocamos um texto descritivo do campo e, na segunda coluna, um controle TextBox para que o usuário digite informações.
- ◆ Utilizamos controles de validação do tipo RequiredFieldValidator, para garantir que os seguintes campos sejam preenchidos:
 - ◆ CódigoDoCliente
 - ◆ NomeDaEmpresa
 - ◆ Endereço
 - ◆ Telefone
 - ◆ País

The screenshot shows a Microsoft Internet Explorer window with the URL <http://localhost/Chap12/chap12ex4.aspx>. The page displays a success message: "Registro para o cliente: Maravilhas da Mata Inserido com sucesso". Below it, there is a form titled "CADAstro DE CLIENTES" for "Empresa North Traders Ltda.". The form contains fields for Client ID, Company Name, Contact Name, Contact Position, Address, City, Region, Zip Code, Fax, Phone, and Country. At the bottom, there is a button labeled "Cadastrar Cliente". The status bar at the bottom of the browser window shows "Concluído" and "Intranet local".

Figura 12.10: Cliente cadastrado com sucesso.

NOTA: Na prática, todos os campos da tabela Clientes são configurados para não aceitar valores nulos. Se você, por exemplo, não digitar um valor para o campo Cidade (que não possui um campo de validação associado), na hora de salvar o registro será gerado o erro indicado na Figura 12.11. Isto acontece porque o campo Cidade, na tabela Clientes, do Banco de dados NorthWind.mdb, não aceita valores nulos. Para solucionar esta opção, basta adicionar controles de validação do tipo – RequiredFieldValidator, para todos os controles do formulário. Desta forma, as informações somente serão enviadas para o banco de dados, quando todos os controles estiverem preenchidos.

Observe o trecho da mensagem de erro, transcrita a seguir:

- ◆ **Exception Details:** System.Data.OleDb.OleDbException: O campo ‘Clientes.Cidade’ não pode ser uma seqüência de caracteres de comprimento nulo.
- ◆ Para inserir o registro do cliente, na tabela Clientes, do Banco de dados NorthWind.mdb, utilizamos o evento OnClick do botão CadastraCliente. Criamos o procedimento InserirCliente, que é executado em resposta ao evento OnClick do botão de comando.

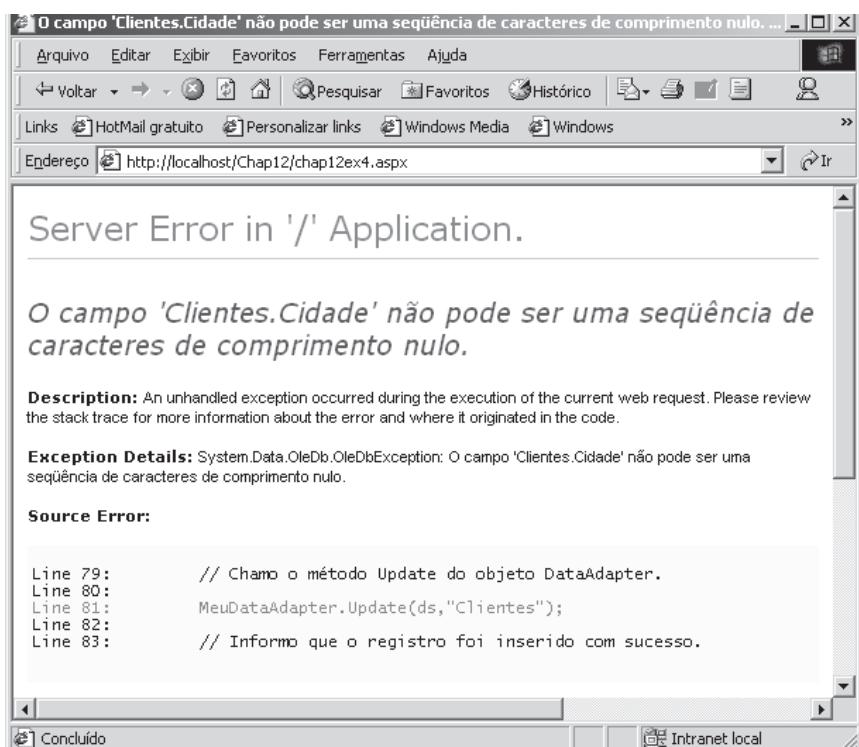


Figura 12.11: Erro ao deixar um campo Requerido em branco.

O evento InserirCliente faz o seguinte:

1. Conecta com o banco de dados utilizando os comandos já vistos em exemplos anteriores.
2. Cria um objeto DataTable – Clientes, e define a chave primária:

```

DataTable Clientes = ds.Tables[0];
Clientes.PrimaryKey = new DataColumn[] {Clientes.Columns["CódigoDoCliente"]};

```

3. Cria uma nova linha, define o valor dos campos desta linha e adiciona a linha à coleção de linhas da tabela Clientes:

```
DataRow Linha = Clientes.NewRow();
```

```
// Defino os valores para a linha a ser inserida.
```

```

Linha["CódigoDoCliente"]      = txtCódigoDoCliente.Text;
Linha["NomeDaEmpresa"]        = txtNomeDaEmpresa.Text;
Linha["NomeDoContato"]        = txtNomeDoContato.Text;
Linha["CargoDoContato"]       = txtCargoDoContato.Text;
Linha["Endereço"]             = txtEndereço.Text;
Linha["Cidade"]               = txtCidade.Text;
Linha["Região"]               = txtRegião.Text;
Linha["CEP"]                  = txtCEP.Text;
Linha["País"]                 = txtPaís.Text;

```

```

Linha["Telefone"]      = txtTelefone.Text;
Linha["Fax"]           = txtFax.Text;

Clientes.Rows.Add(Linha);

```

4. Crio um objeto OleDbCommandBuilder, utilizo o método GetInsertCommand deste objeto para definir a propriedade InsertCommand do objeto DataAdapter e chamo o método Update do objeto DataAdapter:

```

OleDbCommandBuilder CriaComando = new OleDbCommandBuilder(MeuDataAdapter);

MeuDataAdapter.InsertCommand= CriaComando.GetInsertCommand();

MeuDataAdapter.Update(ds,"Clientes")

```

5. Exibo informações de que o cliente foi cadastrado com sucesso e “limpo” os valores contidos nos controles TextBox, para que o usuário possa digitar informações de outro Cliente.

```

Label1.Text= "Registro para o cliente: " + txtNomeDaEmpresa.Text + " Inserido com sucesso";

Label2.Text="Preencha os campos abaixo para cadastrar outro Cliente";

// Limpo o valor dos campos TextBox.

```

```

txtCódigoDoCliente.Text     = "";
txtNomeDaEmpresa.Text       = "";
txtNomeDoContato.Text       = "";
txtCargoDoContato.Text      = "";
txtEndereço.Text            = "";
txtCidade.Text              = "";
txtRegião.Text              = "";
txtCEP.Text                 = "";
txtPaís.Text                = "";
txtTelefone.Text             = "";
txtFax.Text                 = "";

```

- ◆ Ok. Um novo Cliente foi cadastrado no Banco de dados NorthWind.mdb. A seguir veremos como definir as operações de Exclusão e Edição de registros. Para isso utilizaremos as propriedades DeleteCommand e UpdateCommand, do objeto DataAdapter.

Excluindo Registros – a Propriedade DeleteCommand

Para enviar as exclusões feitas em um objeto DataTable, para o banco de dados, utilizamos os seguintes passos:

1. Definimos uma chave primária para o objeto DataTable. Considere o exemplo:

```
Clientes.PrimaryKey = new DataColumn[] {Clientes.Columns["CódigoDoCliente"]};
```

2. Localizo a linha a ser excluída. Para localizar a linha posso utilizar o método Find, da coleção Rows, descrito e exemplificado anteriormente. Também posso utilizar, caso eu conheça, o índice da linha a ser excluído. Vejamos alguns exemplos:

```
// Excluo a primeira linha do DataTable Clientes, utilizando o índice da linha.  
Clientes.Rows[0].Delete();  
  
// Outra alternativa é localizar a linha a ser excluída e depois chamar o  
método Delete da linha.  
  
DataRow Linha = Clientes.Rows.Find(MinhaLista.SelectedItem.Value);  
Linha.Delete();
```

3. Agora precisamos enviar as exclusões para o banco de dados. Em primeiro lugar criamos um objeto do tipo OleDbCommandBuilder, associado ao objeto DataAdapter que estamos utilizando:

```
OleDbCommandBuilder CriaComando = new OleDbCommandBuilder(MeuDataAdapter);
```

Defino a propriedade DeleteCommand, do objeto DataAdapter. Posso definir esta propriedade manualmente ou utilizando o método GetDeleteCommand do objeto OleDbCommandBuilder. No exemplo a seguir, utilizamos o método GetDeleteCommand:

```
MeuDataAdapter.DeleteCommand= CriaComando.GetDeleteCommand();
```

Definidos os comandos necessários é hora de fazer com que eles sejam efetivamente executados no banco de dados. Para tal chamamos o método Update do objeto DataAdapter:

```
MeuDataAdapter.Update(ds, "Clientes");
```

Como parâmetros para este método, passamos o nome do DataSet e o nome da tabela a ser atualizada. O método Update irá executar, no banco de dados, os comandos definidos na propriedade DeleteCommand. O resultado prático desta operação é que todos os registros que foram excluídos do objeto DataTable serão excluídos da tabela no banco de dados, ou seja, estamos sincronizando as alterações feitas na cópia “desconectada” dos dados, com a cópia original dos dados. Após a execução do método Update, os dois conjuntos de dados – desconectados e originais – são cópias idênticas.

Para que possamos gerar os comandos automaticamente, utilizando um objeto do tipo CommandBuilder, algumas condições devem ser observadas:

- ◆ Os dados que formam uma tabela no objeto DataSet devem ter sido obtidos a partir de uma única tabela no banco de dados. Por exemplo, se utilizamos as tabelas Pedidos e Detalhes do pedido, para obter uma listagem com o total por Pedido, precisaremos construir manualmente os comandos de atualização, edição e exclusão. Construir manualmente significa criar uma string com o comando SQL e atribuir esta String à propriedade respectiva.
- ◆ A tabela deve ter uma Chave Primária. Pode ser uma Chave Primária simples (formada por uma coluna) ou uma Chave Primária composta por duas ou mais colunas.
- ◆ Os nomes de tabelas não podem conter caracteres especiais como espaços, pontos, aspas e outros caracteres que não sejam números e letras.

Adicionando Registros – a Propriedade InsertCommand

Para enviar as adições de registros, feitas em um objeto DataTable, para o banco de dados, utilizamos os seguintes passos:

1. Definimos uma chave primária para o objeto DataTable. Considere o exemplo:

```
Clientes.PrimaryKey = new DataColumn[] { Clientes.Columns["CódigoDoCliente"]};
```

2. Crio uma nova linha utilizando o método NewRow, defino os valores para cada campo da linha e utilizo o método Add, da coleção Rows, do objeto DataTable:

```
// Chamo o método NewRow da tabela Clientes.
```

```
DataRow Linha = Clientes.NewRow();
```

```
// Defino os valores para a linha a ser inserida.
```

```
Linha["CódigoDoCliente"] = txtCódigoDoCliente.Text;
Linha["NomeDaEmpresa"] = txtNomeDaEmpresa.Text;
Linha["NomeDoContato"] = txtNomeDoContato.Text;
Linha["CargoDoContato"] = txtCargoDoContato.Text;
Linha["Endereço"] = txtEndereço.Text;
Linha["Cidade"] = txtCidade.Text;
Linha["Região"] = txtRegião.Text;
Linha["CEP"] = txtCEP.Text;
Linha["País"] = txtPaís.Text;
Linha["Telefone"] = txtTelefone.Text;
Linha["Fax"] = txtFax.Text;
```

```
Clientes.Rows.Add(Linha);
```

3. A linha foi adicionada ao conjunto de dados desconectados; agora precisamos enviar a adição para o banco de dados. Em primeiro lugar criamos um objeto do tipo OleDbCommandBuilder, associado ao objeto DataAdapter que estamos utilizando:

```
OleDbCommandBuilder CriaComando = new OleDbCommandBuilder(MeuDataAdapter);
```

Defino a propriedade InsertCommand, do objeto DataAdapter. Posso definir esta propriedade manualmente ou utilizando o método GetInsertCommand do objeto OleDbCommandBuilder. No exemplo a seguir, utilizamos o método GetInsertCommand:

```
MeuDataAdapter.InsertCommand= CriaComando.GetInsertCommand();
```

Definidos os comandos necessários, é hora de fazer com que estes comandos sejam efetivamente executados no banco de dados. Para tal chamamos o método Update do objeto DataAdapter:

```
MeuDataAdapter.Update(ds, "Clientes");
```

Como parâmetros para este método, passamos o nome do DataSet e o nome da tabela a ser atualizada. O método Update irá executar, no banco de dados, os comandos definidos na propriedade InsertCommand. O resultado prático desta operação é que a nova linha será adicionada à tabela do banco de dados original, ou seja, estamos sincronizando as alterações feitas na cópia “desconectada” dos dados, com a cópia original dos dados. Após a execução do método Update, os dois conjuntos de dados – desconectados e originais – são cópias idênticas.

Conclusão

Neste capítulo aprendemos técnicas fundamentais para o trabalho com bancos de dados.

Inicialmente aprendemos a realizar edições, inclusões e exclusões na cópia de dados desconectados, ou seja, nos dados de um objeto DataTable. Aprendemos a utilizar diversas propriedades e métodos do objeto DataTable.

Porém as alterações precisam ser sincronizadas com o banco de dados originais. Na parte final do capítulo aprendemos a enviar as alterações feitas no objeto DataTable para o banco de dados original. Com isso mantemos sincronizadas as duas cópias dos dados.

No próximo capítulo falaremos sobre Web Services e sobre o novo ambiente de desenvolvimento do Framework .NET: Visual Studio .NET.

P A R T E

3

Conceitos Avançados
do ASP.NET e Segurança

Introdução

Na década de 70 tínhamos os dados e a lógica de programação instalados no Mainframe. Acessávamos estas aplicações utilizando terminais para conectar com o Mainframe. Como uma evolução surgiu o modelo Cliente/Servidor, onde tradicionalmente temos o banco de dados rodando no Servidor e a lógica e apresentação da aplicação, instaladas na estação do cliente. Este modelo mostrou-se de difícil manutenção e atualização e surgiu o modelo em três camadas, onde a lógica fica armazenada no servidor de aplicações, normalmente no formato de componentes COM/COM+, os dados no servidor de banco de dados e a apresentação no programa instalado no Cliente. Com o crescimento explosivo da Internet e, consequentemente, com a consolidação dos protocolos Web, passamos a utilizar o Navegador como cliente e o modelo de desenvolvimento baseado em padrões Web é uma realidade.

Mas a Tecnologia de Informação está sempre evoluindo, buscando mais eficiência e melhores resultados a custos mais razoáveis. Dentro deste contexto surge a idéia da construção e utilização de software como se fossem serviços. Por exemplo, ao invés de pagar um valor X, por uma licença do Office e receber um CD para instalação, você paga, simplesmente, um valor mensal de assinatura e acessa o Office através da Internet. Utilizando o Navegador você tem acesso ao Word, Excel, etc., podendo optar por salvar seus arquivos na máquina local ou no servidor do prestador de serviços. A idéia do aluguel de software está movimentando o mercado. As empresas que fornecem este tipo de serviço são os chamados Applications Services Providers – ASP.

Para a criação de programas que possam ser acessados como serviços, precisamos de alguma maneira poder criar pequenas unidades de software, com funcionalidades específicas e depois juntar estas “pequenas unidades”, para formar nossos programas. Em determinadas situações pode ser vantajoso criar a “funcionalidade” internamente; em outras situações pode ser mais interessante simplesmente pagar para ter acesso a um componente que já fornece a funcionalidade desejada. Por exemplo, se você cria um site de Comércio Eletrônico, pode ser mais vantajoso permitir que o seu sistema utilize um componente de validação de cartão de crédito, disponível no servidor da empresa que criou o componente, mediante um pagamento por acesso ou por mês, do que criar um componente a partir do zero.

Com os Web Services do Framework .NET, podemos criar componentes que tenham as funcionalidades descritas no parágrafo anterior, ou seja, um componente com uma funcionalidade específica, que pode ser acessado por qualquer aplicação, através da Web. Neste componente posso definir permissões de acesso, de tal maneira que somente possam utilizá-lo os usuários que estão pagando pelo serviço.

CAPÍTULO 13

Web Services e Visual Studio .NET

Com isso podemos criar aplicações realmente distribuídas, onde as diversas funcionalidades do sistema estão localizadas em diferentes servidores, inclusive de diferentes empresas. Neste capítulo veremos como criar Web Services e depois utilizar os Web Services criados em nossas páginas ASP.NET.

Em seguida aprenderemos a utilizar o ambiente de desenvolvimento do Framework .NET – Visual Studio .NET. Aprenderemos a criar páginas ASP.NET simples e também exemplos mais elaborados, que fazem conexão com bancos de dados. Veremos que o Visual Studio .NET traz, para o desenvolvimento Web, as mesmas facilidades que temos no desenvolvimento de aplicações Windows tradicionais. Com o Visual Studio .NET construímos aplicações Web, baseadas em ASP.NET, arrastando componentes na página e configurando as diversas opções, propriedades e eventos dos componentes.

Com o Visual Studio .NET a produtividade do programador, na criação de aplicações Web, aumenta enormemente, uma vez que o Visual Studio .NET traz para o desenvolvimento de páginas ASP.NET um modelo já conhecido, onde simplesmente vamos arrastando elementos na página e fazendo as configurações necessárias.

Uma Introdução aos Web Services

O fator que mais provocou mudanças nas metodologias e tecnologias de desenvolvimento de aplicações, nos últimos tempos, foi Internet. A criação de aplicações distribuídas e o compartilhamento de informações passaram a ser mais uma necessidade do que uma realidade. Como fazer o sistema de uma empresa, baseado no Mainframe, trocar informações com o sistema de outra empresa, baseado no modelo Cliente/Servidor?

Muitas vezes, dentro da mesma empresa temos diferentes ambientes e a questão do compartilhamento de informações e integração de aplicações é uma das mais difíceis de serem resolvidas. Com a Internet iniciou-se um processo de padronização em diversas frentes:

- ◆ TCP/IP como protocolo da Internet e também das redes internas, das empresas.
- ◆ HTML como um formato padrão para a publicação de informações.
- ◆ HTTP como protocolo de transporte.

Porém alguns problemas ainda persistiam. Como viabilizar a troca de informações entre empresas que, na maioria das vezes, utilizam formatos de dados proprietários e completamente incompatíveis? Como resposta a esta questão, o padrão XML (Extensible Markup Language) tem sido uma resposta eficaz e amplamente aceita pelo mercado. Porém ainda persiste a questão de como fazer que uma aplicação possa utilizar funcionalidades de outras aplicações, sem ter que reescrever toda a lógica que já está implementada.

Acredito que a utilização dos Web Services seja a resposta para esta última questão. Com Web Services podemos construir aplicações e componentes de software capazes de interagirem. Os Web Services podem ser criados em diferentes linguagens ou plataformas. O que garante a interoperabilidade é a utilização de um formato padrão para troca de mensagens (XML) e um protocolo padrão para o envio e recebimento destas mensagens (SOAP – Simple Object Access Protocol); sendo que as mensagens no formato SOAP são empacotadas e transportadas utilizando-se HTTP ou SMTP, protocolos amplamente utilizados na Internet.

O fato de utilizar padrões amplamente aceitos é que torna os Web Services uma alternativa atraente. Cabe ressaltar que o conceito de Web Services não é uma novidade do Framework .NET. Outras empresas como IBM e Sun já trabalham com o conceito de Web Services, sendo que a IBM também utiliza os padrões XML e SOAP.

Um Web Service é um componente ou unidade de software (eu prefiro o termo “Pedaço de Código”, mas reconheço que não é um termo muito, digamos, elegante) que fornece uma funcionalidade específica, como por exemplo uma rotina para validação do número de Cartão de Crédito ou do Dígito Verificador de um número de CPF; unidade esta que pode ser acessada por diferentes sistemas, através da utilização de padrões da Internet, como por exemplo XML, HTTP e SOAP. A utilização destes padrões é de vital importância para que possamos criar aplicações distribuídas utilizando Web Services.

Um Web Service pode ser utilizado internamente, por uma única aplicação ou por várias aplicações da mesma empresa; ou pode ser “exposto” através da Internet, para utilização por qualquer aplicação. Por exemplo, imagine que o Governo do Estado disponibiliza um Web Service para cálculo do ICMS. Todas as máquinas registradoras de Supermercados, Padarias e demais estabelecimentos possuem um programa que utiliza este Web Service. Se a legislação do ICMS mudar, tudo o que o governo precisaria fazer seria alterar o Web Service e as máquinas registradoras já passariam a fazer o cálculo do ICMS, baseado na nova versão. Pelo exemplo podemos ver que a utilização de Web Services é baseada em um “mundo” amplamente conectado, através de padrões utilizados na Internet. Como a forma de acesso a um Web Service é padronizada, isto permite que diferentes sistemas possam acessar e trocar dados com um mesmo Web Service. Para que uma aplicação possa fazer uso de um Web Service, basta que ela seja capaz de entender SOAP e XML.

A Microsoft vem trabalhando com diversas companhias e junto ao W3C (www.w3.org), que é uma entidade responsável pela padronização de diversas linguagens e protocolos de Internet, em uma proposta para a padronização do SOAP e de outros padrões que dão suporte à utilização de Web Services.

Para que um Web Service possa ser utilizado, algumas funções precisam estar disponíveis:

- ◆ **Descoberta:** Precisa existir uma maneira de “descobrirmos” a existência do Web Service. Por exemplo, quando você está criando uma aplicação de Comércio Eletrônico, você precisa de mecanismos para localizar Web Services com funcionalidades específicas e que possam ser utilizados pela aplicação que está sendo criada. Para resolver esta questão existem duas especificações sendo atualmente analisadas: 1) UDDI (Universal Description, Discovery and Integration – www.UDDI.org); 2) DISCO (Discovery).
- ◆ **Descrição das funcionalidades e métodos disponibilizados pelo Web Service:** Para documentar e expor as funcionalidades de um Web Service, a Microsoft propôs ao W3C a especificação WSDL (Web Service Description Language). O WSDL define regras para a descrição das funcionalidades de um Web Service, utilizando XML. São descritos os métodos e propriedades suportados pelo Web Service, os tipos de dados e os protocolos que podem ser utilizados para o envio e recebimento de mensagens. Maiores detalhes em www.w3.org/TR/wsdl.
- ◆ **Protocolo padrão para troca de mensagens:** Conforme descrito anteriormente, é utilizado o protocolo SOAP. Maiores informações em www.w3.org/TR/SOAP.

Possíveis Utilizações Para um Web Service

Vamos ver alguns exemplos onde poderíamos utilizar Web Services como uma forma de criar aplicações mais flexíveis e de fácil manutenção.

- ◆ **Criação de uma funcionalidade específica para ser utilizada por um programa cliente:** Esta é uma das situações mais simples possíveis. Criamos um Web Service, cuja funcionalidade será utilizada por um outro programa. Por exemplo, podemos criar um Web Service que contém informações sobre as taxas de imposto de todos os estados Brasileiros e a forma de cálculo dos impostos. Este Web Service pode ser utilizado por sites de comércio eletrônico para calcular o preço final do produto, de acordo com o local de entrega.

Neste cenário, o sistema do site de comércio eletrônico formata uma mensagem com as informações do produto e do local de destino, na forma de uma mensagem XML, a qual é encapsulada no formato SOAP para ser transportada pelo protocolo HTTP. No destino o Web Service “desempacota a mensagem”, lê as informações no formato XML, faz os cálculos com base nos valores recebidos, empacota o resultado no formato XML, o qual é encapsulado no formato SOAP para ser transportado pelo protocolo HTTP, de volta para o programa que fez a solicitação de cálculo.

- ◆ **Integração entre diferentes aplicações:** Este é um dos grandes problemas a serem resolvidos pela área de TI das empresas. Podemos utilizar Web Services para fazer a integração entre diferentes aplicações, criadas em diferentes plataformas e que trabalham com diferentes formatos de dados. Com Web Services, podemos expor a funcionalidade e os dados de cada aplicação como um Web Service. Em seguida criamos uma aplicação que utiliza o conjunto de Web Services, criados a partir das aplicações individuais, aplicação esta que habilita a interoperabilidade entre as diversas aplicações. Esta situação é ilustrada na Figura 13.1.

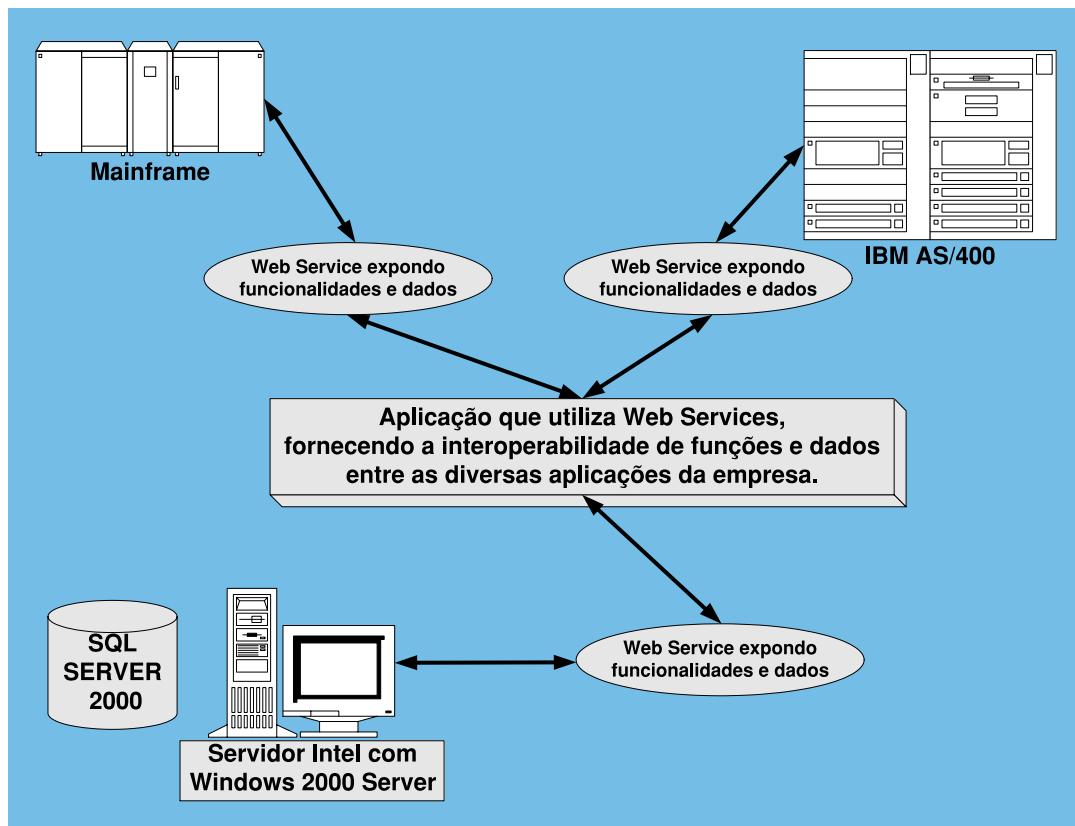


Figura 13.1: Interoperabilidade entre aplicações com Web Services.

- ◆ A utilização de Web Services também é de grande utilidade em aplicações de Workflow, onde um determinado processo ou tarefa é realizado em diferentes seções da empresa. O exemplo clássico é o caso de aprovação das despesas de viagens para um funcionário. O funcionário envia uma solicitação com a justificativa para o chefe imediato, o qual aprova a solicitação e envia para o setor de Recursos Humanos para as providências necessárias. Uma vez alocados os recursos necessários, o setor de Recursos Humanos informa o funcionário sobre horários, datas, etc. Para automatizarmos um Workflow como este do exemplo, precisamos interagir com dados e sistemas de diferentes departamentos da empresa, os quais normalmente não estão integrados, com formatos de dados diferentes. Neste ponto é que podemos utilizar Web Services para criar um sistema de Workflow que, do ponto de vista do usuário, funciona como um sistema integrado, que utiliza um único modelo de dados e conjunto de funcionalidades.

A Microsoft fornece um produto que facilita a criação de um sistema como o descrito no parágrafo anterior: BizTalk Server. Com este produto podemos criar processos de negócios e automatizar a geração de mensagens no formato XML, para interoperabilidade entre os sistemas.

O que Diferencia Web Services das Tecnologias de Componentes Como COM ou CORBA?

Caso você conheça a tecnologia COM /COM+ da Microsoft ou CORBA, coordenada por um grupo de grandes empresas como Sun, Netscape, IBM, etc., pode estar se perguntando se Web Services não é exatamente a mesma coisa.

Em termos de funcionalidade devo concordar que a idéia é bastante semelhante, mas o que diferencia um Web Service é o fato de este poder ser acessado a partir de qualquer servidor da Internet, utilizando um protocolo padrão como o HTTP. Já a tecnologia COM utiliza interfaces e formatos de mensagens, proprietários. Surgiram algumas soluções para acesso a componentes COM através da Internet, porém soluções proprietárias. Os mesmos comentários são válidos para a tecnologia CORBA.

O modelo de programação para a criação de Web Services é semelhante, em muitos aspectos, ao modelo de criação e utilização de componentes COM. O principal objetivo da tecnologia COM é possibilitar aos programadores a criação de uma aplicação a partir de componentes prontos, componentes estes que fornecem funcionalidades específicas. Para atingir este objetivo, COM utiliza uma padrão binário (e proprietário), para a definição das interfaces expostas por um componente COM. Embora o modelo COM facilite a localização e utilização de componentes, o modelo está restrito àquelas plataformas capazes de entender o método proprietário de comunicação, definido na especificação COM.

O principal objetivo dos Web Services é possibilitar aos desenvolvedores uma maneira fácil para integrar aplicações e dados de diferentes plataformas. Com Web Services somos capazes de integrar aplicações criadas em diferentes plataformas e em diferentes linguagens. Para tal, diferentemente do COM, os Web Services são baseados em padrões abertos (XML, SOAP, HTTP, etc.), padrões estes amplamente utilizados atualmente. Ao invés de um método binário para comunicação entre aplicações, os Web Services utilizam um mecanismo de comunicação baseado em XML. O XML passa a ser uma espécie de “Linguagem Universal para troca de dados e mensagens entre aplicações”.

Bem, chega de teoria. Agora vamos aprender a criar e a utilizar Web Services. Primeiro aprenderemos a criar um Web Service para, em seguida, aprender a utilizar Web Services em uma página ASP.NET.

Criando um Web Service

Vamos criar um Web Service que tem um único método: Calcula_Imposto. Este método recebe três parâmetros:

- ◆ O valor total da compra.
- ◆ O valor do desconto.
- ◆ O estado de destino.

Com base no estado de destino, o método calcula o valor do imposto sobre o preço final, já aplicado o desconto.

Por simplicidade vamos considerar pedidos apenas para cinco estados e um valor padrão para os demais estados. O percentual de imposto para cada estado está descrito na Tabela 13.1.

Tabela 13.1 Imposto a ser aplicado para cada estado.

Estado	% Imposto
RS	12
SC	15
PR	17
SP	20
RJ	22
Outros	25

Sintaxe Para a Criação de um Web Service

O primeiro passo é criar o código para o Web Service, código este que é gravado em um arquivo com a extensão .asmx. Não existe nenhuma tradução para a extensão asmx. Para caracterizar o código como um Web Service, incluímos a diretiva, como primeira linha do código:

```
<%@ WebService Language="C#" class="NomeDaClasse" %>
```

No nosso exemplo, vamos chamar a classe de “CalculosLegais”. Para definir a classe CalculosLegais utilizamos o seguinte comando:

```
<%@ WebService Language="C#" class="CalculosLegais" %>
```

O próximo passo é fazer referência ao namespace System.Web.Services. Este namespace contém as classes que dão suporte à criação de Web Services:

```
using System.Web.Services;
```

Agora implementamos a classe CalculosLegais e os métodos da classe. Um detalhe importante é que, após o nome da classe, colocamos dois-pontos e a palavra WebService. Antes de cada método colocamos a palavra WebMethod entre colchetes. Vamos chamar o método que faz o cálculo dos impostos de: “CalculaImposto”. A estrutura básica para a definição do WebService CalculosLegais está indicada na Listagem 13.1.

Listagem 13.1 – Estrutura básica para a criação de um Web Service.

```
<%@ WebService Language="C#" class="CalculosLegais" %>

using System.Web.Services;

public class CalculosLegais : WebService
{
    [WebMethod]
    public double CalculaImposto(long Total,int Desconto, string Estado)
    {
        long ValorComDesconto;
        ValorComDesconto = Total * (1-(Desconto/100));
        switch (Estado)
        {
            case "RS":
                return ValorComDesconto*1.12;
            break;
            case "SC":
                return ValorComDesconto*1.15;
            break;
            case "PR":
                return ValorComDesconto*1.17;
            break;
            case "SP":
                return ValorComDesconto*1.2;
            break;

            case "RJ":
                return ValorComDesconto*1.22;
            break;

            default:
                return ValorComDesconto*1.25;
            break;
        }
    }
}
```

Digite o código da Listagem 13.1 e salve o mesmo em um arquivo chamado CalculosLegais.asmx, na pasta chap13, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Uma Maneira Fácil de Testar a Funcionalidade de um Web Service

Com o Framework .NET podemos, facilmente, testar o funcionamento de um Web Service. Para isto basta acessar o arquivo .asmx, utilizando o navegador, como se fosse uma página .aspx normal. Para o nosso exemplo vamos utilizar o seguinte endereço: <http://localhost/chap13/CalculosLegais.asmx>

Ao acessarmos um arquivo .asmx, o Framework .NET utiliza um template chamado DefaultWsdlHelpGenerator.aspx, que fica localizado na pasta \WinNT\Microsoft.NET\Framework\[Version], onde WinNT é a pasta onde foi instalado o Windows 2000.

O template DefaultWsdlHelpGenerator.aspx identifica a requisição para um arquivo .asmx e gera, automaticamente, uma página de saída, onde temos a possibilidade de testar os métodos do Web Service que está sendo acessado, conforme indicado na Figura 13.2.

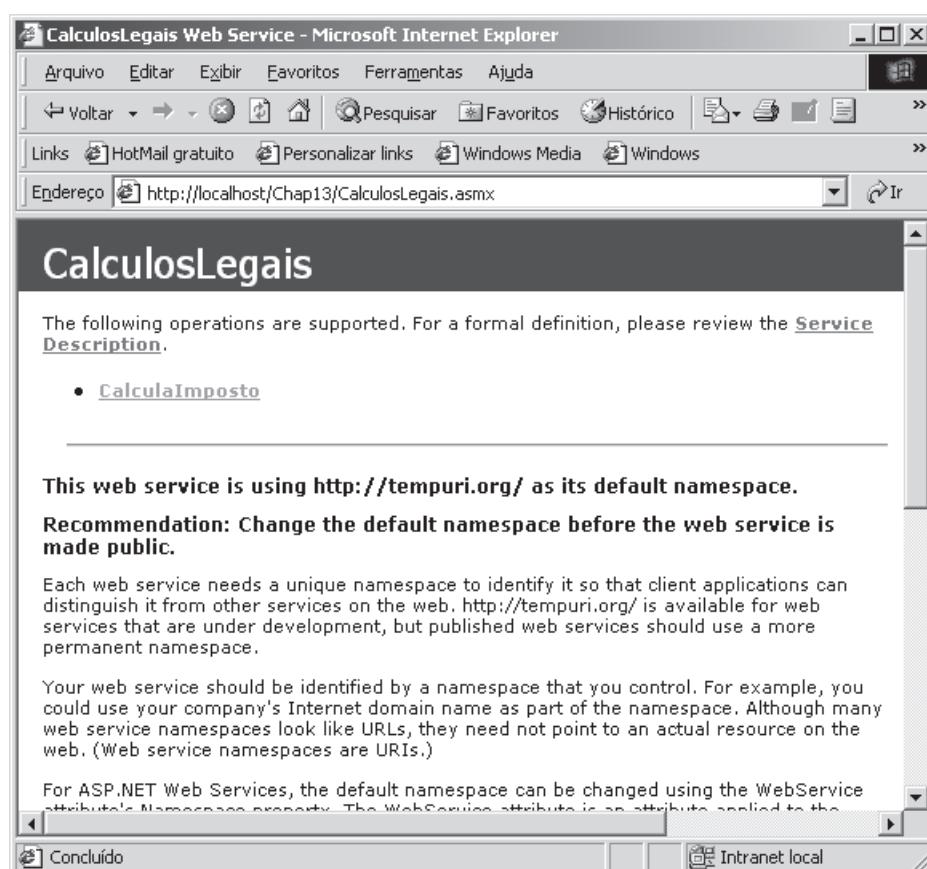


Figura 13.2: Página para teste do Web Service – gerada automaticamente.

Esta página traz informações genéricas sobre o Web Service que está sendo testado. Mas o principal componente desta página, para o nosso exemplo, é um link para o método CalculaImposto. Dê um clique neste link. Observe

que são exibidos campos para que digitemos os valores para os parâmetros do método. Digite os valores indicados na Figura 13.3.

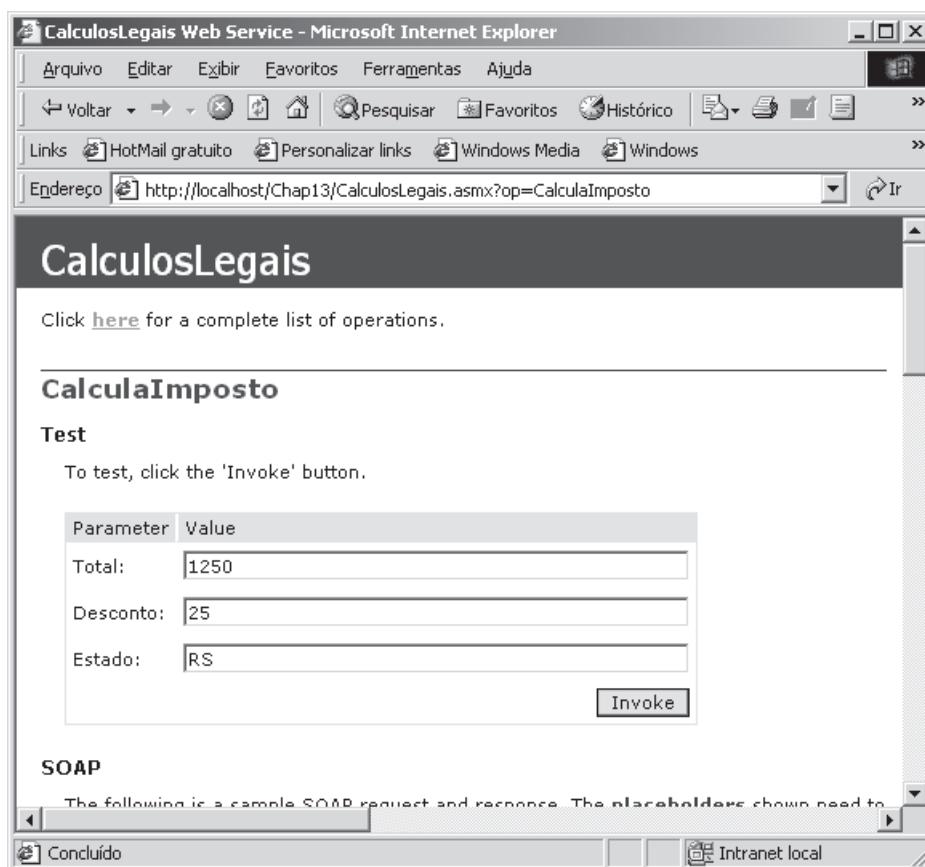


Figura 13.3: Definindo valores para os parâmetros do método CalculoImposto.

Dê um clique no botão Invoke. O método será executado e os resultados, retornados em uma nova janela, no formato XML, conforme indicado na Figura 13.4.

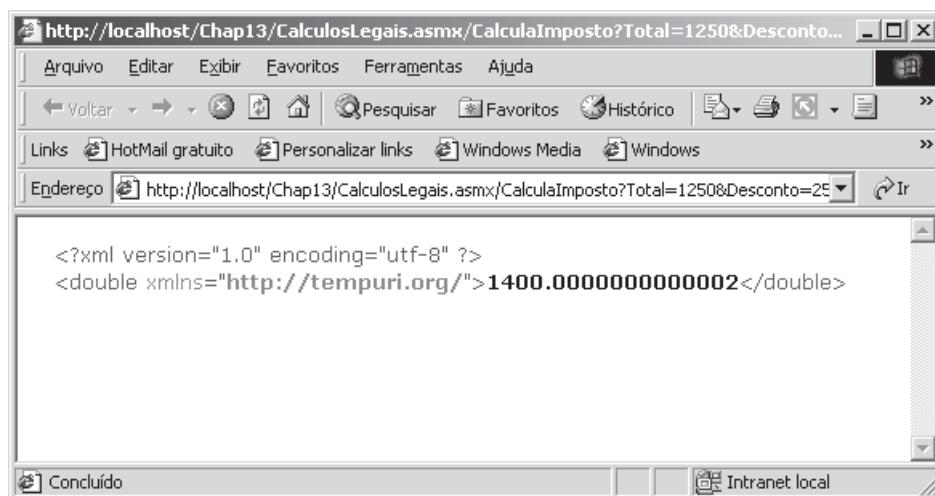


Figura 13.4: Valor retornado pelo método CalculoImposto, no formato XML.

Esta funcionalidade é bastante útil, pois nos permite testar os métodos de um Web Service antes de gerar um proxie (veremos com gerar proxies mais adiante) e utilizar o Web Service em nossas páginas ASP.NET (aprenderemos a utilizar Web Services em páginas ASP.NET mais adiante).

Na Página da Figura 13.3, onde podemos definir valores para os parâmetros do método CalculaImposto, temos uma série de informações sobre os formatos possíveis para a comunicação com um Web Service. Conforme visto nesta página, temos três opções possíveis:

- ◆ **SOAP:** É um protocolo baseado em XML, para troca de informações, mais precisamente: de mensagens, entre diferentes componentes de software, através da Web. A implementação do protocolo SOAP, no Framework .NET, utiliza o protocolo HTTP como protocolo de transporte. Com isso o SOAP segue o comportamento padrão do HTTP, ou seja, envia uma requisição para o servidor (que no caso do SOAP é um Web Service) e aguarda uma resposta.

Para maiores detalhes sobre o protocolo SOAP, consulte o endereço: www.w3.org/TR/SOAP

A seguir temos o exemplo de uma requisição e resposta utilizando o protocolo SOAP. Observe que os dados estão no formato XML:

```
POST /Chap13/CalculosLegais.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/CalculaImposto"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/
  soap/envelope/">
  <soap:Body>
    <CalculaImposto xmlns="http://tempuri.org/">
      <Total>long</Total>
      <Desconto>int</Desconto>
      <Estado>string</Estado>
    </CalculaImposto>
  </soap:Body>
</soap:Envelope>
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
```

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/
soap/envelope/">
<soap:Body>
<CalculaImpostoResponse xmlns="http://tempuri.org/">
<CalculaImpostoResult>double</CalculaImpostoResult>
</CalculaImpostoResponse>
</soap:Body>
</soap:Envelope>
```

- ◆ **HTTP GET:** Este é um dos métodos mais antigos utilizados para enviar informações através de uma requisição HTTP. Com este método, as informações são enviadas na própria URL, conforme pode ser visto pelo exemplo de requisição/resposta a seguir:

```
GET /Chap13/CalculosLegais.asmx/
```

```
CalculaImposto?Total=string&Desconto=string&Estado=string HTTP/1.1
```

```
Host: localhost
```

```
HTTP/1.1 200 OK
```

```
Content-Type: text/xml; charset=utf-8
```

```
Content-Length: length
```

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<double xmlns="http://tempuri.org/">double</double>
```

Observe que os dados são passados, na forma de string, no próprio endereço. Onde temos Total=string, devemos substituir string pelo valor realmente definido para o parâmetro, como por exemplo:

```
/Chap13/CalculosLegais.asmx/CalculaImposto?Total=1250&Desconto=25&Estado=RS
```

- ◆ **HTTP POST:** Este método é um pouco mais sofisticado do que o método GET. A principal diferença do método POST é que, com este método, as informações são enviadas na requisição HTTP e não na URL, como acontecia com o método GET. Observe o exemplo de requisição/resposta a seguir:

```
POST /Chap13/CalculosLegais.asmx/CalculaImposto HTTP/1.1
```

```
Host: localhost
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: length
```

```
Total=1250&Desconto=25&Estado=RS
```

```
HTTP/1.1 200 OK
```

```
Content-Type: text/xml; charset=utf-8
```

```
Content-Length: length
```

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<double xmlns="http://tempuri.org/">1400</double>
```

O nosso objetivo é criar um Web Service e poder utilizar as funcionalidades disponibilizadas por ele, em nossas páginas ASP.NET. Porém ainda temos um passo antes de que o Web Service CalculosLegais esteja disponível para uso. O passo que falta é a criação de um “proxy” para o Web Service. No próximo tópico veremos o que é um proxy e como criá-lo.

Proxies: Conceito e Criação

Conceito

Um proxy é um elemento intermediário entre a página ASP.NET e o Web Service. Vamos imaginar a situação onde temos uma página ASP.NET em um servidor www.abc.com, acessando um web service localizado em um segundo servidor: www.xyz.com. Com a utilização de um proxy, fazemos com que o Web Service, localizado no servidor www.xyz.com, pareça estar disponível localmente para a página ASP.NET que o está utilizando. O Proxy intercepta o pedido da página ASP.NET e envia o pedido para o Web Service no servidor remoto. Quando a resposta retorna, o Proxy captura a resposta e a encaminha para a página ASP.NET que fez a solicitação. O Proxy também é responsável por formatar o pedido no formato do protocolo SOAP, antes de o pedido ser enviado para o Web Service. Na Figura 13.5, temos uma pequena ilustração deste conceito:

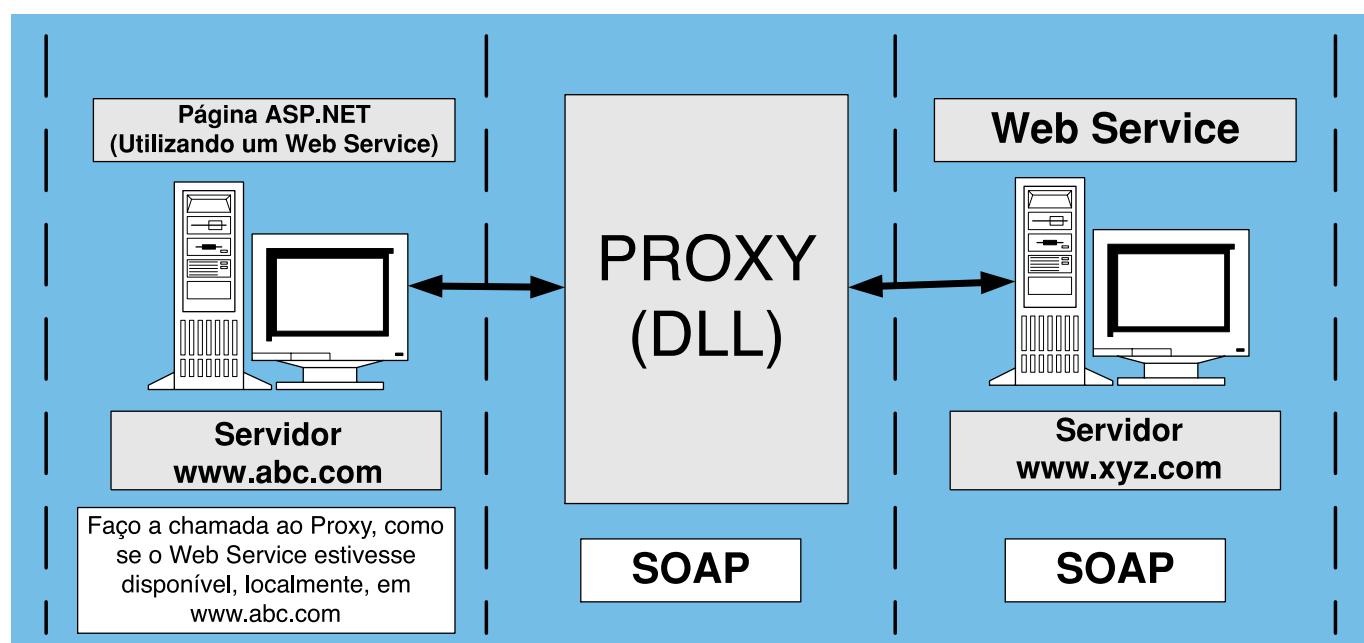


Figura 13.5: O papel do Proxy, no acesso ao Web Service.

Vamos recapitular os passos na criação do Web Service CalculosLegais:

1. Criação do código-fonte em um arquivo .asmx (já feito).
2. Geração de um proxy utilizando o utilitário Wsdl.exe (próximo tópico).
3. Compilação do código gerado no item 2, para a geração da DLL representativa do Web Service.

4. Distribuição da DLL criada no item anterior, para que ela possa ser acessada através da Web.
5. Utilização do Web Service, nas páginas ASP.NET onde a funcionalidade dele for necessária.

Criando o Proxy Utilizando o Utilitário Wsdl.exe

Para criarmos o Proxy, utilizamos um utilitário fornecido com o Framework .NET: Wsdl.exe. Este utilitário pode ser encontrado no seguinte caminho:

X:\Arquivos de programas\Microsoft .NET\FrameworkSDK\Bin

Onde X: é o drive onde está instalado o Framework .NET. Se você estiver utilizando o Windows 2000 em inglês, ao invés da pasta “Arquivos de Programas”, procure na pasta “Programs Files”.

A sintaxe para o utilitário Wsdl.exe é a seguinte:

```
Wsdl /language:language /protocol:protocol /namespace:myNameSpace /out:filename /
username:username /password:password /domain:domain <url or path>
```

Os parâmetros para o utilitário Wsdl.exe são explicados na Tabela 13.2.

Tabela 13.2 Parâmetros para Wsdl.exe.

Parâmetro	Descrição
/language:language	Opcional. Pode ser utilizado para definir uma das linguagens habilitadas ao .NET: CS para CSharp, VB para VB.NET e JS para JScript.NET. Se não for especificado, será utilizado CS, que corresponde ao C#.
/protocol:protocol	Opcional. Define o protocolo utilizado para invocar os métodos do Web Service. O padrão é SOAP. Também pode ser utilizado: HttpGet e HttpPost.
/namespace:myNameSpace	Opcional. Define o namespace do Proxy gerado.
/out:filename	Opcional. Define o nome do arquivo que será gerado, contendo o proxy. O nome padrão é baseado no nome do Web Service.
/username:username	Opcional. Nome do usuário com o qual fazer a conexão, quando o servidor, onde está o Web Service, requer autenticação.
/password:password	Opcional. Senha para o usuário definido no parâmetro anterior.
/domain:domain	Opcional. Nome do domínio ao qual pertence o usuário especificado no parâmetro /username:username.<url ou path>: Este é o único parâmetro obrigatório. Define uma URL ou um caminho para o arquivo que descreve o Web Service, arquivo este que deve estar no formato WSDL (Web Services Description Language). Se for um arquivo, especificamos o caminho para um arquivo com a extensão .wsdl. Se utilizarmos uma URL, a URL deve apontar para o arquivo .asmx ou para uma página que retorna uma descrição do arquivo, no formato WSDL.

Para Web Services utilizados com o ASP.NET, podemos retornar a descrição do Web Service, simplesmente concatenando “?WSDL” a URL que aponta para o arquivo .asmx. No nosso exemplo, o arquivo .asmx está no seguinte endereço: <http://localhost/Chap13/CalculosLegais.asmx>. Para retornar a descrição do mesmo, no formato WSDL, simplesmente vamos concatenar “?WSDL”, no final da URL, quando utilizarmos o comando Wsdl.exe, para gerar o proxy.

Vamos utilizar o comando Wsdl.exe para gerar o proxy para o Web Service CalculosLegais.asmx. Para isso, abra um Prompt de Comando (Iniciar -> Programas -> Acessórios -> Prompt de comando). Na janela que surge, digite o seguinte comando:

```
Wsdl http://localhost/Chap13/CalculosLegais.asmx?WSDL
```

e pressione ENTER. Será gerado um arquivo .cs (lembre que a linguagem padrão é o CSharp, para a geração de proxies), no mesmo diretório onde o comando foi executado. No nosso exemplo, indicado na Figura 13.6, foi gerado o arquivo CalculosLegais.cs, no drive D:

```
D:\>Wsdl http://localhost/Chap13/CalculosLegais.asmx?WSDL
Microsoft (R) Web Services Description Language Utility
[Microsoft (R) .NET Framework, Version 1.0.2914.16]
Copyright (C) Microsoft Corp. 1998-2001. All rights reserved.

Writing file 'D:\CalculosLegais.cs'.

D:\>_
```

Figura 13.6: Geração do proxy.

Na Listagem 13.2 temos o código do Arquivo CalculosLegais.cs. Este código é gerado automaticamente pelo utilitário wsdl.exe e não precisamos alterá-lo.

Listagem 13.2 – Código gerado pelo utilitário wsdl.exe.

```
//
// <autogenerated>
// This code was generated by a tool.
// Runtime Version: 1.0.2914.16
//
// Changes to this file may cause incorrect behavior and will be lost if
```

```
//      the code is regenerated.  
// </autogenerated>  
//  
//  
// This source code was auto-generated by wsdl, Version=1.0.2914.16.  
//  
using System.Diagnostics;  
using System.Xml.Serialization;  
using System;  
using System.Web.Services.Protocols;  
using System.Web.Services;  
  
[System.Web.Services.WebServiceBindingAttribute(Name="CalculosLegaisSoap",  
Namespace="http://tempuri.org/")]  
public class CalculosLegais : System.Web.Services.Protocols.SoapHttpClientProtocol  
  
{  
  
    [System.Diagnostics.DebuggerStepThroughAttribute()]  
    public CalculosLegais() {  
        this.Url = "http://localhost/Chap13/CalculosLegais.asmx";  
    }  
  
    [System.Diagnostics.DebuggerStepThroughAttribute()]  
    [System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://tempuri.org/  
CalculaImposto", Use=System.Web.Services.Description.SoapBindingUse.Literal,  
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]  
  
    public System.Double CalculaImposto(long Total, int Desconto, string Estado)  
    {  
        object[] results = this.Invoke("CalculaImposto", new object[]  
        {  
            Total,  
            Desconto,  
            Estado});  
        return ((System.Double)(results[0]));  
    }  
}
```

```

[System.Diagnostics.DebuggerStepThrough()]
public System.IAsyncResult BeginCalculaImposto(long Total, int Desconto, string
Estado, System.AsyncCallback callback, object asyncState) {
    return this.BeginInvoke("CalculaImposto", new object[]
{
    Total,
    Desconto,
    Estado}, callback, asyncState);
}

[System.Diagnostics.DebuggerStepThrough()]
public System.Double EndCalculaImposto(System.IAsyncResult asyncResult)
{
    object[] results = this.EndInvoke(asyncResult);
    return ((System.Double)(results[0]));
}
}

```

Compilando o Arquivo CalculosLegais.cs Para Gerar a DLL Correspondente

Agora precisamos compilar o arquivo com código-fonte – CalculosLegais.cs, para gerar a DLL correspondente. Para isso utilizaremos o compilador de linha de comando do CSharp: csc. Este compilador possui muitos parâmetros de linha de comando. Vamos explicar apenas os parâmetros utilizados para a criação da DLL.

Abra um Prompt de comando e execute o seguinte comando:

```
csc /out:CalculosLegais.dll /t:library /r:System.XML.dll /r:System.Web.Services.dll
CalculosLegais.cs
```

O arquivo CalculosLegais.cs é compilado e a DLL CalculosLegais.dll é gerada, conforme indicado na Figura 13.7.

Parâmetros utilizados para o compilador csc:

- ◆ **O parâmetro /t:library:** Indica que o código-fonte deve ser compilado e o resultado deve ser uma DLL. Outras opções seriam: /t:winexe, para gerar um programa executável do Windows ou /t:exe, para gerar um programa do tipo console.
- ◆ **/out:** Define o nome do arquivo gerado pela compilação.
- ◆ **CalculosLegais.cs:** O arquivo a ser compilado.
- ◆ **/r:** Faz referência aos metadados dos arquivos .dll especificados.

```
D:\>csc /out:CalculosLegais.dll /t:library /r:System.XML.dll /r:System.Web.Services.dll CalculosLegais.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

D:\>dir CalculosLegais.* 
0 volume na unidade D é W2K-SERVER
0 número de série do volume é 70BF-F202

Pasta de D:\

21/08/2001  21:42           2.243 CalculosLegais.cs
21/08/2001  22:29           4.608 CalculosLegais.dll
              2 arquivo(s)      6.851 bytes
              0 pasta(s)   2.518.970.368 bytes disponíveis

D:\>
```

Figura 13.7 Geração da DLL – CalculosLegais.dll.

Disponibilizando a DLL Para que a Mesma Possa Ser Utilizada

Para que a DLL, contendo o WebService CalculosLegais, possa ser utilizada por uma aplicação Web, devemos copiá-la para a pasta BIN da respectiva aplicação. Caso esta pasta ainda não exista, deverá ser criada e a DLL, copiada para a pasta BIN. No nosso exemplo, vamos criar uma página ASP.NET que utiliza o Web Service CalculosLegais. Neste caso, a página ASP.NET estará na pasta Chap13. Lembrando que, no servidor que estou utilizando, a pasta Chap13 encontra-se no seguinte caminho:

D:\InetPub\wwwroot\Chap13.

Vou criar a pasta bin, dentro da pasta Chap13 e depois copiar a DLL – CalculosLegais.dll para a pasta bin, conforme indicado na Figura 13.8:

NOTA: Para uma relação completa de todas as opções do compilador csc, digite: **csc /?** e pressione Enter.

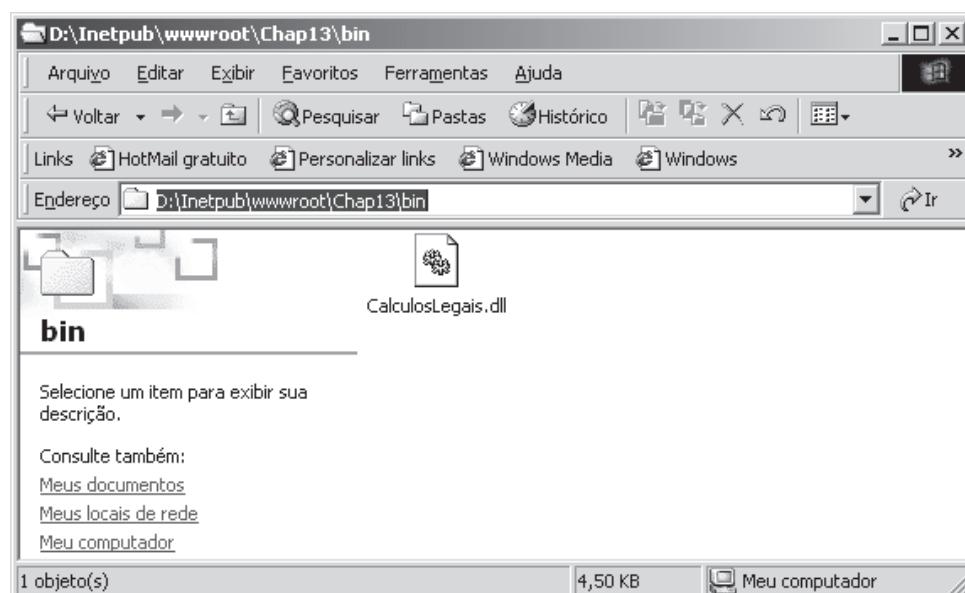


Figura 13.8: Disponibilizando a DLL – CalculosLegais.dll para uso.

IMPORTANTE: Além de disponibilizar a DLL, é importante que a pasta onde está a página ASP.NET que utilizará o Web Service seja configurada como um aplicativo. No nosso exemplo, criaremos uma página na pasta D:\inetPub\wwwroot\Chap13, porém esta pasta ainda não está configurada como um Aplicativo; Vamos configurar a pasta Chap13 como um aplicativo, para isso siga os seguintes passos:

1. Abra o gerenciador do IIS (Iniciar -> Programas -> Ferramentas Administrativas – Gerenciador de serviços de Internet).
2. Na janela que surge dê um clique no sinal de mais ao lado do nome do servidor que você está utilizando. No meu exemplo, o nome do equipamento é servidor.
3. Nas opções que surgem dê um clique no sinal de mais ao lado da opção “Site da Web padrão”, para expandi-la.
4. Nas pastas que surgem, clique com o botão direito na pasta Chap13 e, no menu de opções, clique em Propriedades.
5. Surge a janela “Propriedades de Chap13”. Na guia Pasta, temos as opções para tornar Chap13 uma aplicação Web.
6. Dê um clique no botão Criar. Certifique-se de que as opções da guia Pasta estejam configuradas conforme indicado na Figura 13.9.

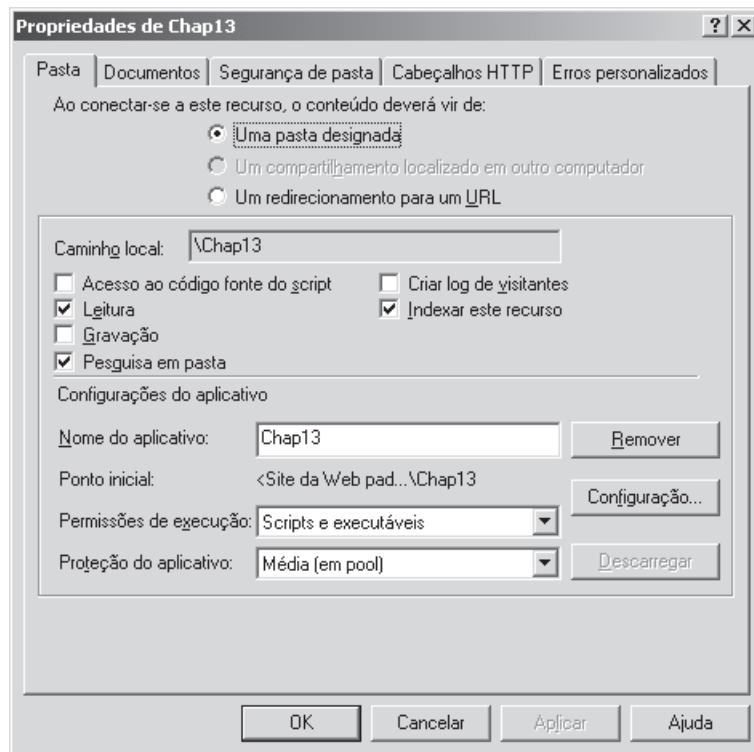


Figura 13.9: Tornando a pasta Chap13 uma aplicação Web.

7. De volta ao Gerenciador de serviços de Internet, observe que o ícone de Chap13 foi alterado. O novo ícone indica que Chap13 é uma aplicação Web, conforme indicado na Figura 13.10.

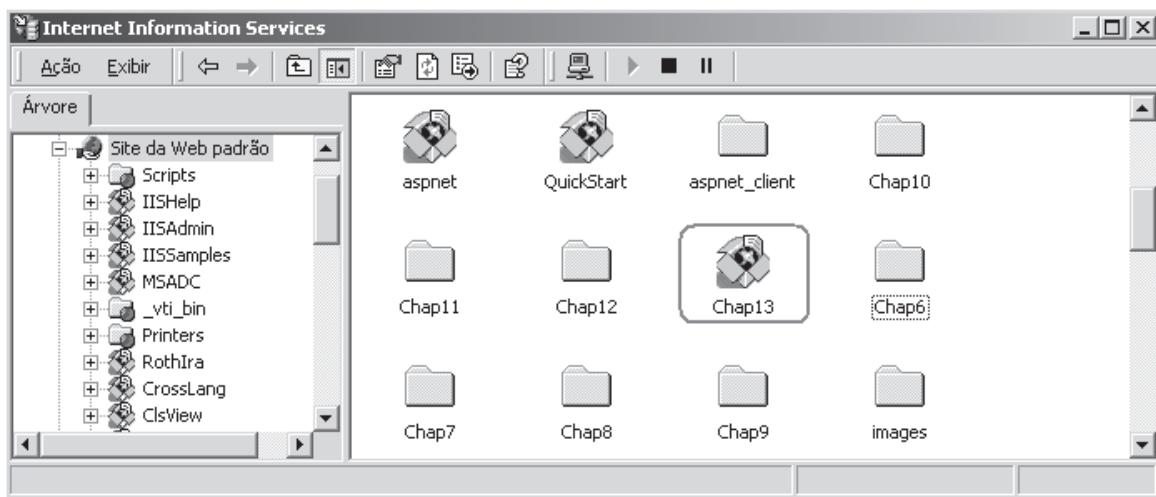


Figura 13.10: Ícone indicando que Chap13 é uma aplicação Web.

8. Feche o Gerenciador de serviços de Internet.

Utilizando o Web Service em uma Página ASP.NET

Agora o passo final: Vamos criar uma página ASP.NET que utiliza o Web Service CalculosLegais, que criamos nos passos anteriores.

Exemplo: Vamos criar uma página ASP.NET onde temos três controles:

- ◆ Um controle DropDownList, onde são exibidas as opções: RS, SC, PR, SP, RJ e Outros.
- ◆ Um controle TextBox, onde o usuário digita o valor da compra.
- ◆ Um controle TextBox, onde o usuário digita o valor do desconto.
- ◆ Um controle Label, onde é exibido o valor final, calculado pelo método CalculaImposto, do Web Service CalculosLegais.

Na Listagem 13.3 temos o código para o exemplo proposto.

Listagem 13.3 – Página ASP.NET utilizando um Web Service – Chap13Ex1.aspx.

```
<%@ Page Language="C#" Debug="true" %>
<html>

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

public void Enviar_Click(Object sender,EventArgs e)
{
    CalculosLegais ChamaCalculo = new CalculosLegais();
```

```

// Declaro variáveis para conter valores do formulário.

string txtEstado;
string txtValor;
string txtDesconto;

txtEstado=Estado.SelectedItem.Value;
txtValor=Valor.Text;
txtDesconto=Desconto.Text;

double txtValorFinal;

txtValorFinal=ChamaCalculo.CalculaImposto(Convert.ToInt32(txtValor),
Convert.ToInt32(txtDesconto), txtEstado);

Exibe.Value = "Preço Inicial      : " + txtValor +"\n"+
"Estado       : " + txtEstado + "\n"+
"Desconto     : " + txtDesconto + "\n"+
"*****" + "\n"+
"PREÇO FINAL   : " + Convert.ToString(txtValorFinal);

}

</script>

<body>

<form method=post runat="server">
<H2> Utilização do Web Service - CalculosLegais!!</H2>

<asp:RequiredFieldValidator
    id="Requer_Valor"
    ControlToValidate="Valor"
    Type="String"
    ErrorMessage="*"
    Text="DIGITE UM VALOR PARA O PREÇO."
    ForeColor="Red"
    runat="server">

```

```
>

<asp:RequiredFieldValidator
    id="Requer_Desconto"
    ControlToValidate="Desconto"
    Type="String"
    ErrorMessage="*"
    Text="DIGITE UM VALOR PARA O DESCONTO."
    ForeColor="Red"
    runat="server"
/>

<table>
<tr>
    <td><b>Selecione o Estado:</b> </td>
    <td>
        <asp:DropDownList id="Estado" runat="server">
            <asp:ListItem>RS</asp:ListItem>
            <asp:ListItem>SC</asp:ListItem>
            <asp:ListItem>PR</asp:ListItem>
            <asp:ListItem>SP</asp:ListItem>
            <asp:ListItem>RJ</asp:ListItem>
            <asp:ListItem>Outros</asp:ListItem>
        </asp:DropDownList>
    </td>
</tr>

    <td><b>Preço:</b> </td>
    <td>
        <asp:TextBox runat=server
            id="Valor"
            Text=""
            Font_Face="Arial"
            Font_Size="3"
            BackColor="Cyan"
            ForeColor="Blue"
            TextMode="SingleLine"
        />
    </td>
</tr>
```

```
        Columns="40"  
    />  
  
    </td>  
  </tr>  
  
  <tr>  
    <td><B>Desconto:</B> </td>  
    <td>  
  
      <asp:TextBox runat=server  
        id="Desconto"  
        Text=""  
        Font_Face="Arial"  
        Font_Size="3"  
        BackColor="Cyan"  
        ForeColor="Blue"  
        TextMode="SingleLine"  
        Columns="40"  
      />  
  
      </td>  
    </tr>  
  
    <tr>  
      <td><B>Valor Final:</B></td>  
      <td><textarea  
        id=>Exibe>  
        cols=>40"  
        rows="5"  
        runat="server"  
      />  
    </td>  
  </tr>  
  
  <tr>  
    <td><B>Clique no botão-></B></td>  
    <td><input type=submit  
value="Calcular"
```

```
OnServerClick="Enviar_Click"
runat="server">
</td>
</tr>
</table>

</form>

</body>
</html>
```

Digite o código da Listagem 13.1 e salve o mesmo em um arquivo chamado chap13ex1.aspx, na pasta chap13, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap13/chap13ex1.aspx>

Ao carregar a página, digite os seguintes valores:

- ◆ Selecione RS na lista de Estados.
- ◆ Digite 1200 no campo Preço.
- ◆ Digite 20 no campo Desconto.

Dê um clique no botão Calcular.

Você obtém o resultado indicado na Figura 13.11.

Comentários sobre o código do exemplo – Chap13ex1.aspx.

- ◆ Para construção do formulário, além dos controles para entrada de informações, utilizamos dois controles de validação, para garantir que os campos Preço e Desconto sejam preenchidos.
- ◆ Utilizamos o evento Click do botão, para criar a lógica da página. Em resposta ao evento Click, definimos o procedimento Enviar_Click. Neste procedimento nós seguimos as seguintes etapas:

1. Criamos uma variável ChamaCalculo, a qual é uma instância do WebService CalculosLegais:

```
CalculosLegais ChamaCalculo = new CalculosLegais();
```

2. Utilizamos algumas variáveis do tipo String, para armazenar os valores que o usuário inseriu no formulário:

```
string txtEstado;
string txtValor;
string txtDesconto;
txtEstado=Estado.SelectedItem.Value;
txtValor=Valor.Text;
txtDesconto=Desconto.Text;
```

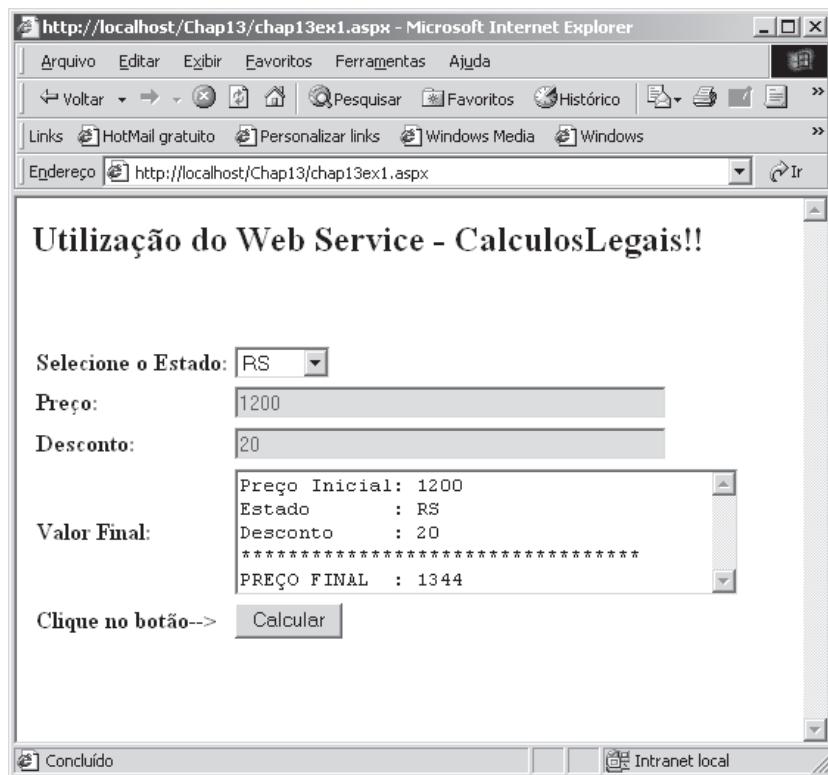


Figura 13.11: Página ASP.NET utilizando o WebService CalculosLegais.

- Declaramos uma variável do tipo double, que irá conter o valor retornado pelo método CalculaImposto, do WebService CalculosLegais.

```
double txtValorFinal;
```

- Como a variável ChamaCalculo é do tipo CalculosLegais, esta variável herda o método CalculaImposto do Web Service CalculosLegais. Chamamos este método para fazer o cálculo do preço final, com base no Estado selecionado, no Preço informado e no percentual de desconto. Estes valores são passados como parâmetros para o método CalculaImposto. Observe que temos que fazer as devidas conversões, pois caso contrário obteremos um erro. Por exemplo, o parâmetro txtValor é do tipo String, porém o método CalculaImposto espera receber um parâmetro do tipo int. Utilizamos Convert.ToInt32 para fazer a conversão:

```
txtValorFinal = ChamaCalculo.CalculaImposto(Convert.ToInt32(txtValor),
                                              Convert.ToInt32(txtDesconto), txtEstado);
```

- Em seguida montamos uma string que exibe os valores digitados pelo usuário e o valor calculado pelo método CalculaImposto. Esta string é atribuída à propriedade Text, do controle Exibe:

```
Exibe.Text = "Preço Inicial: " + txtValor + "\n" +
             "Estado      : " + txtEstado + "\n" +
             "Desconto    : " + txtDesconto + "\n" +
             "*****" + "\n" +
             "PREÇO FINAL : " + Convert.ToString(txtValorFinal);
```

Observe que agora podemos reaproveitar a funcionalidade do método CalculaImposto em qualquer página ASP.NET da aplicação Chap13.

Apenas para recapitular, eis os passos necessários para a criação e utilização de um Web Service, utilizando C# e ASP.NET:

1. Criação do código-fonte em um arquivo .asmx.
2. Testar a funcionalidade do arquivos .asmx.
3. Criar um proxy para o Web Service, utilizando o utilitário wsdl.exe.
4. Compilar o proxy gerado no item 3, para gerar uma DLL.
5. Se a pasta onde está a página ASP.NET que vai utilizar o Web Service ainda não for uma aplicação Web, utilizar o Gerenciador de serviços de Internet, para transformá-la em uma aplicação Web.
6. Criar uma pasta bin dentro da pasta correspondente à aplicação Web.
7. Copiar a DLL para dentro da pasta bin.
8. Criar uma ou mais páginas ASP.NET que utilizam o Web Service.

O Novo Ambiente Gráfico de Desenvolvimento – Visual Studio .NET

O Visual Studio .NET é o ambiente gráfico, para desenvolvimento de aplicações .NET, dentre elas páginas ASP.NET e aplicações Web. Conforme veremos neste tópico, uma das grandes vantagens do Visual Studio .NET é que podemos criar páginas ASP.NET, com a mesma facilidade com que criávamos aplicações para o Windows, utilizando o Visual Basic 6.0 ou o Delphi 6.0.

Com o Visual Studio .NET podemos criar páginas ASP.NET, simplesmente arrastando elementos em uma interface gráfica, configurando propriedades para estes elementos e definindo código para ser executado em resposta a eventos disparados pelos elementos da página.

Inicialmente veremos quais as principais novidades do Visual Studio .NET e daremos uma olhada rápida no Ambiente de Desenvolvimento. Em seguida construiremos alguns exemplos, utilizando instruções detalhadas, passo a passo.

O que há de Novo no Visual Studio .NET

Em primeiro lugar uma interface única; ou seja, quer estejamos criando uma aplicação Win32 utilizando VB.NET ou uma aplicação Web utilizando C# e ASP.NET, o ambiente RAD (Rapid Application Development) é o mesmo. Nas versões anteriores do Visual Studio, cada linguagem tinha o seu próprio ambiente de desenvolvimento, e agora com o Visual Studio .NET todas as linguagens e projetos compartilham um ambiente comum, apenas com menus e barras de ferramentas que se adaptam de acordo com projeto e elemento que está sendo editado.

IMPORTANTE: Para que você possa acompanhar os exemplos deste tópico, faça-se necessário que você tenha o Visual Studio .NET instalado. A instalação do Visual Studio .NET é bastante simples, sendo idêntica à instalação de qualquer aplicativo Windows. Embora simples, o processo todo demora cerca de duas horas. Você pode obter uma versão de avaliação do Visual Studio .NET, no seguinte endereço: <http://msdn.microsoft.com/vstudio/nextgen/beta.asp>

Start Page: Ao abrirmos o Visual Studio .NET é carregada uma página chamada de Start Page. Esta página permite que o usuário configure o ambiente de desenvolvimento de acordo com as suas preferências. Por exemplo, pode ser que você esteja acostumado com a interface do Visual Basic 6.0 e prefira ter os menus, barras de ferramentas e teclas de atalho configuradas de uma maneira semelhante ao que era no VB 6. Para solicitar tal configuração, na página inicial (Start Page), você clica na opção My Profile e seleciona as opções indicadas na Figura 13.12.

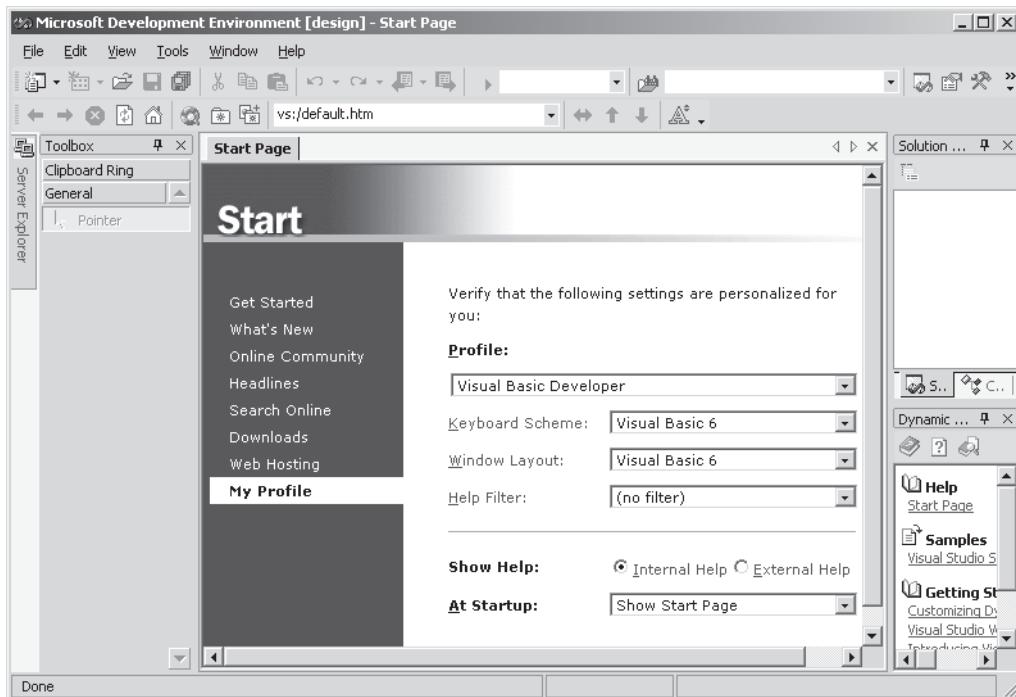


Figura 13.12: Personalizando o ambiente do Visual Studio .NET.

Suporte a Tecnologia Windows Instaler, edição de código HTML e CSS (Cascading Style Sheets), diretamente no ambiente gráfico, edição de XML e visualização do resultado das páginas também diretamente no ambiente gráfico, etc.

São inúmeras as novidades. O próprio Ambiente do Visual Studio .NET serve de assunto para um livro inteiro. Para maiores informações sobre as novidades consulte o seguinte endereço: <http://msdn.microsoft.com/vstudio/nextgen/overview.asp>

Você também pode consultar a documentação do Visual Studio .NET.

Agora vamos conhecer um pouco do novo ambiente, através da criação de alguns exemplos.

Exemplo 1: Criando uma Aplicação “ASP.NET Web Application”

Neste exemplo vamos criar uma aplicação Web, baseada em ASP.NET e C#. Para a criação desta aplicação precisamos de que o IIS (Internet Information Services) e o Visual Studio .NET estejam instalados e funcionando. Estarei utilizando um servidor com os seguintes dados:

Sistema: Windows 2000 Server em Inglês

Nome: Servidor

IP: 10.204.123.1

Diretório raiz do IIS: d:\inetpub\wwwroot

Sempre que for feita referência a uma destas configurações, substitua pelas configurações que você estiver utilizando, caso sejam diferentes das aqui apresentadas.

Criaremos uma aplicação Web chamada: AppWebChap13. Dentro destas aplicações Web, criaremos algumas páginas ASP.NET utilizando o Visual Studio .NET. Para nos conectarmos com as páginas da aplicação AppWebChap13, utilizaremos o seguinte endereço: http://servidor/AppWebChap13/nome_da_página.aspx

Para criar a aplicação Web: AppWebChap13, faça o seguinte:

1. Abra o Visual Studio .NET: Iniciar -> Programas -> Microsoft Visual Studio .NET 7.0 -> Microsoft Visual Studio .NET 7.0 (é duas vezes mesmo, uma é a pasta de opções e outra é o atalho para o Visual Studio).
2. O Visual Studio é carregado e a página “Start Page” é exibida, conforme indicado na Figura 13.13.
3. Para criar uma nova aplicação Web, devemos criar um novo Projeto. Isto pode ser feito clicando na opção “New Project” da página de abertura ou selecionando o comando File -> New -> Project ou pressionando Ctrl+Shift+N.
4. Dê um clique na opção New Project. Será aberta a janela “New Project”, onde podemos selecionar o tipo de aplicação que será criada.
5. Em Project Types, dê um clique em “Visual C# Projects”.
6. Nas opções que são exibidas, no painel da direita, dê um clique em “ASP.NET Web Application”.
7. No campo Name, digite: AppWebChap13.
8. No campo Location, digite: <http://SERVIDOR>.
9. A janela New Project deve estar com as configurações indicadas na Figura 13.14.

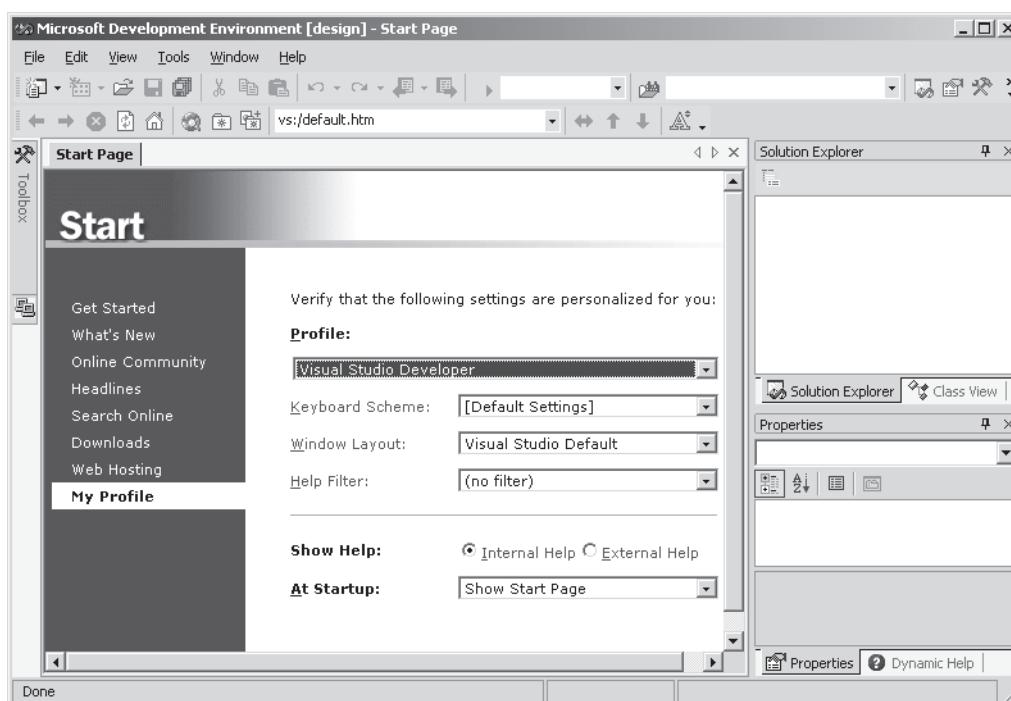


Figura 13.13: Personalizando o ambiente do Visual Studio .NET.

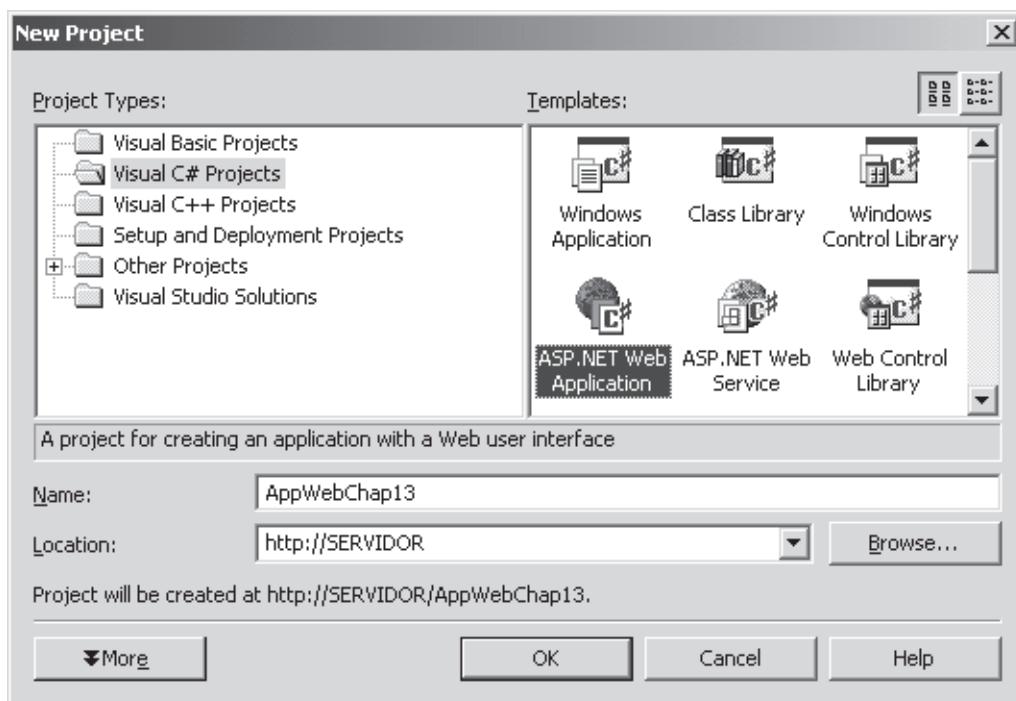


Figura 13.14: Criando o projeto AppWebChap13.

10. Dê um clique no botão OK e aguarde alguns instantes.
11. Surge uma janela indicando que uma nova aplicação Web está sendo criada, conforme indicado na Figura 13.15.

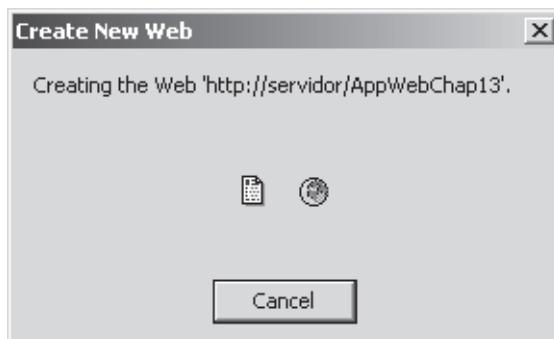


Figura 13.15: Novo projeto sendo criado.

12. O projeto é criado e, por padrão, é criada, dentre outros elementos, uma página ASP.NET chamada WebForm1.aspx. Este formulário pode ser acessado através do endereço: <http://servidor/AppWebChap13/WebForm1.aspx>.

Toda aplicação Web tem uma página que é chamada de página inicial do projeto. É o mesmo conceito de página padrão de um diretório virtual, ou seja, se simplesmente digitarmos o endereço: <http://servidor/AppWebChap13>, será carregada a página inicial do projeto, no nosso exemplo: WebForm1.aspx.

13. Além da página inicial, o Visual Studio .NET cria uma série de outros arquivos que são utilizados para a configuração da aplicação Web. Estes arquivos podem ser gerenciados através da janela Solution Explorer, destacada na Figura 13.16.

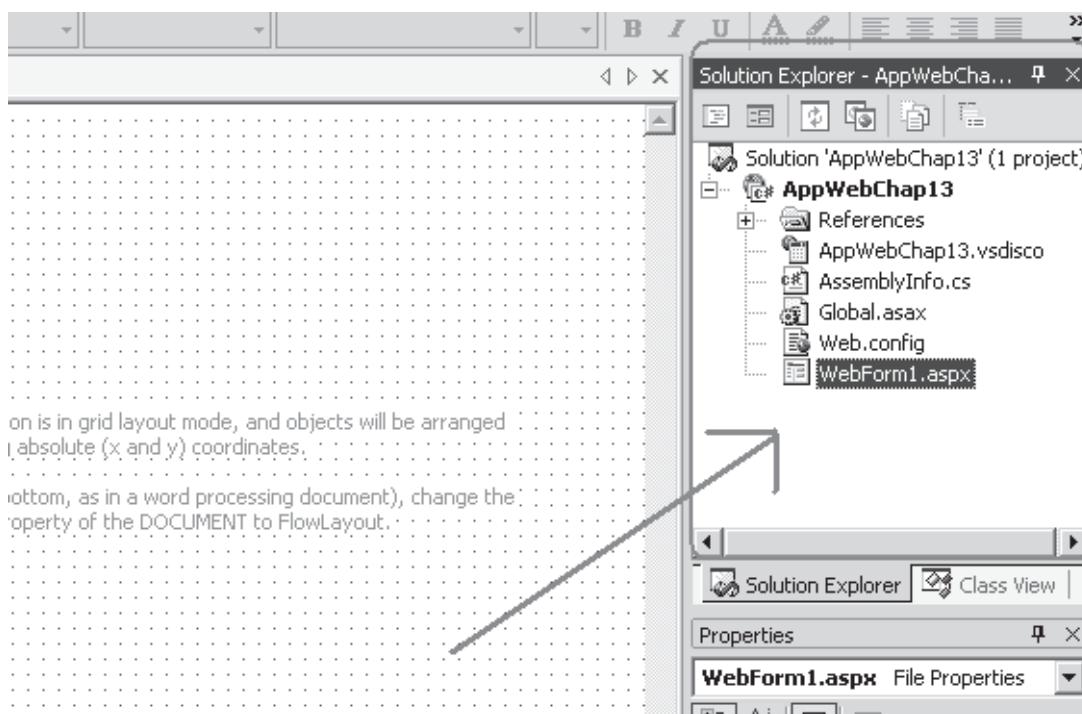


Figura 13.16: Os elementos que compõem a aplicação Web.

14. Outro elemento importante do ambiente de desenvolvimento é a Barra de Ferramentas – Toolbox. Quando você coloca o mouse sobre esta opção, é exibida uma barra com os diversos elementos que podemos colocar em nossas páginas ASP.NET, conforme destacado na Figura 13.17.

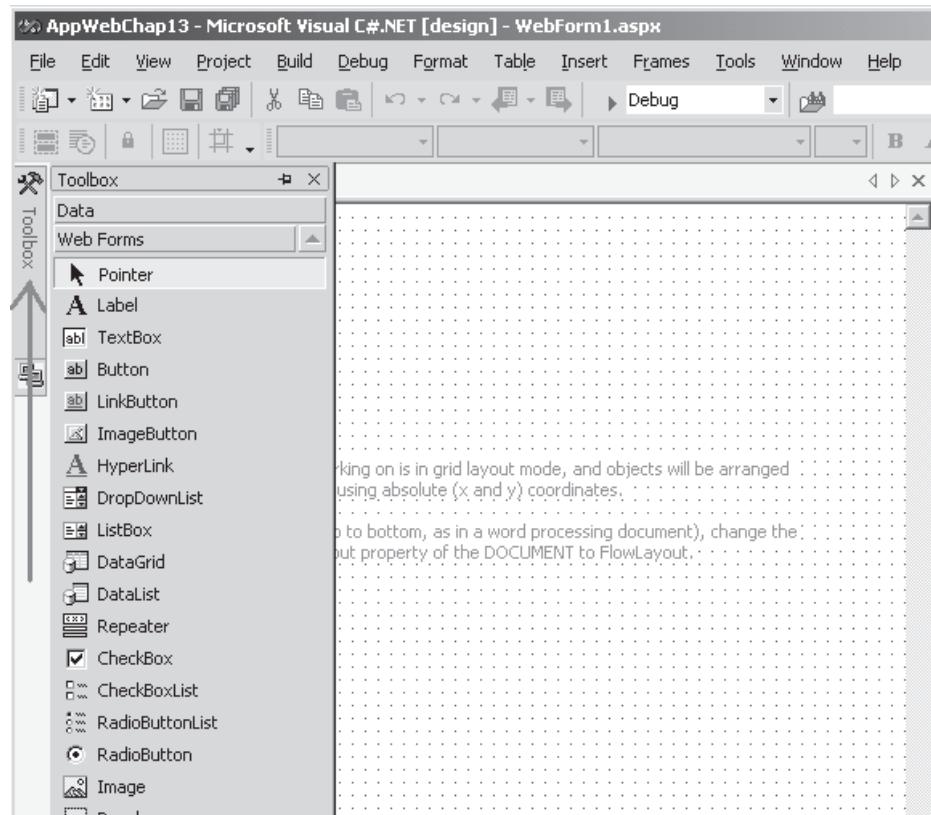


Figura 13.17: A barra de ferramentas para a criação da página ASP.NET.

15. Na Toolbox temos os diversos controles que podem ser colocados em uma página ASP.NET. São os mesmos controles que aprendemos a utilizar nos capítulos 7, 8 e 9. A diferença é que, com o Visual Studio .NET, podemos criar a página, utilizando um ambiente gráfico, onde simplesmente arrastamos os controles, configuramos as propriedades destes e criamos código que executa em resposta aos eventos gerados na página.
16. Vamos adicionar um controle do tipo Label, onde colocaremos o seguinte texto: Primeira página criada como o Visual Studio. Em seguida vamos salvar a página WebForm1.aspx e enviar as alterações para o servidor, para que a página possa ser testada.
17. Aponte o mouse para Toolbox. Nas opções que surgem, dê um clique no controle Label e arraste-o para a página WebForm1.aspx. Utilize o mouse para aumentar o tamanho do controle.
18. Para alterar o texto do label, utilizamos a janela Properties que, por padrão, encontra-se no lado direito da tela, abaixo da janela Solution Explorer.
19. Certifique-se de que o controle Label esteja selecionado e localize a propriedade Text, na janela de propriedades. Altere o valor desta propriedade para “Primeira página criada com o Visual Studio”.
20. Observe que temos diversas propriedades que podem ser alteradas, como por exemplo a cor de segundo plano (BackColor), a cor das bordas (BorderColor), etc. À medida que vamos alterando estas propriedades, o Visual Studio vai, automaticamente, gerando o código da página ASP.NET para refletir as mudanças.
21. Na propriedade BackColor, selecione um cinza claro. A sua janela deve estar semelhante à indicada na Figura 13.18.

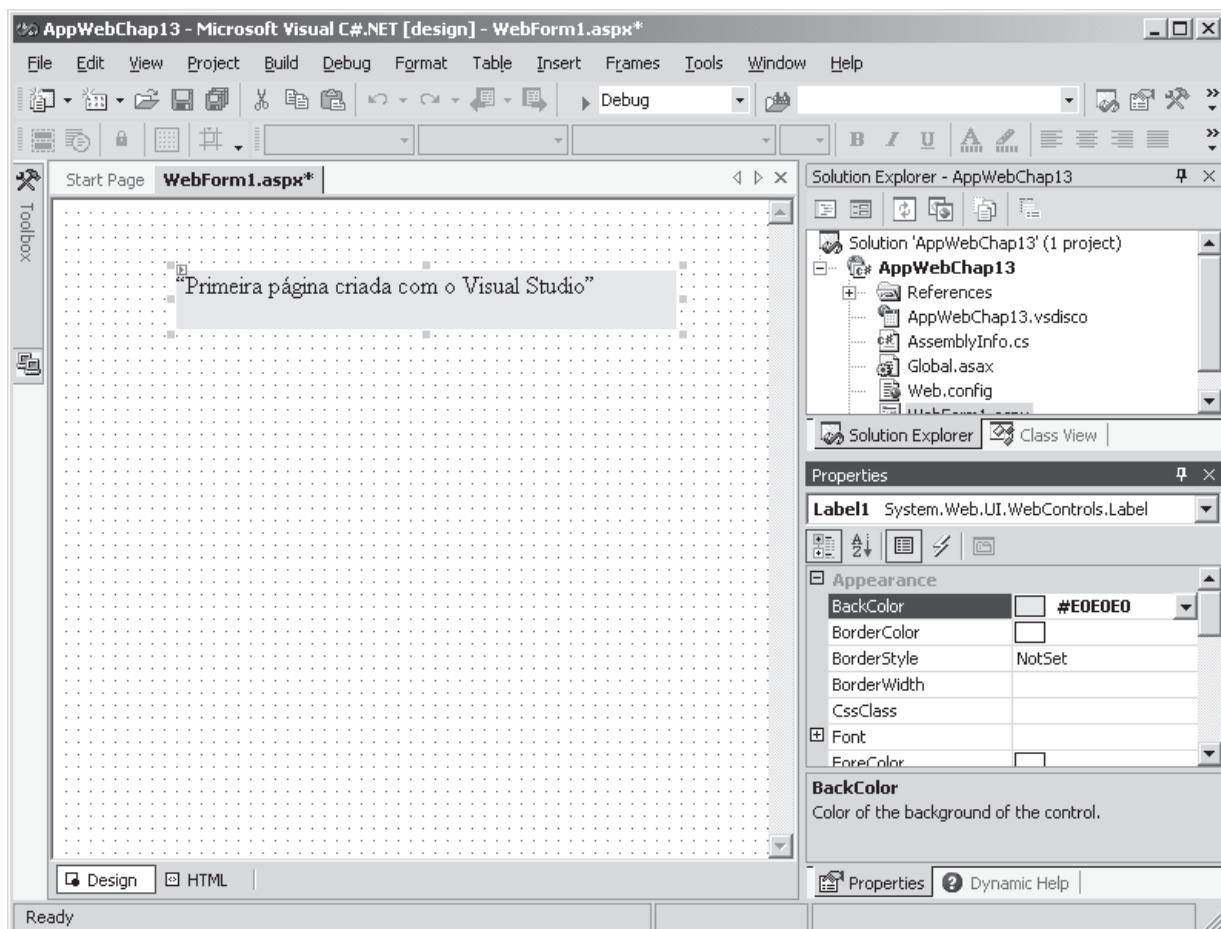


Figura 13.18: Criando a primeira página.

22. Agora vamos salvar o projeto.
23. Selecione o comando File -> Save All.
24. O Visual Studio primeiro salva uma cópia local dos arquivos e páginas que fazem parte do projeto. Em seguida precisamos enviar estas alterações para o servidor. Para isso selecione o comando Build -> Build.
25. Na parte inferior do Visual Studio surge a janela Output, informando sobre o andamento do processo de envio da aplicação para o servidor. No final, se não for encontrado nenhum erro, será exibida, na janela Output, a seguinte mensagem:

Done

Build: 1 succeeded, 0 failed, 0 skipped

26. Abra o Internet Explorer e acesse o seguinte endereço: <http://servidor/AppWebChap13/WebForm1.aspx>
27. Você obterá os resultados indicados na Figura 13.19.
28. Agora vamos analisar o código gerado pelo Visual Studio.
29. Para acessar o código da página WebForm1.aspx, dê um clique na opção HTML, que aparece ao lado da opção Design, na base da página, conforme indicado na Figura 13.20. O modo Design é o modo gráfico, onde vamos arrastando os elementos para a página e o modo HTML é o modo onde é exibido o código da página.

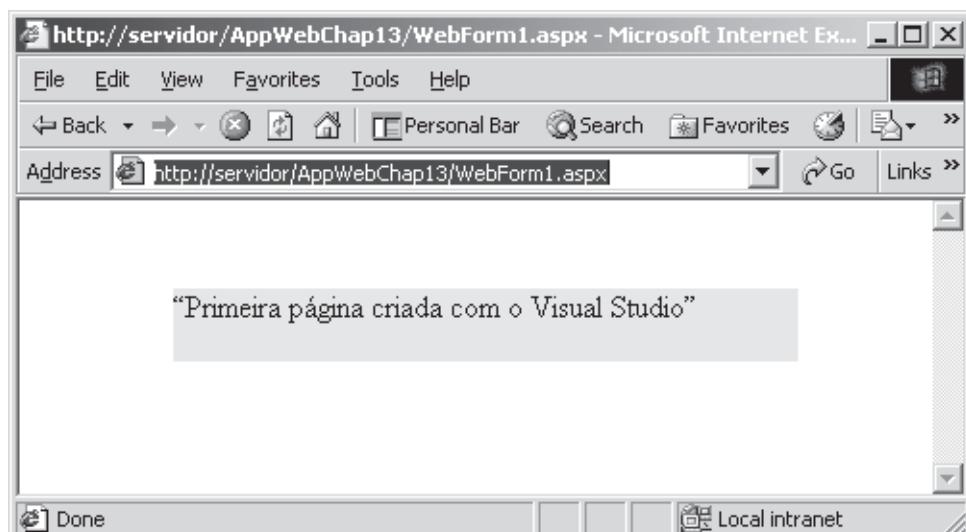


Figura 13.19: Carregando a página criada com o Visual Studio.



Figura 13.20: Exibindo o código da página.

30. Observe que o controle Label é definido com a utilização de um Web Server Control do tipo asp.Label. Este código é gerado, automaticamente, pelo Visual Studio, conforme indicado a seguir:

```
<asp:Label id=Label1
style="Z-INDEX: 101; LEFT: 80px; POSITION: absolute; TOP: 46px"
runat="server"
Width="325px"
Height="38px"
BackColor="#E0E0E0">
"Primeira página criada com o Visual Studio"
</asp:Label>
```

31. Agora vamos ver uma das pequenas maravilhas do Visual Studio.
32. Clique no controle Label, após a opção runat="server". Vamos definir que o texto deve ser exibido em negrito. Para isso você lembra que existe uma propriedade Font..., alguma coisa.
 33. Experimente, simplesmente teclar a barra de espaços e observe: Será exibida uma lista com todas as propriedades, métodos e eventos disponíveis para o controle do tipo Label, conforme indicado na Figura 13.21.

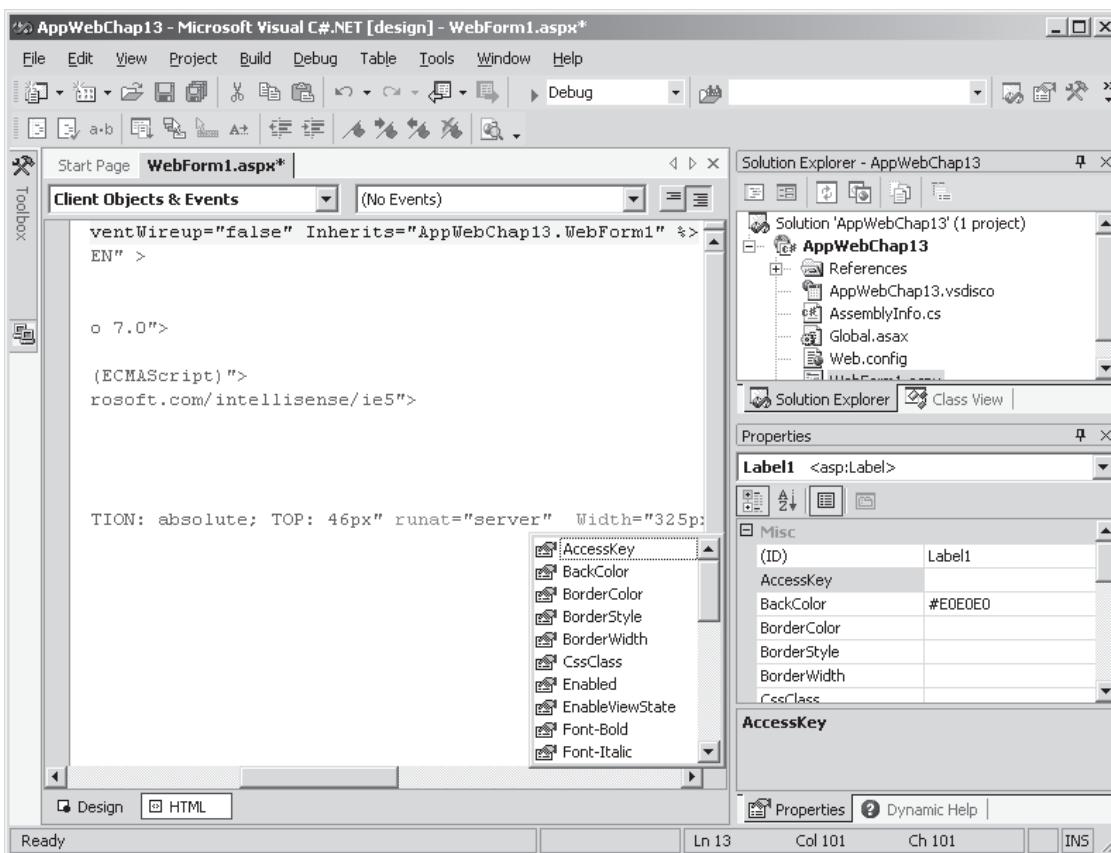


Figura 13.21: Com o Visual Studio é fácil localizar os membros de um controle.

34. Na lista que é exibida selecione a opção Font-Bold e tecle o sinal de igual (=). Observe outra maravilha: O Visual Studio mostra a lista de valores possíveis para a propriedade selecionada, no nosso exemplo: True ou False. Basta

selecionar True e pronto, está habilitado Negrito para o nosso controle. Com este recurso do Visual Studio fica muito mais fácil localizar as propriedades, métodos e eventos de um determinado elemento. Quando selecionamos um método, são destacados os parâmetros e os tipos de cada parâmetro. Este recurso é de grande valor, pois, além de localizar rapidamente a propriedade ou método desejado, reduz drasticamente a entrada de valores incompatíveis para a propriedade ou para os parâmetros de um método.

35. Vamos salvar novamente a nossa aplicação: File -> Save All.
36. Vamos atualizar a versão da aplicação no Servidor: Build -> Build.
37. Agora é só testá-la novamente, utilizando o endereço: <http://servidor/AppWebChap13/WebForm1.aspx>
38. Você obterá os resultados indicados na Figura 13.22.
39. Mantenha o Visual Studio .NET aberto.

Agora que já conhecemos os aspectos básicos da criação de uma aplicação Web, com o Visual Studio, vamos criar mais alguns exemplos, para aprender a utilizar melhor a interface gráfica.



Figura 13.22: Versão atualizada da nossa aplicação.

Exemplo 2: Criando uma Nova Página na Aplicação AppWebChap13

Neste exemplo vamos criar uma nova página ASP.NET, na aplicação AppWebChap13. Vamos chamar esta página de Form2.aspx. Neste formulário colocaremos um controle do tipo DropDownList e dois controles do tipo TextBox, para o usuário digitar informações. Também colocaremos um botão Enviar. Quando o usuário clicar no botão Enviar, utilizaremos o evento OnClick do botão, para exibir os valores digitados pelo usuário em um controle TextArea.

1. Para criar uma nova página ASP.NET utilize o seguinte comando:
File - Add New Item.
2. Na janela Add New Item que é aberta, seleciona a opção Web Form. No campo Name, digite Form2.aspx, conforme indicado na Figura 13.23.

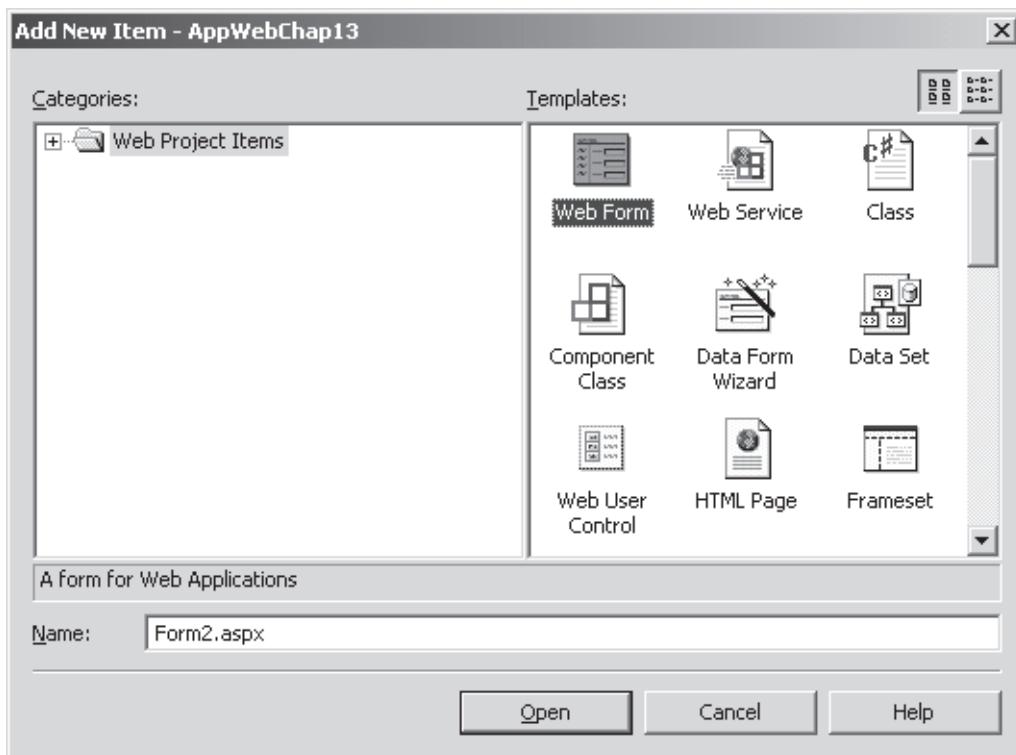


Figura 13.23: Adicionando um novo Web Form.

3. Dê um clique no botão Open. A página Form2.aspx será criada e exibida no Visual Studio.

4. Agora vamos arrastar os seguintes controles:

Um controle do tipo DropDownList.

Três controles do tipo TextBox.

Quatro controles do tipo Label, um ao lado de cada um dos controles anteriores, para identificar o conteúdo de cada controle.

Um controle do tipo Button.

5. Neste momento os controles estão desalinhados. Para alinhar os controles facilmente, você pode selecionar os controles a serem alinhados e depois utilizar o comando Format -> Align. Para selecionar vários controles você pode segurar a tecla Shift e ir clicando nos controles a serem selecionados.

6. Agora que alinhamos os controles vamos configurar as diversas propriedades de cada controle.

7. Configure a propriedade Text, para os controles Label e para o controle Button, com os valores indicados na Figura 13.24.

8. Agora vamos configurar a propriedade Id para o controle DropDownList, para os controles TextBox e para o controle Button, conforme indicado a seguir:

DropDownList	->	ID= SeleccionaPaís
Primeiro TextBox	->	ID= Nome
Segundo TextBox	->	ID= Email
Terceiro TextBox	->	ID= Exibe
Controle Button -	>	ID= Enviar

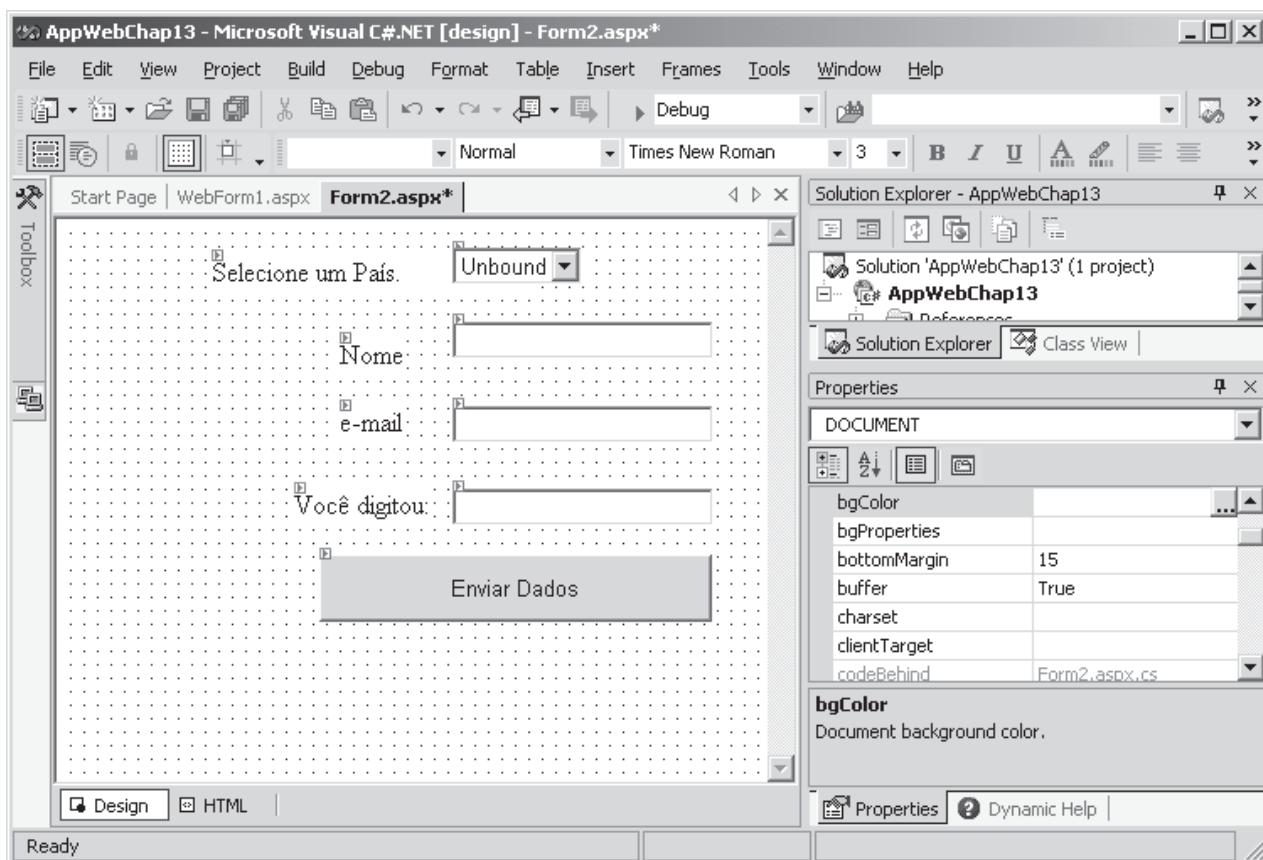


Figura 13.24: Definindo os rótulos para os controles.

9. Agora vamos configurar as demais opções dos controles do nosso formulário.
10. Vamos adicionar os seguintes itens para o controle DropDownList:

Argentina**Brasil****Chile****EUA****França**

11. Para adicionar itens em um controle DropDownList utilizamos a propriedade Items, do controle SelecionaPaís. Dê um clique neste controle para selecioná-lo. Na lista de propriedades, na janela de propriedades, localize a propriedade Items. Ao clicar nesta propriedade será habilitado um botão com reticências (...), ao lado da propriedade. Dê um clique no botão com as reticências, e será exibida a janela “ListItem Collection Editor”. Para adicionar um item à lista, clique no botão Add. Na coluna da direita, serão exibidos os campos para definição do item. Para o primeiro item, defina as propriedades conforme indicado na Figura 13.25.
12. Utilize o botão Add para adicionar os demais itens. No final a sua janela deve estar conforme indicado na Figura 13.26.
13. Dê um clique no botão OK e, pronto, a lista será construída.

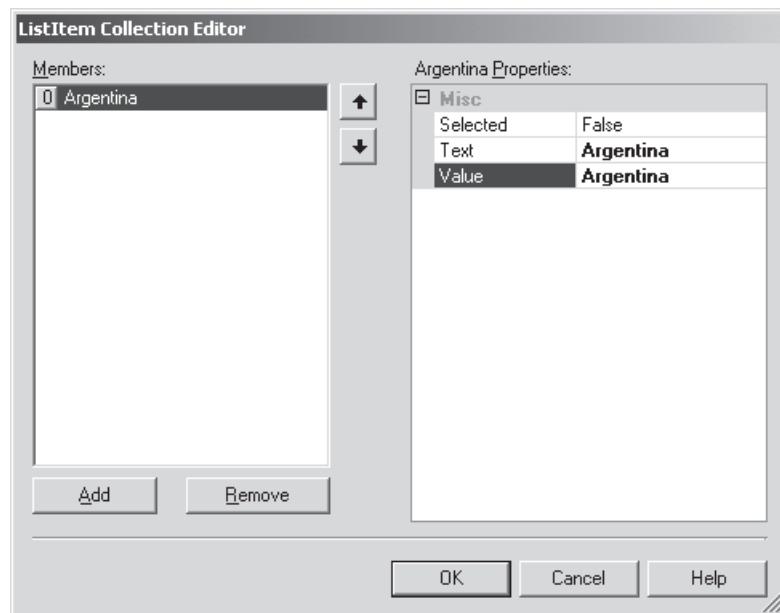


Figura 13.25: Adicionando itens ao controle DropDownList.

14. Agora precisamos criar o código que executa em resposta ao evento OnClick do botão de comando. Para definir este procedimento, basta dar um clique duplo no botão “Enviar Dados”. Automaticamente é aberta a janela de código, com a estrutura do procedimento já definida. Tudo o que temos que fazer é digitar o seguinte código:

```
string Dados;
Dados= "País: " + SelecionaPais.SelectedItem.Value + "\n"+
"Nome: " + Nome.Text+ "\n"+
"e-mail:" + Email.Text ;
Exibe.Text = Dados;
```

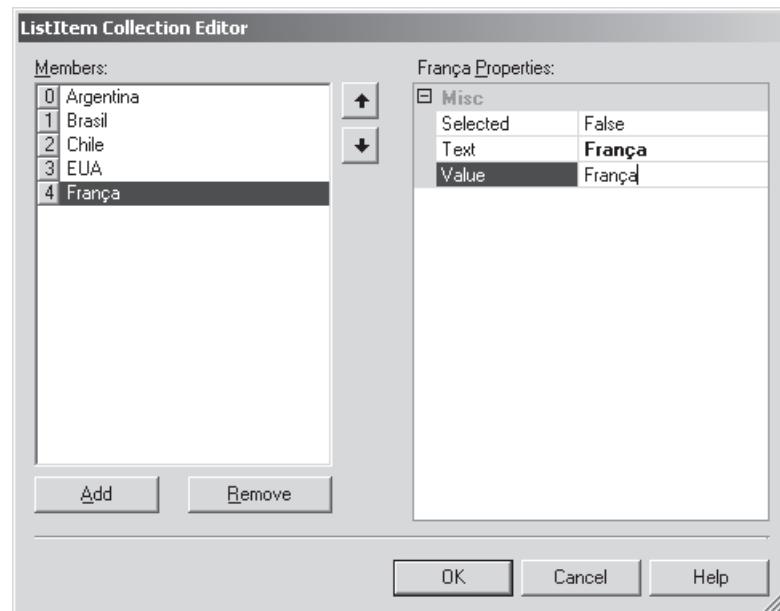


Figura 13.26: Completando a lista.

15. Agora só falta configurar a propriedade TextMode, do controle Exibe. Altere de SingleLine para MultiLine. O formulário Form2.aspx deve estar conforme indicado na Figura 13.27.

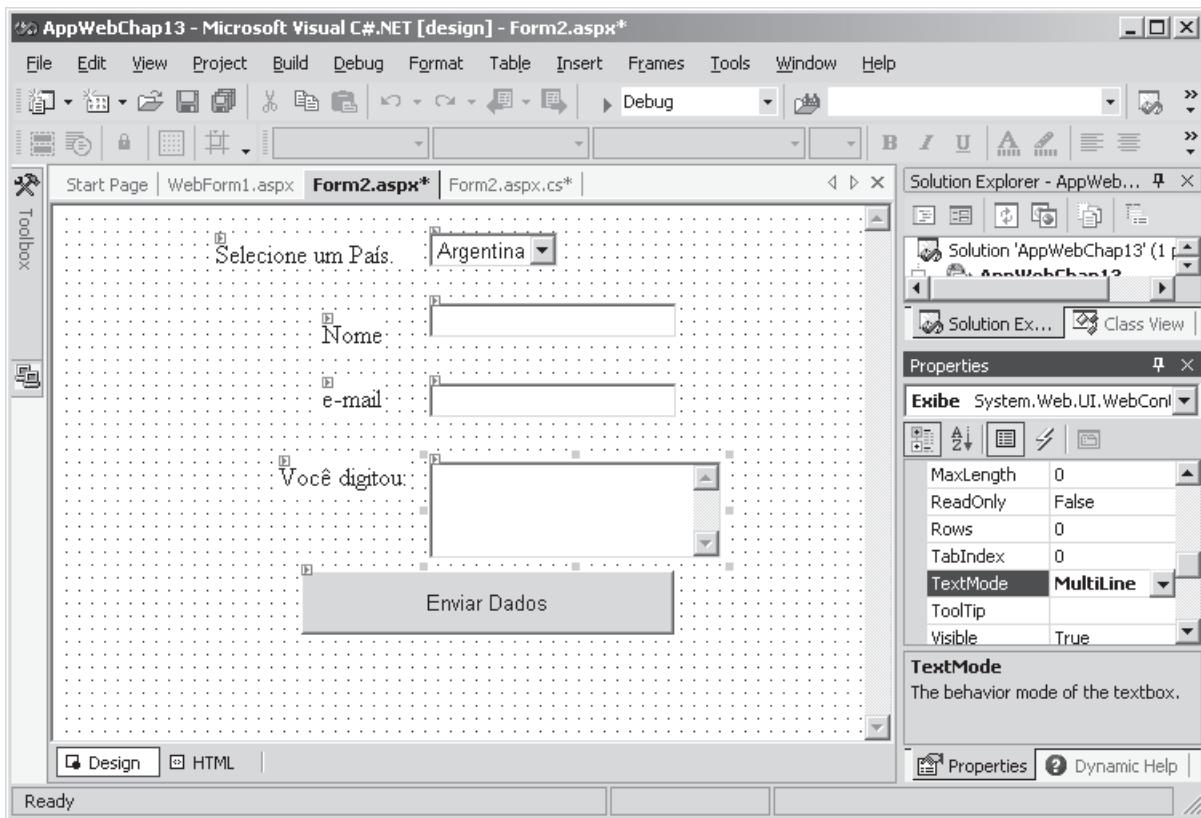


Figura 13.27: Completando o formulário.

16. Salve as alterações: File -> Save All.
 17. Vamos enviar as alterações para o servidor: Build -> Build.
 18. Agora vamos testar a página Form2.aspx, utilizando o endereço:

<http://servidor/AppWebChap13/Form2.aspx>

19. Na lista de países selecione Chile.
 20. No campo Nome digite: José da Silva.
 21. No campo e-mail digite: jsilva@abc.com.br.
 22. Dê um clique no botão Enviar Dados.
 23. Você obtém os resultados indicados na Figura 13.28.
 24. Mantenha o Visual Studio .NET aberto.

Vamos criar mais um exemplo, onde aprenderemos a conectar com um banco de dados, utilizando o Visual Studio.

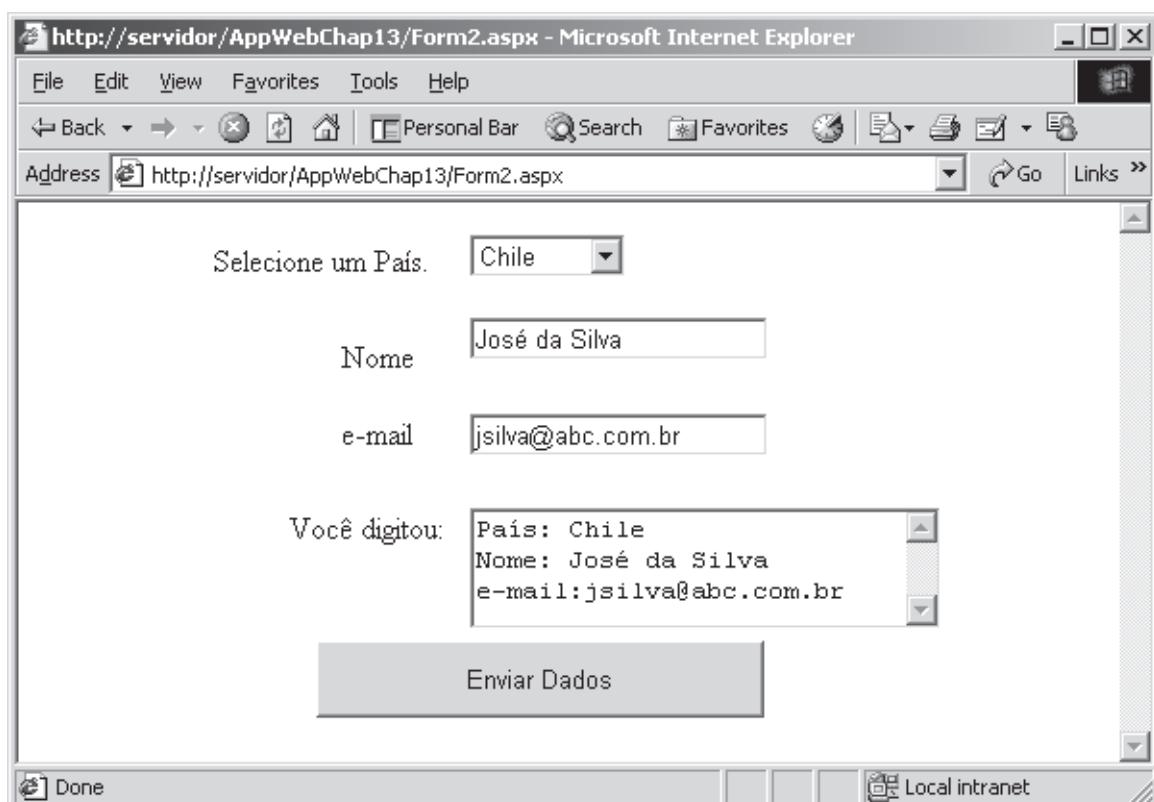


Figura 13.28: Testando a página Form2.aspx.

Exemplo 3: Conectando com um Banco de Dados do Microsoft Access

Neste exemplo vamos criar uma nova página ASP.NET, na aplicação AppWebChap13. Vamos chamar esta página de dados.aspx. Vamos utilizar um assistente do Visual Studio, assistente este que criará a página automaticamente.

A página dados.aspx, que iremos construir, exibirá dados da tabela Customers (Clientes) e da tabela Orders (Pedidos), do Banco de dados NorthWind, do SQL Server 2000. No equipamento que estou utilizando para este exemplo, tenho instalada a seguinte instância do SQL Server 2000: SERVIDOR\INSTANCIA1. Se você estiver utilizando uma instalação do SQL Server 2000, com um nome diferente, sempre que houver referência a SERVIDOR\INSTANCIA1, substitua pelo nome que você está utilizando.

1. Para criar a página dados.aspx, estando com a aplicação AppWebChap13 aberta, execute o seguinte comando:
File – Add New Item.
2. Na janela Add New Item que é aberta, selecione a opção Data Form Wizard. No campo Name, digite data.aspx, conforme indicado na Figura 13.29.
3. Dê um clique no botão Open.
4. A primeira tela do assistente “Data Form Wizard” será exibida.
5. Dê um clique no botão Next, seguindo para a próxima etapa do assistente.
6. Nesta etapa temos a opção de criar um novo objeto DataSet.
7. Dê um clique na opção “Create a new dataset named:” e no campo nome digite: ListaDePedidos, conforme indicado na Figura 13.30.

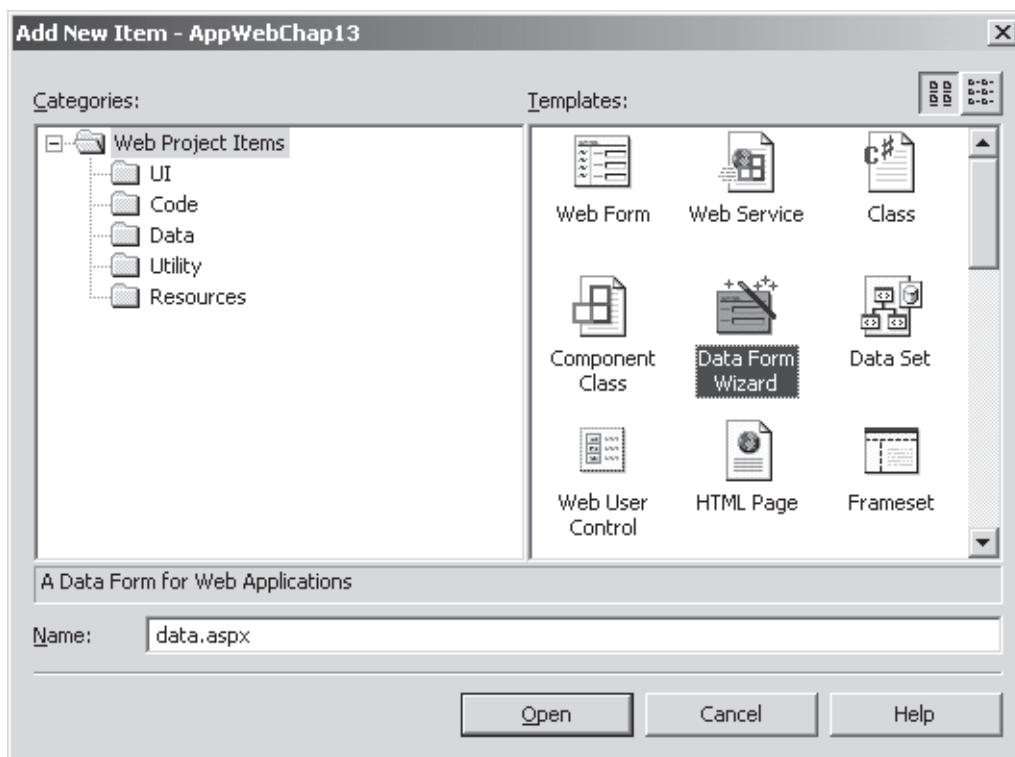


Figura 13.29: Utilizando um assistente do Visual Studio.

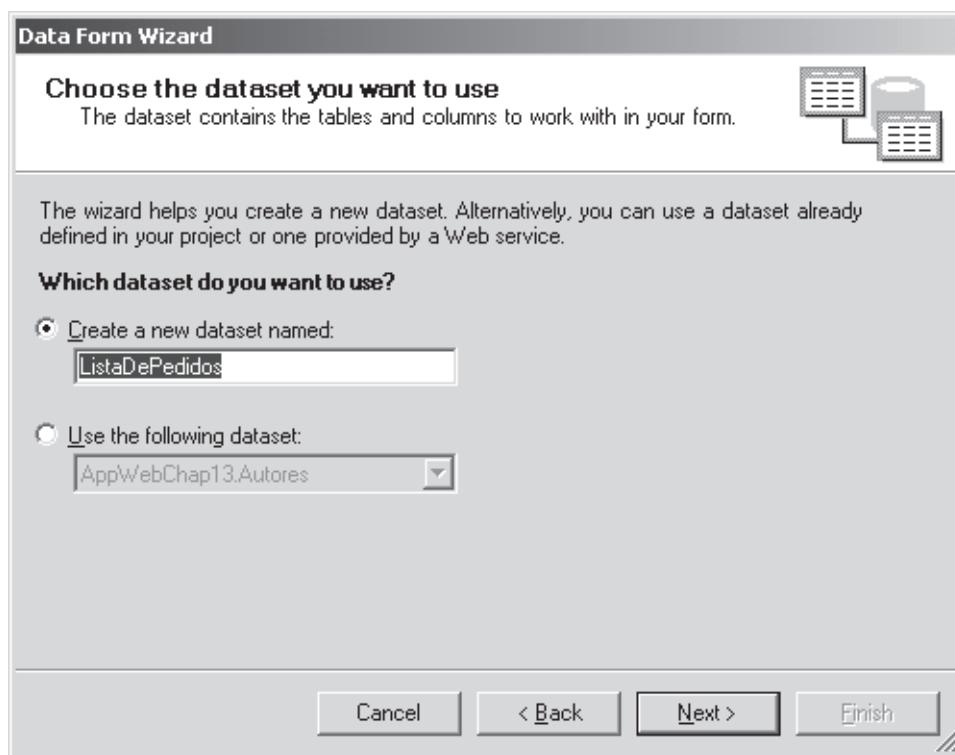


Figura 13.30: Criando um novo objeto do tipo DataSet.

8. Dê um clique no botão Next, seguindo para a próxima etapa do assistente.
9. Nesta etapa vamos definir a conexão com o banco de dados.
10. Dê um clique no botão New Connection. Será exibida a janela Connection.

11. Dê um clique na guia Provider e certifique-se de que a opção “Microsoft OLE DB Provider for SQL Server” esteja selecionada, conforme indicado na Figura 13.31.
12. Dê um clique na guia Connection.
13. Na lista “Select or enter a server name”, selecione SERVIDOR\INSTANCIA1 (ou o nome do servidor que você estiver utilizando).

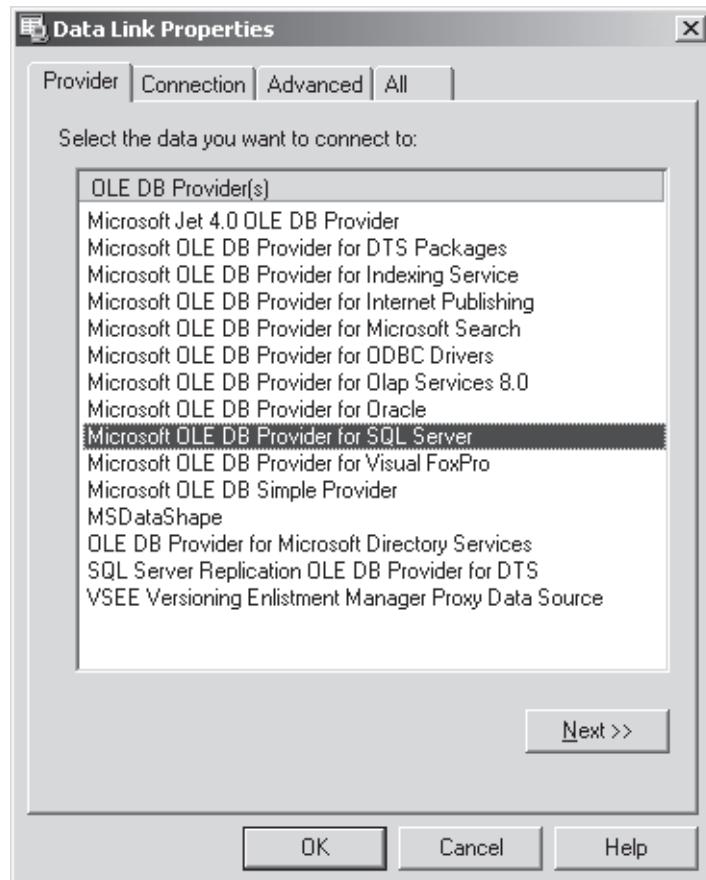


Figura 13.31: Selezionando o OLE DB Provider para o SQL Server.

14. No item 2, da guia Connection, defina as informações de conexão. No meu exemplo estou utilizando a opção “Use a specific user name and password”. Para usuário estou utilizando “as”, com senha em branco. Marquei as opções “Blank Password” e “Allow saving password”. No item 3, selecionamos o banco de dados com o qual queremos nos conectar. Para o nosso exemplo, selecione NorthWind. A guia Connection deve estar com as configurações indicadas na Figura 13.32.
15. Dê um clique no botão OK. Você estará de volta ao assistente Data Form Wizard.
16. Dê um clique no botão Next, seguindo para a próxima etapa do assistente.
17. Nesta etapa definimos de quais tabelas queremos acessar informações. Clique na tabela Customers e dê um clique no botão (>) para adicionar a tabela à coluna da direita. Repita a operação para a tabela Orders. Sua tela deve estar conforme indicado na Figura 13.33.
18. Dê um clique no botão Next, seguindo para a próxima etapa do assistente.

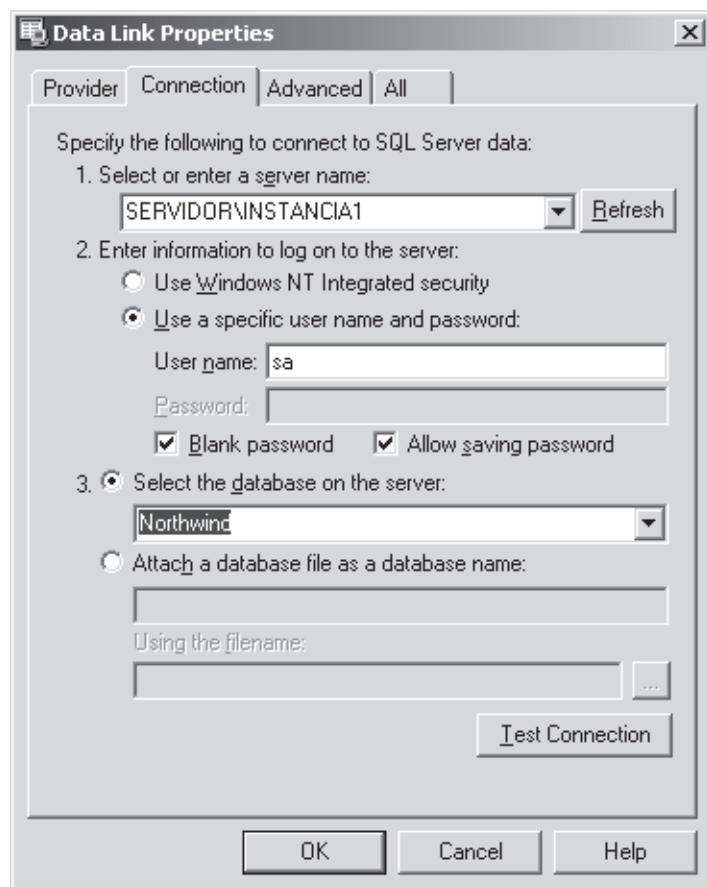


Figura 13.32: Definindo as configurações de conexão.

19. Nesta etapa, que só existe quando temos duas ou mais tabelas, podemos definir um relacionamento entre as tabelas. Para o nosso exemplo, vamos definir um relacionamento do tipo “Um para Vários”, entre a tabela Customers (lado um) e a tabela Orders (lado vários). O relacionamento será estabelecido através do campo comum às duas tabelas: CustomerID. Na prática este relacionamento está definindo que cada cliente é cadastrado uma única vez e pode fazer vários pedidos. Para maiores informações sobre relacionamentos e o modelo de dados relacionais, consulte o Anexo II.
20. No campo Name digite RelClientesPedidos.
21. Na lista Parent table, selecione Customers.
22. Na lista Child table, selecione Orders.
23. Na lista Keys, selecione CustomerID, para as duas listas da primeira linha. Depois dê um clique no botão (>), para criar o relacionamento. Observe que o relacionamento RelClientesPedidos é exibido na coluna Relations. Clique neste relacionamento e as propriedades definidas para o relacionamento RelClientesPedidos serão exibidas, conforme indicado na Figura 13.34.

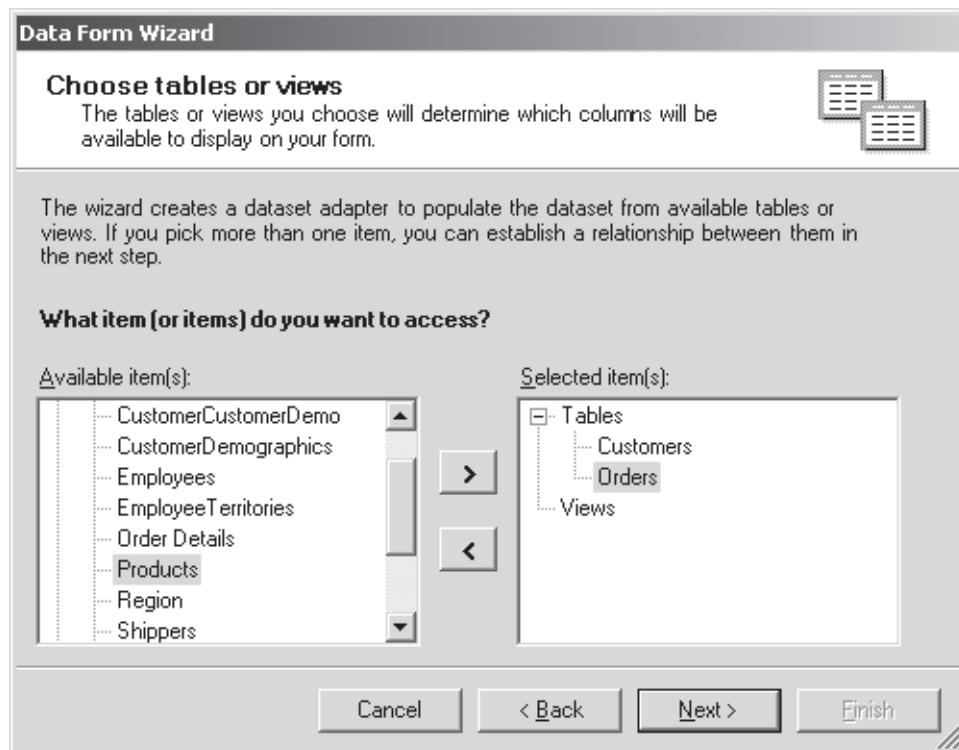


Figura 13.33: Definindo as tabelas que serão acessadas.

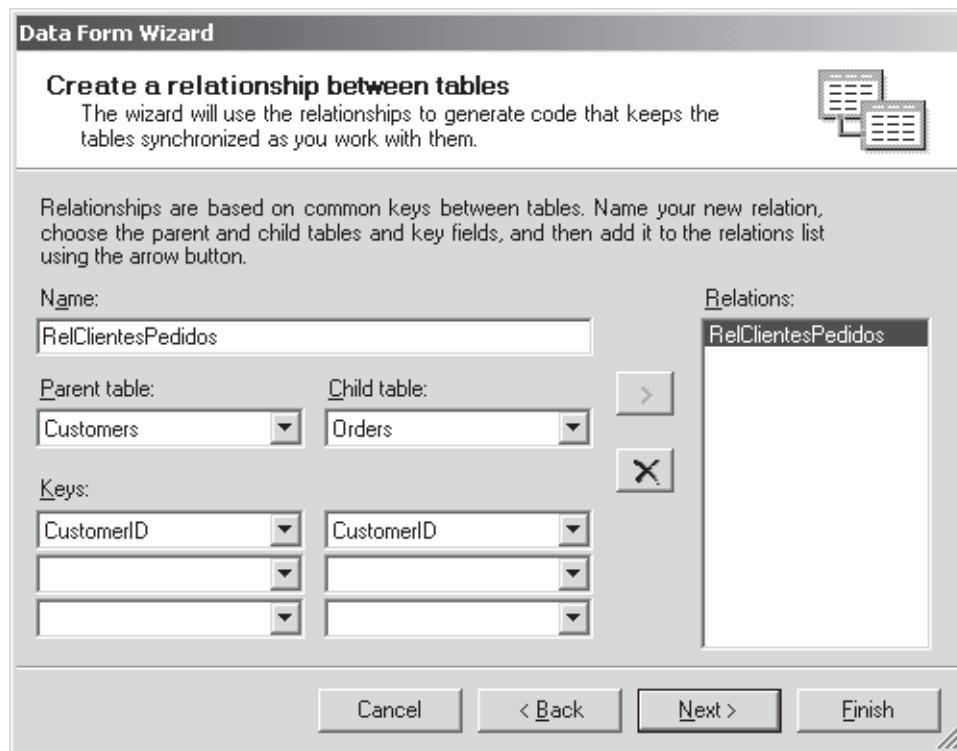


Figura 13.34 Definindo um relacionamento entre as tabelas Customers e Orders.

24. Dê um clique no botão Next, seguindo para a próxima etapa do assistente.

25. Nesta etapa você define quais campos de cada tabela devem ser exibidos na página. Deixe marcados apenas os seguintes campos:

Tabela Customer: CompanyName

Phone

Tabela Orders: OrderID

Freight

ShipCity

ShipCountry

26. Dê um clique no botão Finish.

27. O assistente será encerrado e a página data.aspx. será criada, conforme indicado na Figura 13.35.

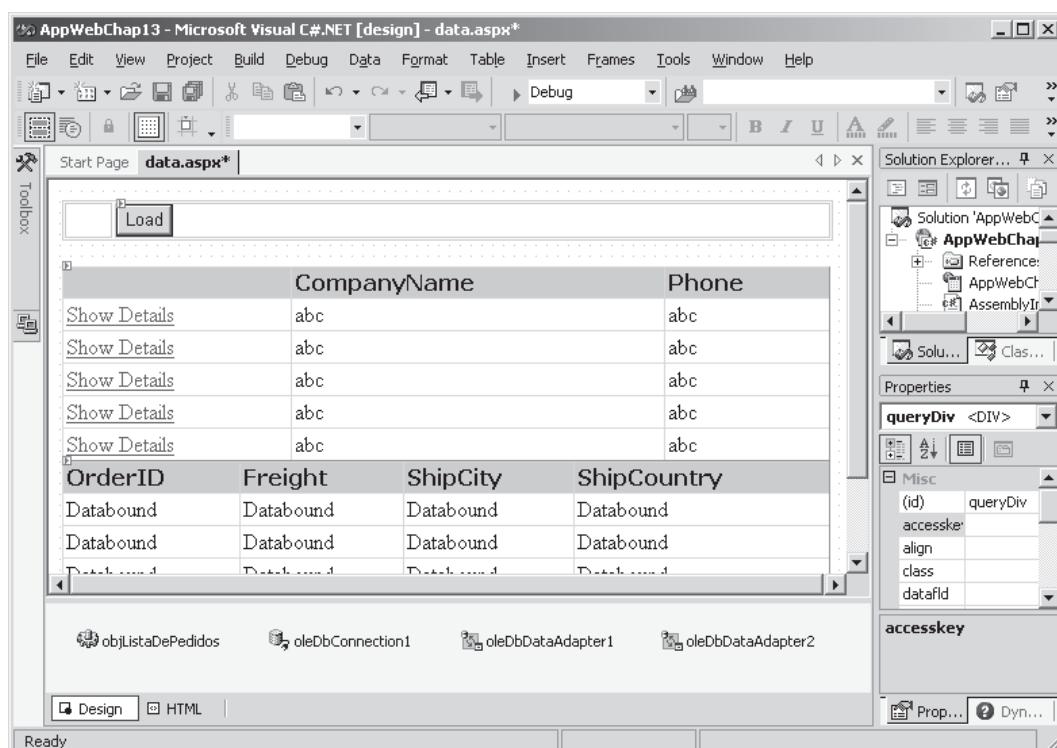


Figura 13.35: Página criada com o assistente Data Form Wizard.

28. Observe, na parte de baixo da página, que foram criados diversos objetos que dão suporte a conexão:

DataSet objListaDePedidos

OleDbConnection oleDbConnection1

OleDbDataAdapter oleDbDataAdapter1

OleDbDataAdapter oleDbDataAdapter2

29. Agora vamos testar a página dados.aspx.

30. Salve as alterações: File -> Save All.

31. Vamos enviar as alterações para o servidor: Build -> Build.

32. Agora vamos testar a página dados.aspx, utilizando o endereço: <http://servidor/AppWebChap13/data.aspx>
33. Será carregada a página data.aspx, onde é exibido um botão Load. Dê um clique no botão Load. É exibida uma lista com o nome e o telefone dos clientes, além de um link “Show Details”. Ao clicar neste link, será exibida, no final da página, uma lista com os pedidos para o respectivo cliente, conforme indicado na Figura 13.36.

The screenshot shows a Microsoft Internet Explorer window with the URL <http://servidor/AppWebChap13/data.aspx>. The page displays two tables. The first table has columns for 'Show Details', 'Nome do Cliente' (Client Name), and 'Telefone' (Phone). The second table has columns for 'OrderID', 'Freight', 'ShipCity', and 'ShipCountry'. The data from both tables is summarized below:

Clients		Order ID	Freight	Ship City	Ship Country
Show Details	Wennington Importadora	10374	3,94	Warszawa	Poland
Show Details	White Clover Markets	10611	80,65	Warszawa	Poland
Show Details	Wilman Kala	10792	23,79	Warszawa	Poland
Show Details	Wolski Zajazd	10870	12,04	Warszawa	Poland
		10906	26,29	Warszawa	Poland
		10998	20,31	Warszawa	Poland
		11044	8,72	Warszawa	Poland

Figura 13.36: Página data.aspx criada com o assistente Web Data Wizard.

Observe que com a ajuda do assistente criamos rapidamente um formulário que exibe dados de duas tabelas diferentes, do Banco de dados NorthWind, do SQL Server 2000. Evidentemente que o assistente não é a solução para todos os casos. Eu aconselho que você utilize o assistente para criar as funcionalidades básicas da página e depois faça as alterações necessárias.

Conforme descrito no início deste tópico, o Visual Studio .NET é assunto para um livro inteiro. Para o escopo proposto, encerramos o nosso estudo do Visual Studio por aqui.

Conclusão

Neste capítulo aprendemos sobre o conceito de Web Services como uma forma de criar aplicações distribuídas, onde as funcionalidades da aplicação podem ser fornecidas por Web Services localizados em diferentes servidores, através da Internet.

Em seguida aprendemos a criar e utilizar um Web Service em uma página ASP.NET. Para isso, seguimos as seguintes etapas:

1. Criação do código-fonte em um arquivo .asmx.
2. Testar a funcionalidade do arquivos .asmx.
3. Criar um proxy para o Web Service, utilizando o utilitário wsdl.exe.
4. Compilar o proxy gerado no item 3, para gerar uma DLL.
5. Se a pasta onde está a página ASP.NET que vai utilizar o Web Service ainda não for uma aplicação Web, utilizar o Gerenciador de serviços de Internet, para transformá-la em uma aplicação Web.
6. Criar uma pasta bin dentro da pasta correspondente à aplicação Web.
7. Copiar a DLL para dentro da pasta bin.
8. Criar uma ou mais páginas ASP.NET que utilizam o Web Service.

Em seguida apresentamos o novo ambiente de desenvolvimento para o ambiente .NET: Visual Studio .NET. Falamos sobre as principais características do novo ambiente e aprendemos, passo a passo, a criar algumas páginas ASP.NET, utilizando o ambiente gráfico do Visual Studio .NET.

No próximo capítulo falaremos sobre a segurança de aplicações e páginas ASP.NET.

Introdução

Quando se fala de Internet nos dias de hoje, o assunto mais tratado, sem nenhuma dúvida, é sobre Segurança. Muitos relatos, alguns verdadeiros e outros mais fantasiosos, sobre invasões mirabolantes, roubo de número de cartões de créditos, acesso a informações sigilosas de órgãos governamentais e assim por diante.

Não podemos negar que o problema de segurança existe e é crítico, principalmente no momento em que o Comércio Eletrônico é, mais do que uma realidade, uma necessidade e um diferencial competitivo para as empresas. O diferencial competitivo não é entrar ou não no mundo do Comércio Eletrônico, o diferencial é criar serviços agregados ao Comércio Eletrônico, capazes de gerar diferenciais competitivos. Assuntos como fidelização do cliente, melhorias nos sistemas de CRM – Customer Relationship Management (Gerenciamento das Relações com o Cliente), B2B – Business to Business, B2C – Business to Consumer e outros, estão em evidência.

Porém sistemas de Comércio Eletrônico, CRM e assemelhados exigem acesso a um conjunto de dados estratégicos da empresa. Uma vez que estes sistemas estão acessíveis à Internet, os dados empresariais precisam estar protegidos. Neste ponto é que a questão segurança é de fundamental importância. Existem os mais variados tipos de ataques pela Internet. Um engano comum é pensar que o único tipo de ataque capaz de causar prejuízos é aquele que rouba ou destrói dados. No caso de um site de comércio eletrônico, qualquer ataque que torne o site indisponível por um determinado período de tempo causa prejuízos incalculáveis, pois além das compras que deixaram de ser feitas no período de indisponibilidade, tem a questão da imagem da empresa, sem contar que o cliente pode ter feito a compra no site do concorrente e passar a fazer as próximas compras também do concorrente.

Por todos estes motivos é que a questão de segurança é fundamental e deveria ser prioritária quando tratamos de aplicações Web. Outro fato importante a ser mencionado é que a maioria dos ataques, ao contrário do que muitos pensam, é originado dentro da Intranet da própria empresa. Pode ser um funcionário descontente ou desonesto, ou um usuário com permissões de acesso indevidas, que causa algum prejuízo por imperícia técnica. O fato é que a questão de segurança não deve ser tratada apenas como uma questão de proteção contra “as forças do mal que vêm da Internet”. O fato é que precisamos definir uma política de segurança que permita que todos possam realizar o seu trabalho, porém com os níveis de permissão adequados.

Além de definir uma política de segurança, é necessária a ampla divulgação da mesma. É alarmante constatar que muitas empresas não possuem uma política de segurança definida, ou, quando têm, a política não é adequadamente divulgada.

CAPÍTULO

14

Segurança de Aplicações Web com o IIS 5.0 e ASP.NET

Outro erro bastante comum é achar que a questão de segurança é responsabilidade somente da equipe de desenvolvimento ou somente do Administrador da Rede. Na verdade o item segurança é bastante complexo e exige que o pessoal de desenvolvimento trabalhe em sintonia com a Administração da rede e com todos na empresa.

Conforme veremos neste capítulo, existem aspectos de segurança que são de responsabilidade do desenvolvimento e outros que são de responsabilidade da Administração de rede. Na verdade o que se faz é criar várias barreiras para que o hacker não tenha sucesso em sua tentativa de invasão. Algumas destas barreiras são criadas na própria rede da empresa e outras em nível de aplicação Web.

Neste capítulo estaremos tratando de questões como:

- ◆ Autenticação do usuário com o servidor Web.
- ◆ Aspectos de segurança em nível de Windows 2000 Server.
- ◆ Aspectos de segurança no IIS 5.0.
- ◆ Implementando uma segurança em nível de Banco de dados.

Este aspectos têm a ver com o ambiente – Sistema Operacional (Windows 2000, Windows XP, Windows 2002) mais Servidor Web (IIS 5.0), onde rodam as aplicações Web criadas com ASP.NET. Estes aspectos são os mesmos, quer estejamos trabalhando com ASP ou ASP.NET.

Na segunda parte do capítulo, estudaremos alguns aspectos de segurança que são específicos do Framework .NET e do ASP.NET. Conforme veremos existe uma série de classes que nos permitem fazer configurações de segurança para aplicações Web criadas com ASP.NET.

De maneira alguma temos a pretensão de que este capítulo seja um guia completo para a segurança na Internet e para o Comércio Eletrônico. O objetivo é fornecer as informações básicas para que o usuário possa tomar as medidas mínimas necessárias para garantir um nível aceitável de segurança para seu site e suas aplicações Web.

Quando trabalhamos com tecnologias da Microsoft como ADO>NET, IIS e Windows 2000 Server, o endereço a seguir é de consulta obrigatória para assuntos relacionados à segurança de tecnologias Microsoft: <http://www.microsoft.com/security>.

Neste site são divulgados boletins de segurança sobre os produtos Microsoft. Sempre que algum novo problema é descoberto, são divulgadas informações sobre o problema, bem como a maneira de corrigi-los. Também são disponibilizados arquivos para Download. Estes arquivos normalmente contêm correções (Hot-fix) que devem ser aplicadas para corrigir problemas de segurança.

NOTA: Para acompanhar este capítulo o usuário deve conhecer alguns aspectos básicos do Windows 2000 Server, tais como:

- ◆ Permissões NTFS.
- ◆ Contas de Usuários e Grupos de Usuários.
- ◆ Utilização do MMC – Microsoft Management Console e de Snap-in.

DICA: Para informações sobre estes itens, consulte o livro “Série Curso Básico & Rápido Microsoft Windows 2000 Server” de minha autoria, publicado pela editora Axcel Books.

Autenticação de Usuários com o IIS 5.0

Quando um usuário tenta acessar uma página, a primeira coisa que o servidor precisa determinar é a identidade deste usuário, isto é, o IIS precisa conhecer “quem” está

tentando acessar a página. Uma das maneiras de saber quem é o usuário que está acessando o site é através da utilização de um Username e senha. Porém não seria nada “simpático” apresentar uma tela de logon para o usuário na primeira vez que ele está acessando o site. Até mesmo nas próximas tentativas de acesso, a necessidade de logon pode acabar afastando o internauta.

Através da autenticação do usuário, podem ser definidos os níveis de acesso a informação que o mesmo terá, bem como podem ser feitos registros das ações realizadas pelo usuário, mediante a gravação de logs de acesso. Existem diversos tipos de autenticação possíveis com o IIS. Passaremos a estudá-los individualmente. Os tipos de autenticação que estudaremos são os seguintes:

- ◆ Autenticação anônima.
- ◆ Autenticação básica.
- ◆ Autenticação avançada.
- ◆ Autenticação integrada ao Windows 2000.
- ◆ Autenticação com certificados.

Autenticação Anônima

Um tipo de autenticação bastante comum é o que permite o acesso anônimo. O IIS permite que seja configurado um tipo de acesso chamado Acesso anônimo, no qual não é necessário que o usuário forneça um Username e senha para ter acesso ao site. Este acesso anônimo está ligado a uma única Conta de usuário do Windows 2000 Server. Todo usuário que acessar um site configurado para permitir Acesso anônimo, será identificado como se estivesse autenticado usando a Conta de usuário configurada para o Acesso anônimo.

A conta de usuário para Acesso anônimo é automaticamente criada quando instalamos o Internet Information Services 5.0. Por padrão esta conta possui o seguinte nome:

IUSR_NOME_DO_COMPUTADOR

Por exemplo, ao instalarmos o IIS em um servidor chamado SERVIDOR, será criada a seguinte conta para permitir o Acesso anônimo:

IUSR_SERVIDOR

A autenticação anônima fornece aos usuários acesso a áreas públicas do seu site, sem solicitar um nome de usuário ou uma senha.

Por padrão, a conta IUSR_NOME_DO_COMPUTADOR é incluída no grupo de usuários “Convidados”. Esse grupo tem restrições de segurança, impostas pelas permissões do NTFS (sistema de arquivos do Windows 2000 que possui recursos de segurança mais avançados do que o sistema FAT ou FAT32), que designam o nível de acesso e o tipo de conteúdo disponível para os usuários públicos. Com isso o usuário possui limitações sobre os recursos que ele pode acessar no servidor, sendo que estas limitações já atuam como um nível inicial de segurança.

Se existem vários sites no seu servidor ou áreas no seu site que exigem privilégios de acesso diferentes, você pode criar várias contas anônimas, uma para cada área do site, diretório ou arquivo.

Por exemplo, você pode querer registrar o nível de acesso a diferentes áreas do seu site, utilizando para isso diferentes contas para o acesso anônimo a cada uma destas áreas.

O IIS usa a conta IUSR_NOME_DO_COMPUTADOR da seguinte forma:

1. A conta IUSR_NOME_DO_COMPUTADOR é adicionada ao grupo Convidados no computador ou do Domínio, conforme descrito anteriormente.
2. Quando uma solicitação é recebida, o IIS representa a conta IUSR_NOME_DO_COMPUTADOR antes de executar qualquer código ou acessar qualquer arquivo. O IIS pode representar a conta IUSR_NOME_DO_COMPUTADOR pois conhece o nome de usuário e a senha dessa conta.
3. Antes de retornar uma página ao cliente, o IIS verifica as permissões dos arquivos e diretórios do NTFS para ver se a conta IUSR_NOME_DO_COMPUTADOR tem permissão para acessar o arquivo. Neste ponto é que podemos limitar as áreas às quais o usuário que entra como anônimo tem acesso. Basta configurar as permissões NTFS para que a conta associada ao Acesso anônimo somente tenha acesso às áreas públicas do site.
4. Se o acesso for permitido, a autenticação é concluída e os recursos tornam-se disponíveis para o usuário.
5. Se o acesso não for permitido, o IIS tenta usar outro método de autenticação. Se nenhum método for selecionado, o IIS retorna uma mensagem de erro “HTTP 403 Acesso negado” ao navegador do cliente.

NOTA: Veremos sobre os outros métodos de autenticação ainda neste capítulo.

Na Figura 14.1, temos uma representação desta seqüência para o Acesso anônimo.

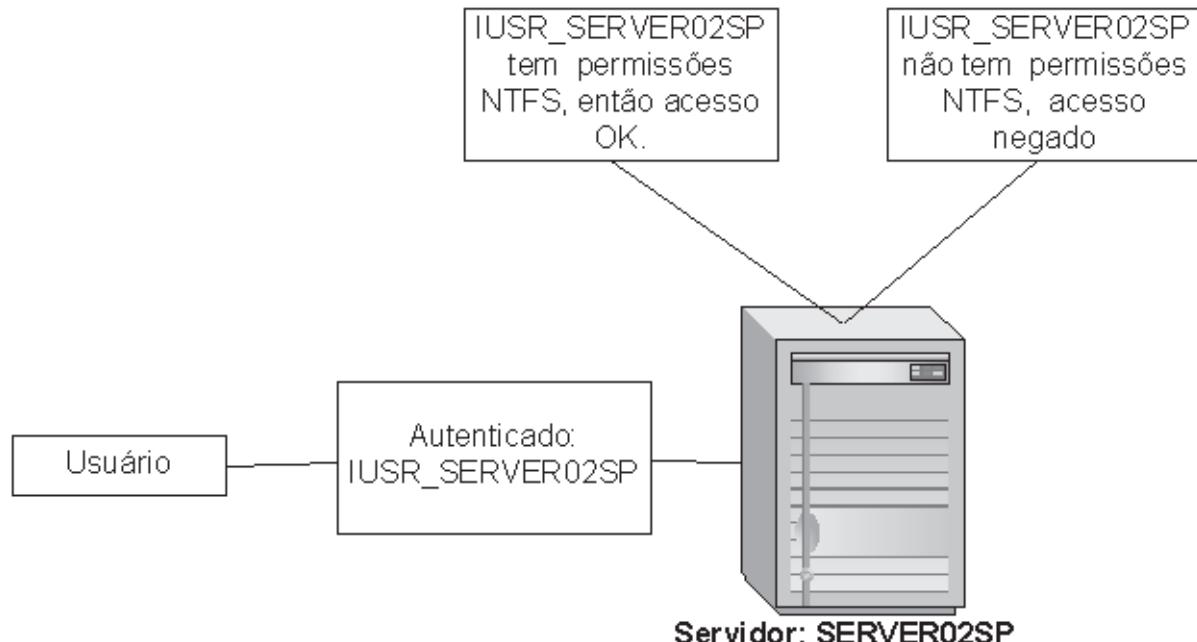


Figura 14.1: Acesso anônimo no IIS.

IMPORTANTE:

- ◆ Se a autenticação anônima for ativada, o IIS tentará sempre a autenticação anônima, usando-a primeiro, mesmo se outros métodos forem ativados.
- ◆ Em alguns casos, o navegador solicitará ao usuário um nome de usuário e uma senha. Você pode alterar a conta usada para a autenticação anônima no Snap-in do IIS, no nível de serviço do servidor Web ou para diretórios virtuais ou arquivos individuais.
- ◆ A conta anônima deve ter o seguinte direito de usuário: "Efetuar logon localmente". Se a conta não tiver a permissão "Efetuar logon localmente", o IIS não poderá atender qualquer solicitação anônima. Ao instalarmos o IIS, automaticamente, a permissão "Efetuar logon localmente" é concedida à conta IUSR_NOME_DO_COMPUTADOR.

Agora vamos aprender a efetuar as seguintes atividades práticas:

- ◆ Definir a conta para acesso anônimo no IIS.
- ◆ Verificando a que grupos pertence a conta IUSR_NOME_DO_SERVIDOR.
- ◆ Verificando a permissão Efetuar logon local para a conta IUSR_NOME_DO_SERVIDOR.
- ◆ Definindo permissões NTFS para uma conta de usuário.

Estas atividades são importantes, não só para as configurações do Acesso anônimo, mas para a configuração de qualquer tipo de acesso.

Como Definir a Conta Para Acesso Anônimo no IIS 5.0

Para definir qual conta será utilizada para o acesso anônimo siga os seguintes passos:

1. Faça o logon no Windows 2000 Server, com permissões de administrador.
2. Abra o Gerenciador do Internet Services: Iniciar -> Programas -> Ferramentas administrativas -> Gerenciador do Internet Services.
3. Surge a janela indicada na Figura 14.2.
4. Esta é a janela do console de administração do IIS 5.0.
5. Dê um clique duplo no nome do computador. No nosso exemplo o nome é servidor (na prática aparece um asterisco ao lado do nome do servidor).
6. Surgem as opções indicadas na Figura 14.3.
7. Podemos configurar o Acesso anônimo para todas as aplicações Web contidas no Servidor ou para cada aplicação individualmente. Inclusive podemos configurar diferentes contas do Windows 2000 Server, para serem utilizadas para o Acesso anônimo em diferentes áreas do site. Podemos configurar uma conta em nível do Servidor Web, de cada aplicação e até mesmo para uma pasta dentro de uma aplicação. Com isso poderíamos ter diferentes pastas,

NOTA: As contas IUSR_NOME_DO_COMPUTADOR em controladores de domínio não são adicionadas ao grupo Convidados do domínio, por padrão, e devem ser alteradas para Efetuar logon localmente a fim de permitir logon anônimo.

DICA: Para informações mais detalhadas sobre estes itens, consulte o livro "Série Curso Básico & Rápido Microsoft Windows 2000 Server" de minha autoria, publicado pela editora Axcel Books.

dentro de uma mesma aplicação Web (que para o IIS é representada por uma pasta virtual, com diferentes contas para acesso anônimo).

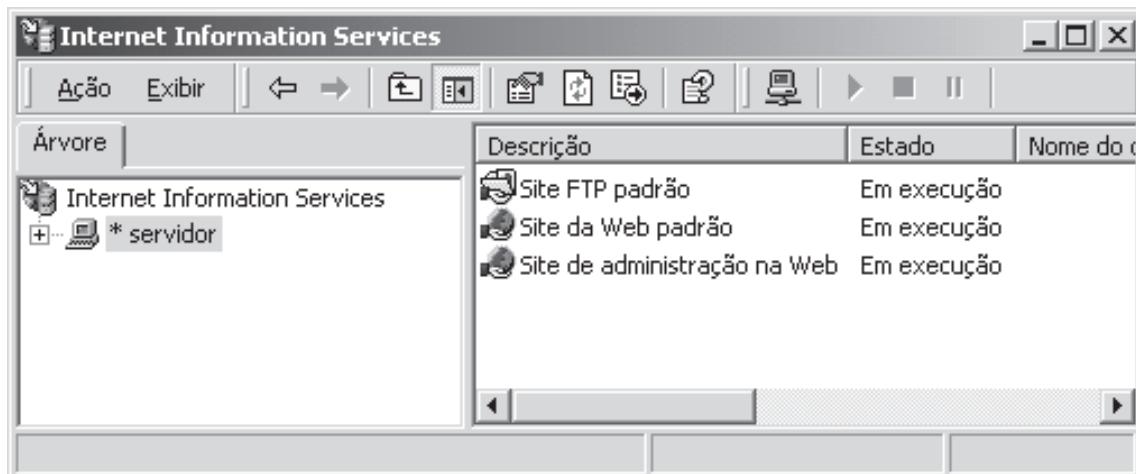


Figura 14.2: O Gerenciador do Internet Services.

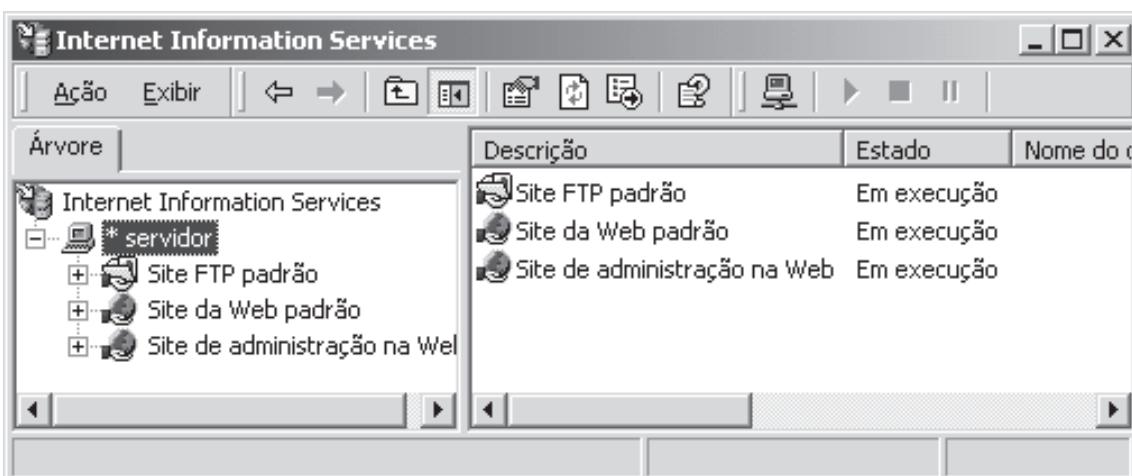


Figura 14.3: Opções de gerenciamento do IIS.

8. No nosso exemplo, iremos configurar uma única conta para Acesso anônimo para todo o servidor. O procedimento é o mesmo quer seja para o site como um todo, para uma aplicação Web do site ou para uma pasta dentro da aplicação Web.
9. Clique com o botão direito do mouse sobre a opção “Site da Web padrão” (ou na opção correspondente, caso você tenha alterado este nome). No menu de opções que surge dê um clique em Propriedades.
10. Será exibida a janela “Propriedades de Site Web padrão”, conforme indicado na Figura 14.4.
11. Dê um clique na guia Segurança de pasta. Serão exibidas as opções indicadas na Figura 14.5.
12. A primeira opção desta guia é Controle de acesso anônimo e autenticação. Dê um clique no botão Editar, ao lado desta opção. Surge a janela Métodos de autenticação, indicada na Figura 14.6.

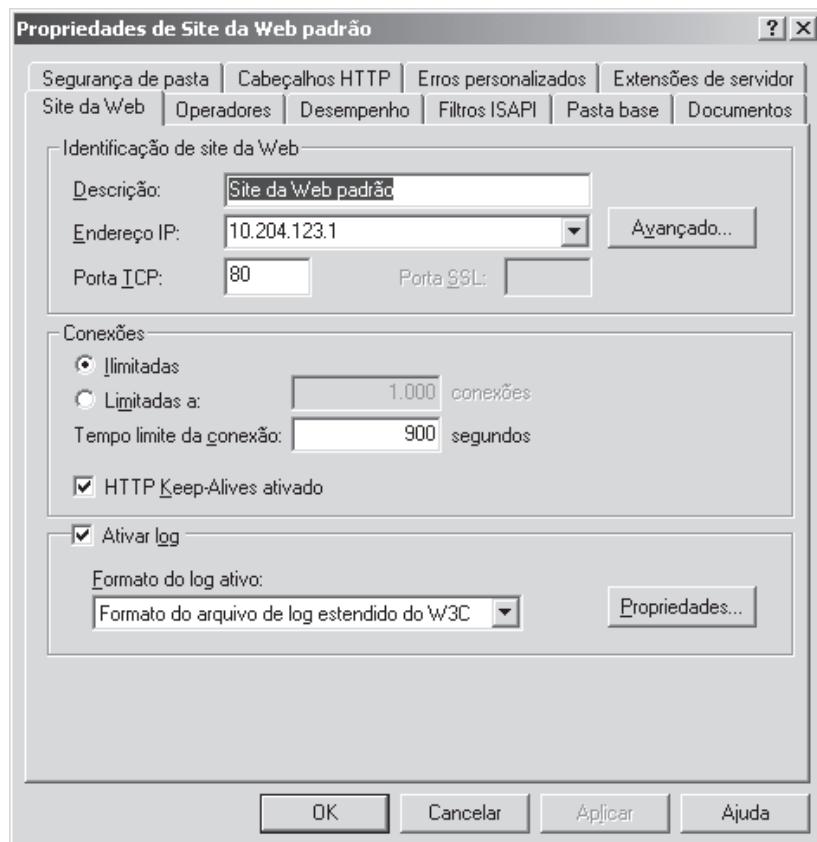


Figura 14.4: Propriedades do Site Web padrão.

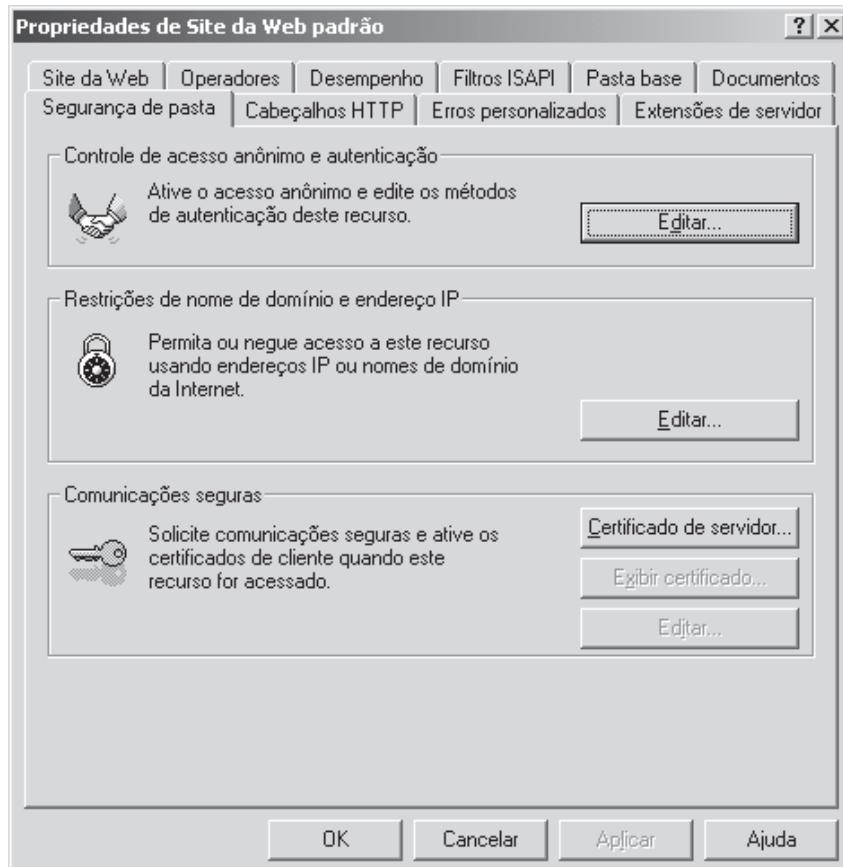


Figura 14.5: As opções da guia Segurança de pasta.

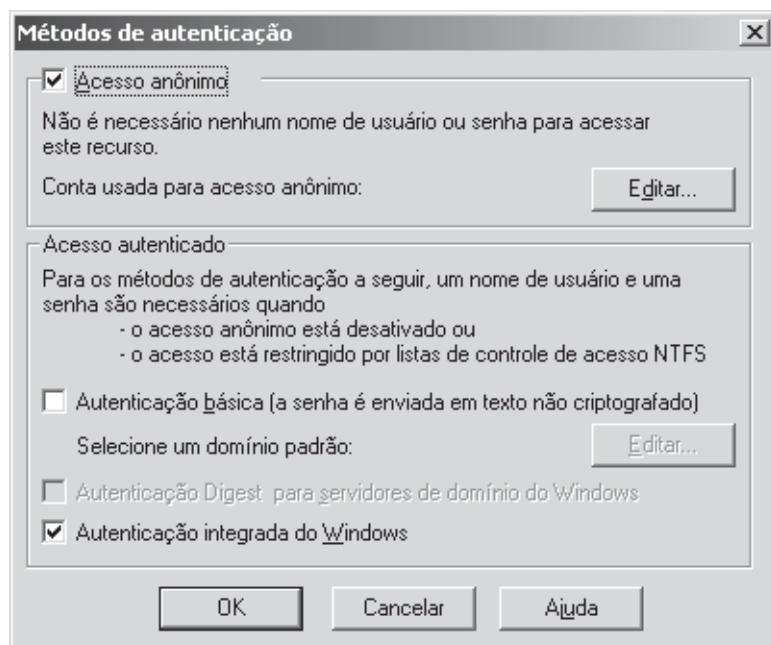


Figura 14.6: As opções para métodos de autenticação.

13. A primeira opção desta janela é Acesso anônimo. Para que o acesso anônimo seja permitido, esta opção deve estar marcada.
14. Para definir a conta que será utilizada para o acesso anônimo, dê um clique no botão Editar, ao lado da opção Acesso anônimo.
15. Será exibida a janela Conta de usuário anônimo, conforme indicado na Figura 14.7.

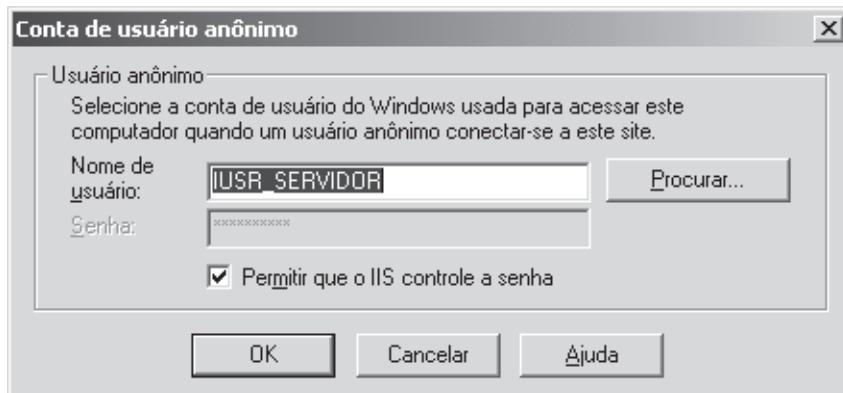


Figura 14.7: Definindo a conta para usuário anônimo.

16. Nesta janela você pode definir a conta que será utilizada para o acesso anônimo. Observe que por padrão é definida a conta IUSR_NOME_DO_COMPUTADOR. No exemplo da Figura 9.7, aparece IUSR_SERVIDOR, pois o computador que estou utilizando para escrever este livro tem o nome de SERVIDOR, conforme já descrito em outras oportunidades.
17. Caso você queira utilizar outra conta e não lembre o nome da mesma, é só clicar no botão Procurar, e será exibida uma lista de usuários cadastrados no Windows 2000.

18. Uma opção interessante a ser comentada é a seguinte:

Permitir que o IIS controle a senha: Esta opção, quando selecionada, permite ao IIS sincronizar automaticamente as configurações de senha anônima com aquelas definidas no Windows 2000. Se a senha fornecida à conta anônima e a senha do Windows para a conta forem diferentes, a autenticação anônima não funcionará. Este é um dos erros mais comuns e a causa mais freqüente de indisponibilidade de um site IIS. Por algum motivo esta opção não está marcada; com isso é preciso digitar a senha para a conta IUSR_NOME_DO_COMPUTADOR. O usuário digita a senha e OK.

Porém mais adiante, por algum motivo ou por solicitação do Administrador do IIS, a senha para esta conta é alterada no Windows 2000. Como a sincronização não está ativada, o IIS continua tentando usar a senha antiga. Como as duas senhas estão diferentes, o acesso é negado e o usuário recebe uma mensagem de acesso negado ao tentar acessar o site.

- 19.** Após ter configurado as informações para a conta de acesso anônimo, dê um clique em OK.
- 20.** Você estará de volta à janela Métodos de autenticação; dê um clique em OK para fechá-la.
- 21.** Você estará de volta à janela Propriedades do site da Web padrão; dê um clique em OK para fechá-la.
- 22.** Você estará de volta ao Gerenciador do Internet Services. Feche-o.

IMPORTANTE: A sincronização de senhas deve ser usada somente com contas de usuário anônimas definidas no computador local e não com contas anônimas de computadores remotos.

Verificando as Configurações da Conta Para Acesso Anônimo no Windows 2000 Server

Conforme descrito anteriormente, a conta IUSR_NOME_DO_COMPUTADOR deve ter algumas configurações especiais (relativas a permissões e direitos), definidas no Windows 2000 Server. Agora veremos como conferir se as configurações para esta conta estão corretas.

Vamos verificar duas configurações a respeito desta conta:

- ◆ A que grupos de usuários do Windows 2000 pertence esta conta.
- ◆ Se a conta possui a permissão “Fazer logon localmente”.

Para verificar a que grupos pertence a conta IUSR_NOME_DO_COMPUTADOR:

1. Faça o logon no Windows 2000 Server, com permissões de administrador.
2. Abra o Console para Gerenciamento do computador: Iniciar -> Programas -> Ferramentas administrativas -> Gerenciador do Computador.

NOTA: Para definir uma conta de acesso anônimo diferente para uma das aplicações Web do site, ou até mesmo para uma subpasta de uma aplicação Web, basta repetir os passos indicados.

NOTA: Se o servidor que você estiver utilizando for um controlador de domínio, você deve abrir o Console para gerenciamento do Active Directory. As opções que surgem podem ser um pouco diferentes das apresentadas neste passo-a-passo.

3. Surge a janela indicada na Figura 14.8.

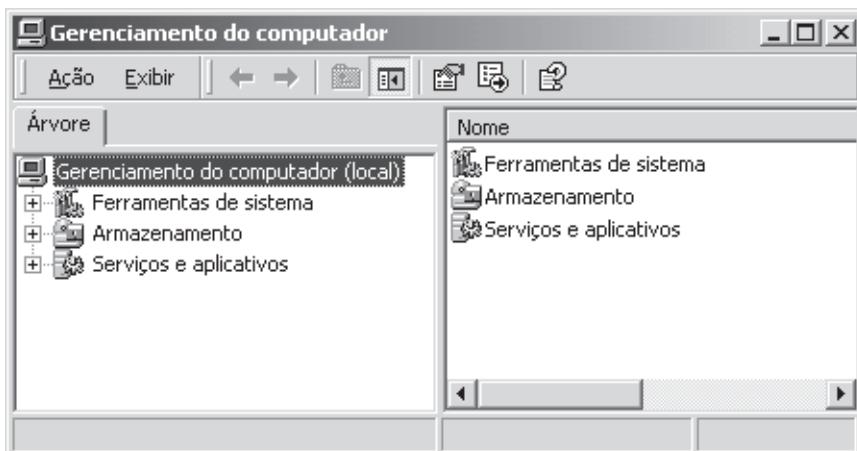


Figura 14.8: O console para Gerenciamento do computador.

4. Dê um clique no sinal de + ao lado da opção Ferramentas de sistema, para abri-la.
5. Nas opções que surgem, abaixo de Ferramentas de sistema, dê um clique no sinal de + ao lado da opção Usuários e grupos locais para abri-la.
6. Surgem as opções indicadas na Figura 14.9.

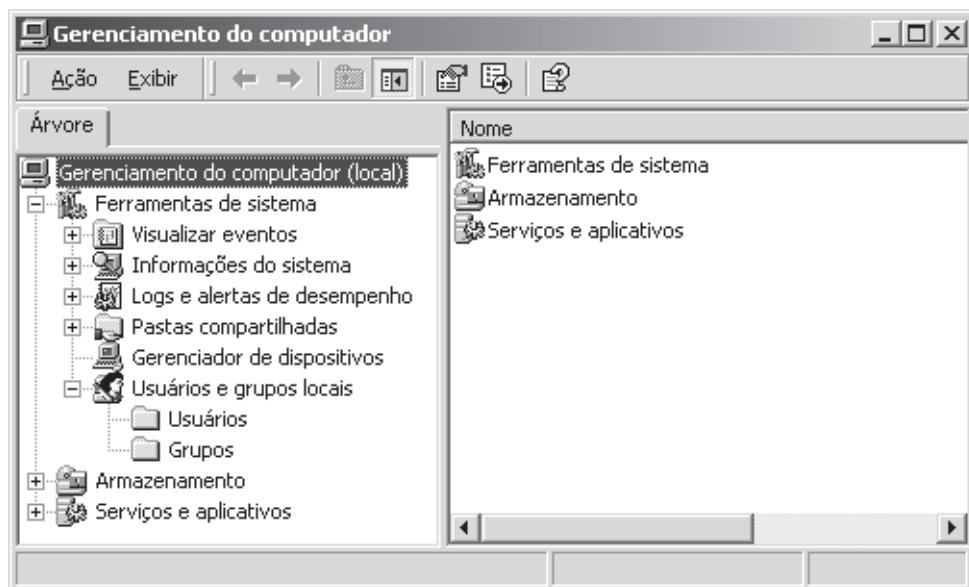


Figura 14.9: Informações sobre os usuários e grupos de usuários do Windows 2000.

7. Dê um clique na opção Usuários. Surge, no painel da direita, uma listagem com o nome de todos os usuários, conforme indicado na Figura 14.10.

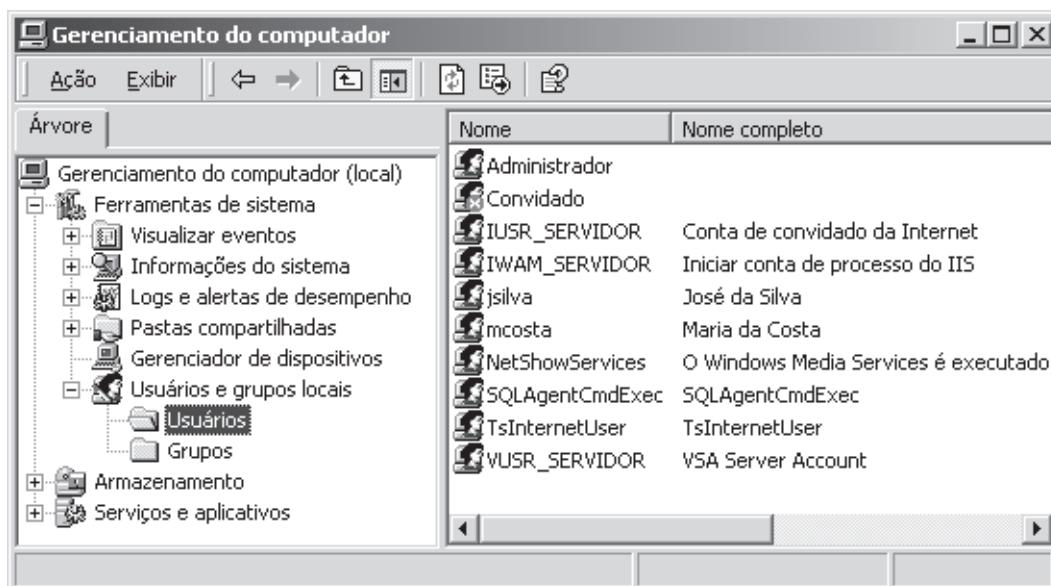


Figura 14.10: Listagem dos usuários cadastrados.

8. Na lista de usuários, do painel da direita, localize o usuário IUSR_NOME_DO_COMPUTADOR e dê um clique duplo sobre o mesmo para abrir a janela de propriedades do usuário, conforme indicado na Figura 14.11. No nosso exemplo é o usuário IUSR_SERVIDOR, pois, conforme descrito anteriormente, o computador que estou utilizando é chamado SERVIDOR.

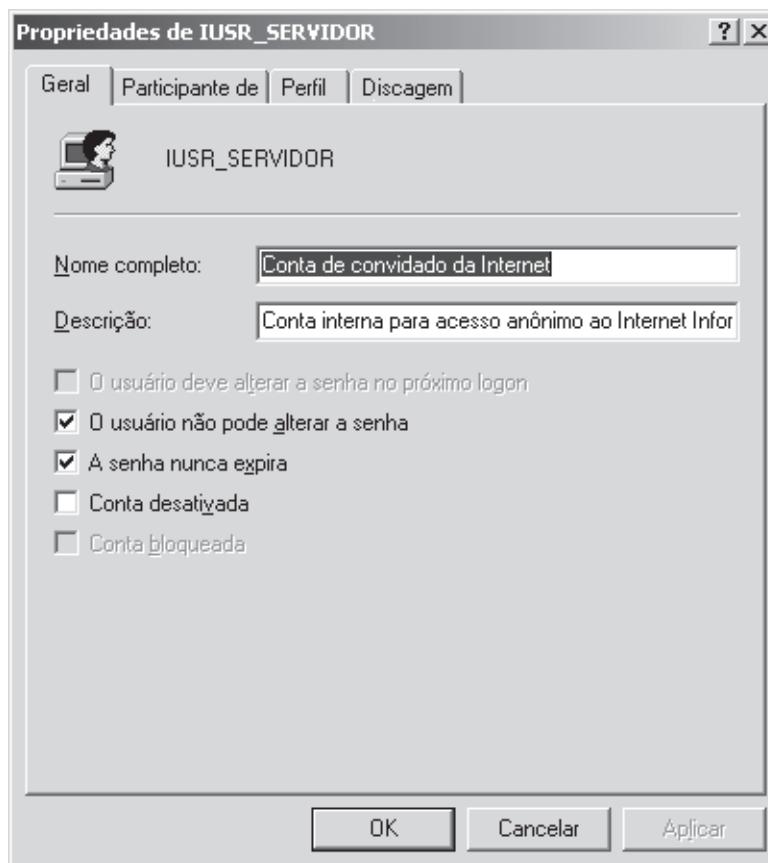


Figura 14.11: Propriedades para o usuário IUSR_SERVIDOR.

9. Para saber a quais grupos o usuário pertence, dê um clique na guia Participante de (ou Membro de, se for um Controlador de Domínio). Podemos conferir que o usuário IUSR_SERVIDOR somente pertence ao grupo Convidados. Se estivéssemos em um Servidor que atua como Controlador de domínio, teríamos o grupo Convidados do domínio.
10. Quando o IIS é instalado, a conta IUSR_SERVIDOR é criada e automaticamente adicionada ao grupo Convidados.
11. Muito cuidado ao adicionar a conta IUSR_SERVIDOR a outros grupos. Quando uma conta de usuário é adicionada a um grupo, ele herda as permissões atribuídas ao grupo e com isso passa a ter acesso aos recursos a que o grupo tem acesso. A maior “insanidade” que o administrador Web poderia cometer seria adicionar a conta IUSR_SERVIDOR ao grupo Administradores. Com isso estaria dando permissões máximas aos usuários que fazem acesso anônimo. Seria caso de internação do seu administrador Web.
12. Clique no botão OK para fechar a janela com as propriedades da conta IUSR_SERVIDOR.
13. Você estará de volta ao Gerenciador do computador; feche-o.

DICA: Para informações mais detalhadas sobre Controladores de domínio e Active Directory, consulte o livro “Série Curso Básico & Rápido Microsoft Windows 2000 Server”, de minha autoria, publicado pela editora Axcel Books.

Agora precisamos verificar se a conta IUSR_SERVIDOR tem a permissão para Efetuar logon local.

Para verificar se a conta IUSR_SERVIDOR tem a permissão para Efetuar logon local, faça o seguinte:

1. Faça o logon no Windows 2000 Server, com permissões de administrador.
2. Abra o Console “Configurações locais de segurança”: Programas -> Ferramentas administrativas -> Diretivas de segurança local.
3. Surge a janela indicada na Figura 14.12.

NOTA: Se o servidor que você estiver utilizando for um controlador de domínio, você deve abrir o Console para gerenciamento do Active Directory. As opções que surgem podem ser um pouco diferentes das apresentadas neste passo-a-passo.

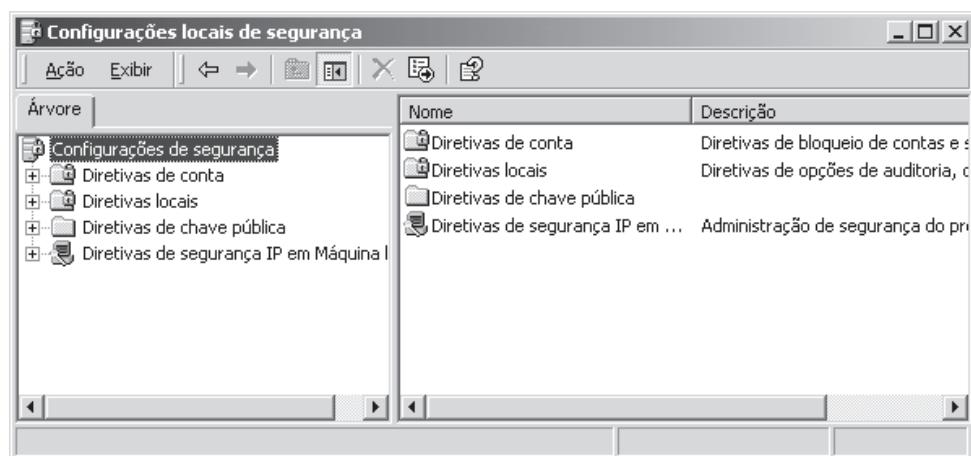


Figura 14.12: Configurações locais de segurança.

4. Dê um clique no sinal de + ao lado da opção Diretivas locais, para abri-la.

5. Nas opções que surgem, abaixo de Diretivas locais, dê um clique no sinal de + ao lado da opção Atribuição de direitos de usuário, para abri-la.
6. Surgem as opções indicadas na Figura 14.13.

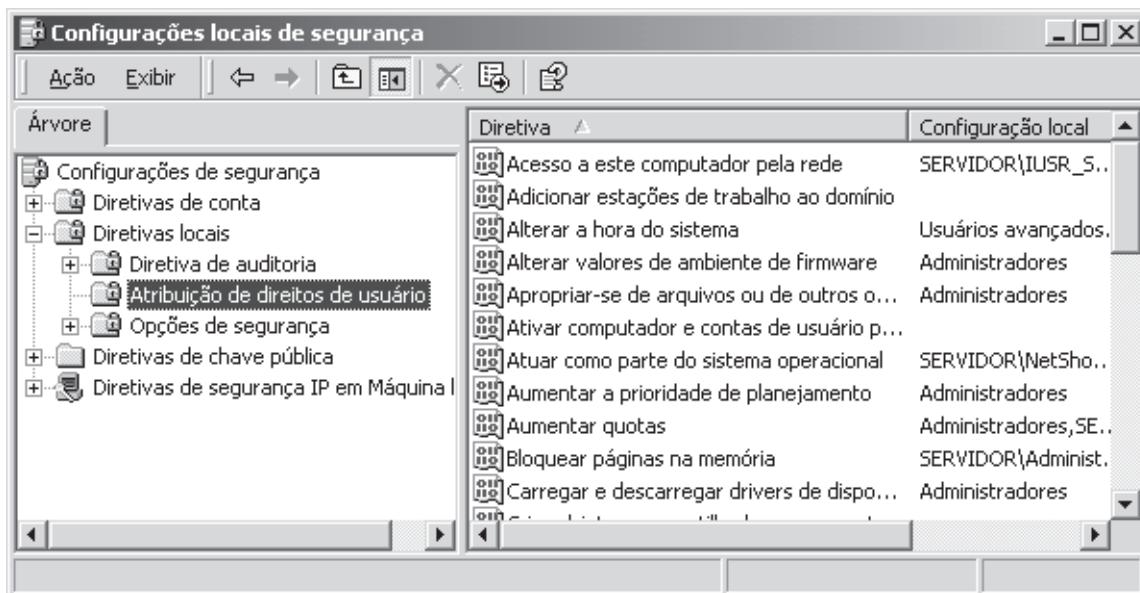


Figura 14.13: Atribuições de direitos para contas e grupos de usuários.

7. No painel da direita surgem as diversas permissões disponíveis.
8. Na listagem de permissões, localize Efetuar logon local, conforme indicado na Figura 14.14.

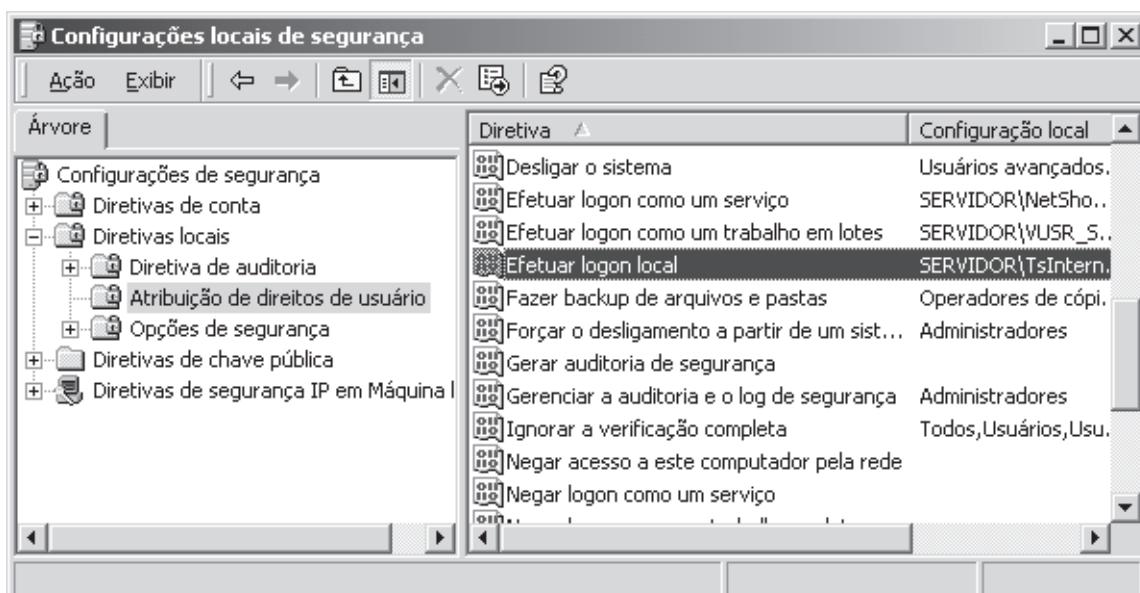


Figura 14.14: A permissão Efetuar logon local.

9. Dê um clique duplo sobre a permissão Efetuar logon local, para exibir a janela de configurações para esta Diretiva de segurança local. Nesta janela surge uma lista dos usuários que possuem a permissão de Efetuar logon local, conforme indicado na Figura 14.15. O usuário IUSR_SERVIDOR deve fazer parte da lista.

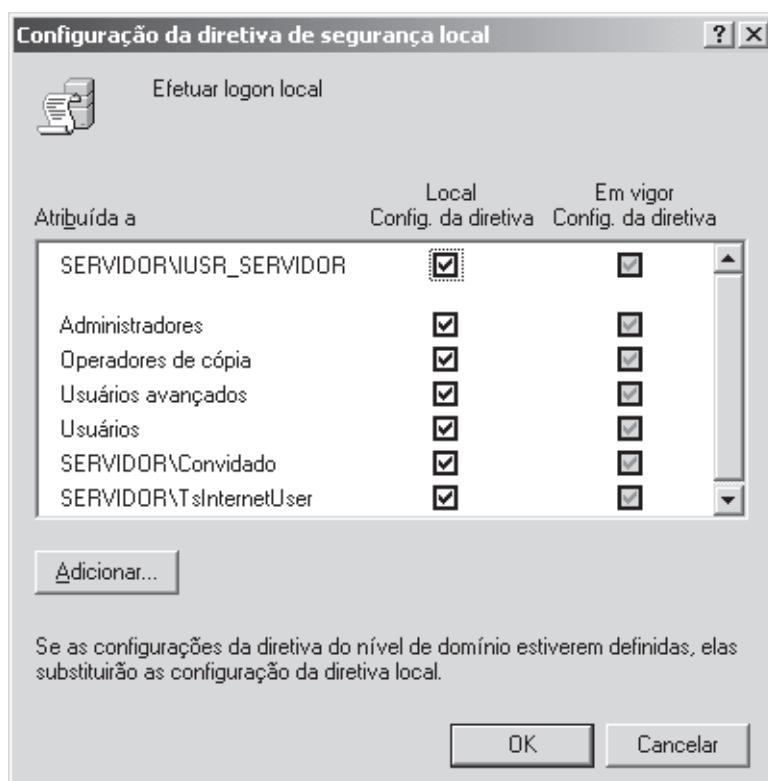


Figura 14.15: Lista de usuários com direito a Efetuar logon local.

10. Caso o usuário IUSR_SERVIDOR não estivesse na lista, você poderia adicioná-lo utilizando o botão Adicionar.
11. A permissão Efetuar logon local autoriza o usuário a fazer o logon no console do Servidor, isto é, localmente no servidor. A conta utilizada para acesso anônimo precisa desta permissão, pois caso contrário o Acesso anônimo não irá funcionar.
12. Dê um clique no botão OK para voltar ao console “Configurações locais de segurança”.
13. Feche-o.

Com isso já sabemos que a conta para acesso anônimo está configurada corretamente. No próximo item vamos retirar as permissões NTFS da conta de acesso anônimo de uma das aplicações Web do servidor. Vamos tentar acessar esta aplicação e observar os resultados obtidos.

Configurando Permissões NTFS em Pastas do Servidor Web

Uma das medidas básicas de segurança é a utilização de drives formatados com o sistema de arquivos NTFS, ao invés de FAT e FAT32. O motivo é bastante simples, pois, através da utilização do sistema de arquivos NTFS, podemos ter um controle bastante refinado sobre o acesso às informações, mediante a atribuição de permissões de pasta e de arquivos.

Quando estamos tratando de permissões NTFS, já estamos em um nível de segurança que é gerenciado pelo Sistema Operacional. É através da utilização do Windows 2000 Server que definimos permissões NTFS. Isso reforça o fato de que a segurança é tanto responsabilidade do grupo de Desenvolvimento, quanto do Administrador da rede.

No exemplo do Acesso anônimo, o usuário é identificado para o Windows 2000, como se fosse o usuário IUSR_NOME_DA_MAQUINA. Este usuário somente terá acesso às pastas e arquivos para os quais o usuário IUSR_NOME_DA_MAQUINA tiver permissões de acesso e com os níveis de permissões definidos.

Antes de aprendermos a definir permissões NTFS vamos aprender um pouco mais sobre as mesmas.

Sistemas de Arquivos no Windows 2000 e Permissões NTFS

Agora vamos ver alguns detalhes sobre os sistemas de arquivos que o Windows 2000 Server reconhece e também sobre permissões NTFS.

Um sistema de arquivos determina a maneira como o Windows 2000 Server organiza e recupera as informações no disco rígido ou em outros tipos de mídia. O Windows 2000 Server reconhece os seguintes sistemas de arquivos:

- ◆ FAT
- ◆ FAT32
- ◆ NTFS
- ◆ NTFS 5

O sistema FAT vem desde a época do DOS e tem sido mantido por questões de compatibilidade. Além disso, se você tiver instalado mais de um Sistema Operacional no seu computador, alguns sistemas mais antigos (DOS, Windows 3.x e as primeiras versões do Windows 95) somente reconhecem o sistema FAT. Com o sistema de arquivos FAT, a única maneira de restringir o acesso ao conteúdo de uma pasta compartilhada é através das permissões de compartilhamento, as quais não têm efeito no caso de acessos pela Internet, através do IIS. Com a utilização do sistema FAT, alguns recursos avançados, tais como compressão, criptografia e auditoria, não estão disponíveis.

O sistema FAT32 apresenta algumas melhorias em relação ao sistema FAT. Existe um melhor aproveitamento do espaço em disco, e, com isso, um menor desperdício. Um grande inconveniente do sistema FAT32 é que ele não é reconhecido pelo Windows NT Server 4.0. Com o sistema de arquivos FAT32, a única maneira de restringir o acesso ao conteúdo de uma pasta compartilhada é através das permissões de compartilhamento. Com a utilização do sistema FAT32, alguns recursos avançados, tais como compressão, criptografia e auditoria, não estão disponíveis.

O sistema de arquivos NTFS é utilizado no Windows NT Server 4.0 e foi mantido e melhorado no Windows 2000 Server, por questões de compatibilidade, já que uma nova versão do NTFS foi introduzida com o Windows 2000 – NTFS 5. É um sistema bem mais eficiente do que FAT e FAT32, além de permitir uma série de recursos avançados, tais como:

- ◆ Permissões em nível de arquivos e pastas
- ◆ Compressão
- ◆ Auditoria de acesso
- ◆ Partições bem maiores do que as permitidas com FAT e FAT32
- ◆ Desempenho bem superior do que com FAT e FAT32

Uma das principais vantagens do NTFS é que o mesmo permite que sejam definidas permissões de acesso em nível de arquivo e de pastas, isto é, posso ter arquivos em uma mesma pasta, com permissões diferentes para usuários diferentes. Além disso, as permissões NTFS têm efeito localmente, isto é, mesmo que o usuário faça o logon no computador onde um determinado arquivo está gravado, se o usuário não tiver as permissões NTFS necessárias, ele não poderá acessar o arquivo. Isso confere um alto grau de segurança, desde que as permissões NTFS sejam configuradas corretamente.

No Windows 2000 Server, conforme descrito anteriormente, temos também o NTFS 5, o qual apresenta diversas melhorias em relação ao NTFS, tais como:

- ◆ Criptografia de arquivos e pastas (A criptografia é uma maneira de “embaralhar” a informação de tal forma que, mesmo que um arquivo seja copiado, o mesmo se torna ininteligível, a não ser para a pessoa que possui a “chave” para descriptografar o arquivo).
- ◆ Cotas de usuário: faz com que seja possível limitar o espaço em disco que cada usuário pode utilizar.
- ◆ Gerenciamento e otimização melhorados.

Um inconveniente do NTFS 5 é que ele não é reconhecido pelas versões anteriores, tais como o Windows NT Server 4.0 (somente com Service Pack 4.0 ou superior). Caso você possua uma rede na qual estão presentes servidores com o Windows 2000 Server e com o Windows NT Server 4.0, planeje com bastante cuidado a utilização do NTFS 5.

Conforme descrito anteriormente, podemos definir permissões de acesso em nível da pasta ou arquivo, mas somente em unidades formatadas com o sistema de arquivos NTFS (seja na versão do NT Server 4.0 ou o NTFS 5 do Windows 2000 Server). Por isso que é aconselhável instalar o Windows 2000 Server sempre em unidades formatadas com NTFS, pois isso melhora a segurança.

Com relação às permissões NTFS, temos um conjunto diferente de permissões quando tratamos de pastas ou arquivos. Nas Tabelas 14.1(para pastas) e 14.2 (para arquivos), são apresentadas as permissões e o nível de acesso para cada uma delas.

Tabela 14.1 Permissões NTFS para pastas.

Permissão	Nível de acesso
Leitura	Permite ao usuário listar as pastas e arquivos dentro da pasta, permite que sejam exibidas as permissões, donos e atributos.
Gravar	Permite ao usuário criar novos arquivos e subpastas dentro da pasta, alterar os atributos da pasta e visualizar o dono e as permissões da pasta.
Listar Conteúdo de pastas	Permite ao usuário ver o nome dos arquivos e subpastas.
Ler e executar	Permite ao usuário navegar através das subpastas para chegar a outras pastas e arquivos, mesmo que o usuário não tenha permissão de acesso às pastas pelas quais está navegando; além disso possui os mesmos direitos que as permissões Leitura e Listar Conteúdo de pastas.

Permissão	Nível de acesso
Modificar	Permite ao usuário eliminar a pasta, mais todas as ações permitidas pela permissão Gravar e pela permissão Ler e executar.
Controle total	Permite que sejam alteradas as permissões, permite ao usuário tornar-se dono da pasta, eliminar subpastas e arquivos, mais todas as ações permitidas por todas as outras permissões NTFS.

Tabela 14.2 Permissões NTFS para arquivos.

Permissão	Nível de acesso
Leitura	Permite ao usuário ler o arquivo, permite que sejam exibidas as permissões, donos e atributos.
Gravar	Permite ao usuário gravar um arquivo com o mesmo nome sobre o arquivo, alterar os atributos da pasta e visualizar o dono e as permissões da pasta.
Ler e executar	Permite ao usuário executar aplicativos (normalmente programas .exe, .bat ou .com), mais todos os direitos da permissão Leitura.
Modificar	Permite ao usuário modificar e eliminar o arquivo, mais todas as ações permitidas pela permissão Gravar e pela permissão Ler e executar.
Controle total	Permite que sejam alteradas as permissões, permite ao usuário tornar-se dono do arquivo, mais todas as ações permitidas por todas as outras permissões NTFS.

Todo arquivo ou pasta em uma unidade formatada com NTFS possui uma “Lista de controle de acesso” (Access Control List) – ACL. Nesta ACL fica uma lista de todas as contas de usuários e grupos para os quais foi garantido acesso para o recurso, bem como o nível de acesso de cada um deles.

Existem alguns detalhes que devemos observar sobre permissões NTFS:

- ◆ Permissões NTFS são cumulativas, isto é, se um usuário pertence a mais de um grupo, o qual tem diferentes níveis de permissão para um recurso, a permissão efetiva do usuário é a soma das permissões.
- ◆ Permissões NTFS para um arquivo têm prioridade sobre permissões NTFS para pastas. Por exemplo se um usuário tem permissão NTFS de escrita em uma pasta, mas somente permissão NTFS de leitura para um arquivo dentro desta pasta, a sua permissão efetiva será somente a de leitura, pois a permissão para o arquivo tem prioridade sobre a permissão para a pasta.
- ◆ Negar uma permissão NTFS tem prioridade sobre permitir. Por exemplo, se um usuário pertence a dois grupos diferentes. Para um dos grupos foi dada permissão de leitura para um arquivo, e para o outro grupo foi negada a permissão de leitura; o usuário não terá o direito de leitura, pois Negar tem prioridade sobre Permitir.

Agora que já conhecemos um pouco mais sobre permissões NTFS, podemos aprender como configurar estas permissões.

Definindo Permissões NTFS

Neste exemplo prático vamos fazer o seguinte:

- ◆ Vamos retirar as permissões NTFS do usuário IUSR_SERVIDOR da pasta de um aplicativo Web.
- ◆ Vamos tentar acessar o aplicativo e observar a mensagem de erro que recebemos.
- ◆ Vamos restaurar as permissões originais e tentar acessar a página novamente.

Para acessar as permissões NTFS de uma pasta e retirar as permissões do usuário IUSR_SERVIDOR, faça o seguinte:

1. Faça o logon com privilégios de Administrador.
2. Utilizando o Windows Explorer, localize a pasta cujas permissões NTFS serão alteradas.
3. Dê um clique com o botão direito do mouse na pasta. No menu de opções que surge dê um clique em Propriedades.
4. Surge a janela indicada na Figura 14.16.

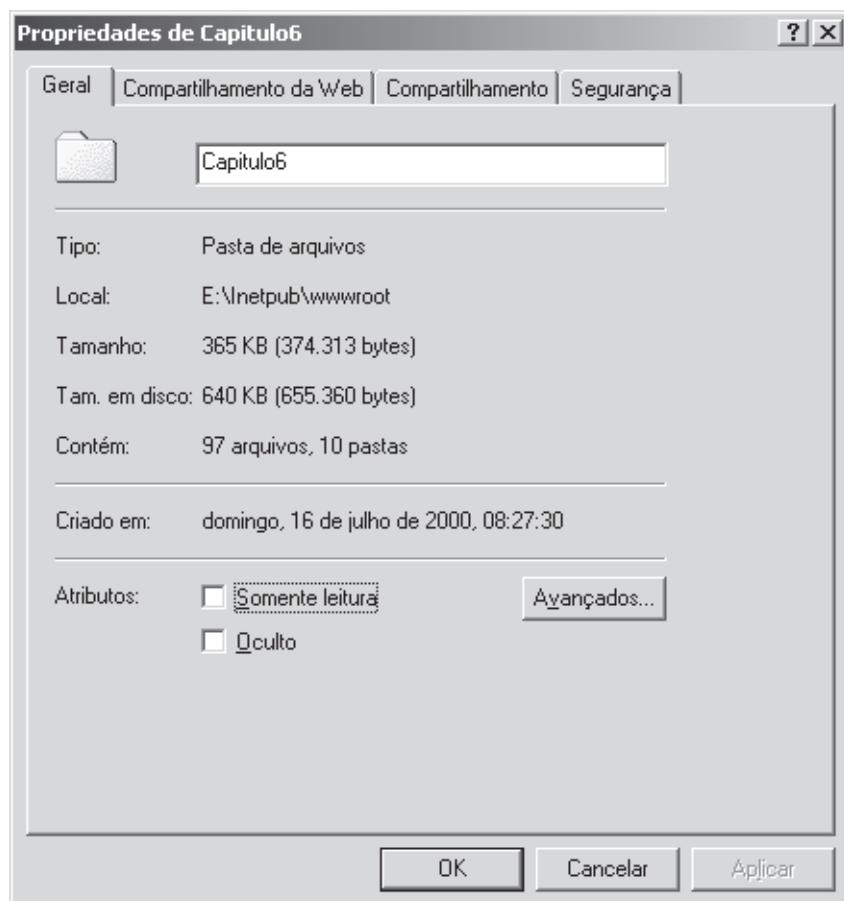


Figura 14.16: Janela com as propriedades da pasta.

5. Dê um clique na guia Segurança.

6. Surge a janela indicada na Figura 14.17.

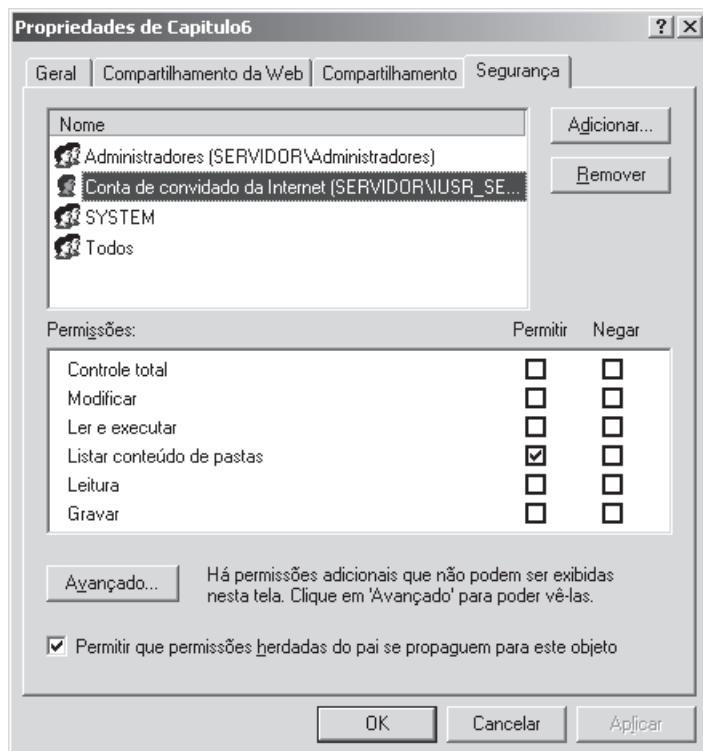


Figura 14.17: As configurações de segurança para a pasta selecionada.

7. Observe que a conta com a descrição “Conta de convidado da Internet” é a conta para acesso anônimo, que no nosso exemplo é IUSR_SERVIDOR. Somente possuem permissão de acesso as contas que fazem parte desta lista.
 8. Dê um clique no botão Avançado. Surge a janela indicada na Figura 14.18.

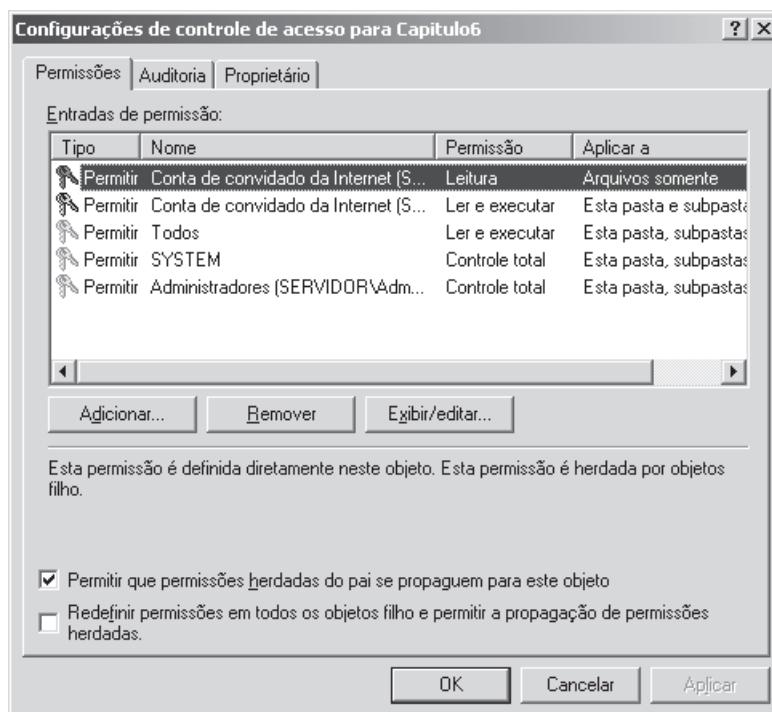


Figura 14.18: A conta IUSR_SERVIDOR possui permissão somente para leitura.

9. Observe que a conta para acesso anônimo possui permissão somente de leitura.
10. Vamos remover as permissões da conta IUSR_SERVIDOR (Conta de convidado da Internet). Para isso dê um clique sobre a conta para marcá-la, e depois dê um clique no botão Remover.
11. A conta IUSR_SERVIDOR não aparece mais na listagem.
12. Marque a opção “Redefinir opções em todos os objetos filhos e permitir a propagação das permissões herdadas.” Esta opção deve ser marcada para que as alterações que estão sendo feitas na pasta sejam propagadas para todas as subpastas e arquivos pertencentes a esta pasta. Esta modificação é necessária para que a permissão para o usuário IUSR_SERVIDOR seja retirada de todos os arquivos pertencentes à pasta que está sendo alterada.
13. Dê um clique no botão OK para voltar à janela de propriedades da pasta.
14. Surge uma janela pedindo confirmação. Dê um clique em Sim para continuar.
15. Você estará de volta à janela de propriedades da pasta.
16. Na lista de usuários, remova os usuários IUSR_SERVIDOR e o grupo Todos. Para isso basta clicar no nome do usuário ou grupo e depois clicar no botão Remover.
17. A sua lista de permissões deve estar semelhante à indicada na Figura 14.19.

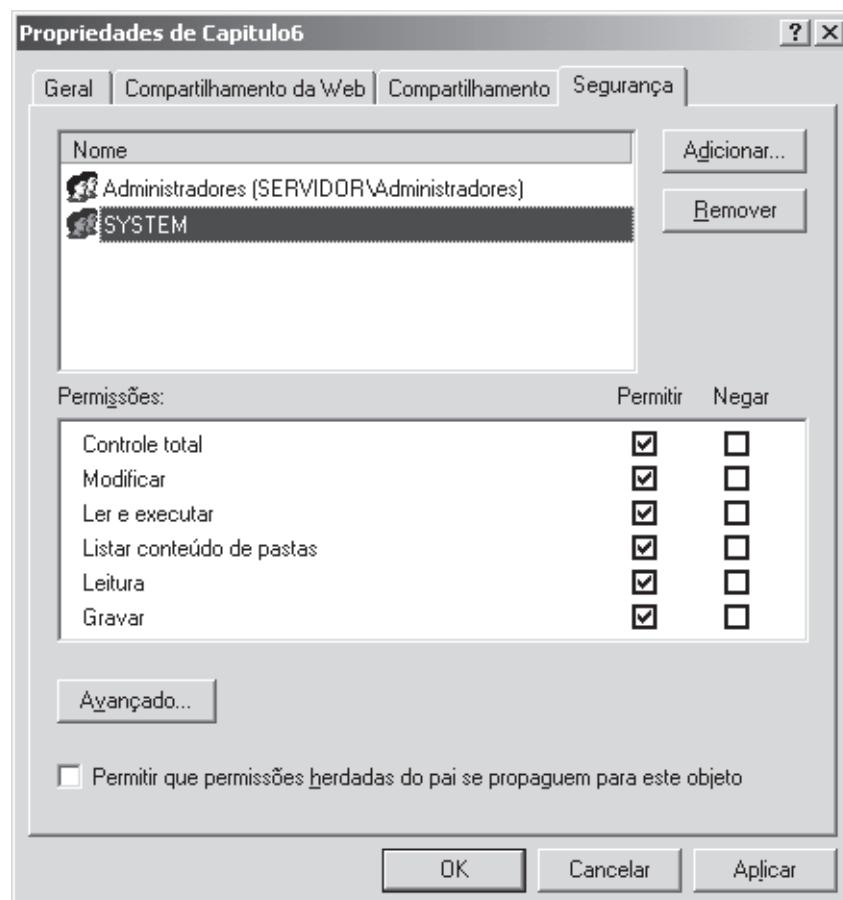


Figura 14.19: Lista de permissões, onde foi retirada a permissão de acesso para o usuário anônimo.

18. Dê um clique no botão OK para fechar esta janela.

Para testar se o acesso realmente foi retirado.

Agora que retiramos as permissões do usuário anônimo, se alguém tentar acessar algum arquivo que está na pasta cujas permissões foram retiradas, irá receber uma mensagem de erro, conforme indicado na Figura 14.20.

Veja que a mensagem informa que o acesso à página solicitada foi negado. Isto acontece porque o usuário IUSR_SERVIDOR não possui as permissões NTFS necessárias.

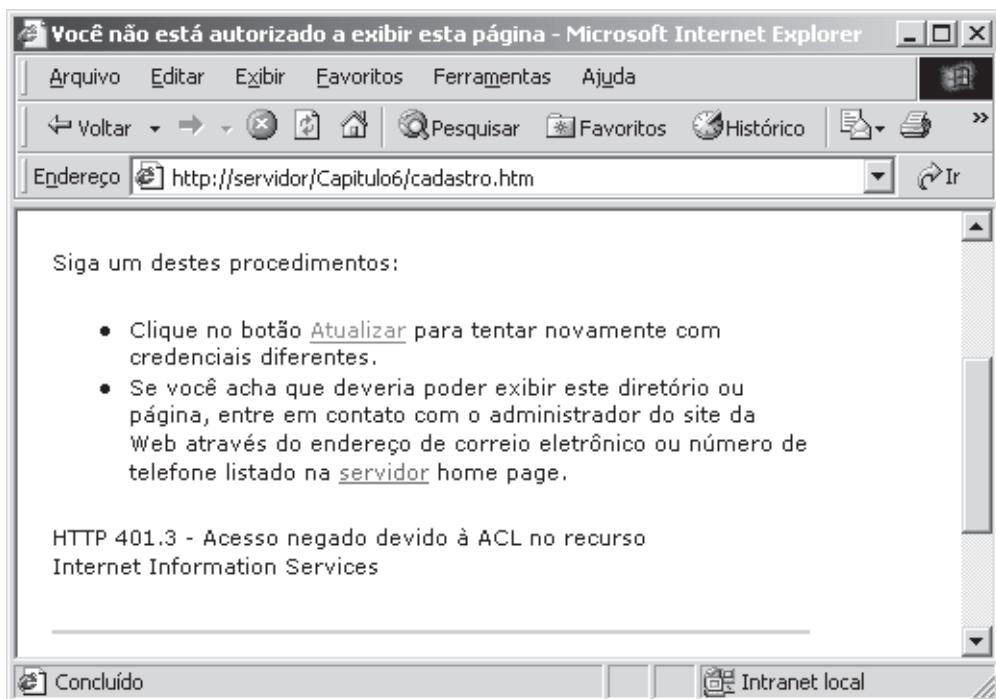


Figura 14.20: Mensagem de erro ao tentar acessar uma página para a qual o usuário anônimo (conta IUSR_SERVIDOR no nosso exemplo) não tem mais permissão de acesso.

Caso você teste o acesso localmente no servidor onde a página está gravada e a autenticação integrada esteja habilitada, você terá acesso à página. Isto acontece porque primeiro o IIS tenta acesso com a autenticação anônima. Não obtém sucesso. Se a autenticação integrada (a autenticação integrada utiliza a conta do Windows que você utilizou para fazer o logon) estiver habilitada, o IIS tenta utilizá-la. Caso a conta que você utilizou para fazer o logon tenha permissão de acesso à página, o IIS libera o acesso. Isto também é válido para usuários da sua rede local que fizeram o logon em um domínio do Windows NT Server 4.0 ou do Windows 2000 e cujas contas ou grupos a que pertencem possuam permissão de acesso para o arquivo solicitado. Nos próximos itens veremos mais sobre a autenticação integrada.

Agora vamos restaurar as permissões NTFS para o usuário IUSR_SERVIDOR e testar para ver se ele voltou a ter acesso.

Para atribuir novamente as permissões NTFS para o usuário IUSR_SERVIDOR.

1. Utilizando o Windows Explorer, localize a pasta cujas permissões NTFS serão alteradas.
2. Dê um clique com o botão direito do mouse sobre a pasta. No menu de opções que surge dê um clique em Propriedades.
3. Na janela de propriedades dê um clique na guia Segurança.

4. Surge a janela indicada na Figura 14.21.

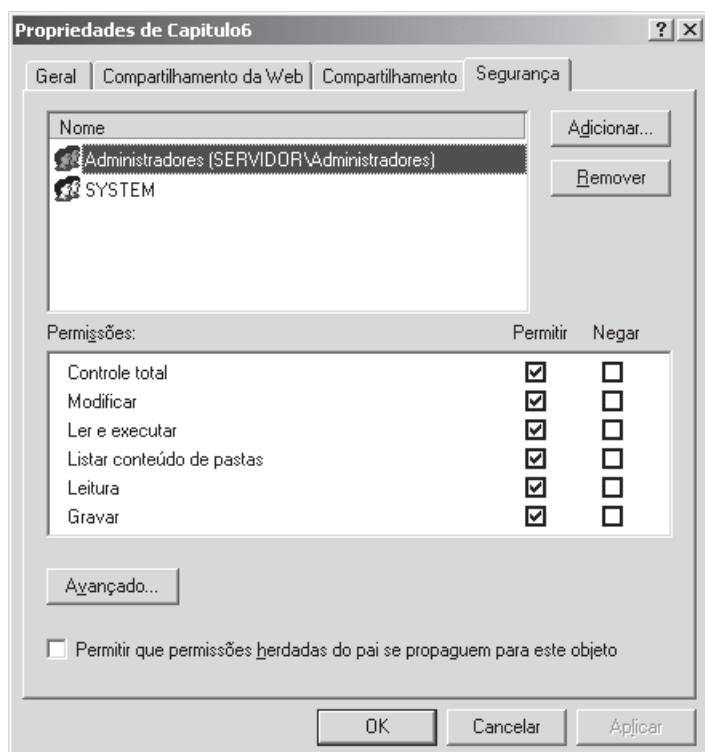


Figura 14.21: As configurações de segurança para a pasta selecionada.

5. Observe que a conta com a descrição “Conta de convidado da Internet” é a conta para acesso anônimo, que no nosso exemplo é IUSR_SERVIDOR. e que esta conta não aparece na lista de usuários; portanto a mesma não possui permissões de acesso.
6. Dê um clique no botão Avançado. Observe que a janela em que surge a conta IUSR_SERVIDOR também não faz parte da listagem, conforme indicado na Figura 14.22.

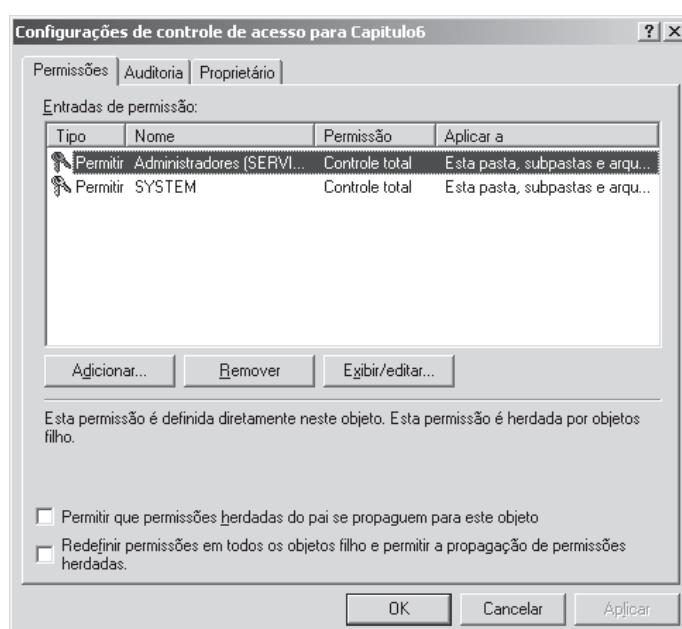


Figura 14.22: A conta IUSR_SERVIDOR não aparece na lista de contas.

7. Dê um clique no botão Adicionar. Surge uma janela com a listagem de usuários. Localize o usuário IUSR_SERVIDOR e dê um clique sobre o mesmo para marcá-lo, conforme indicado na Figura 14.23.

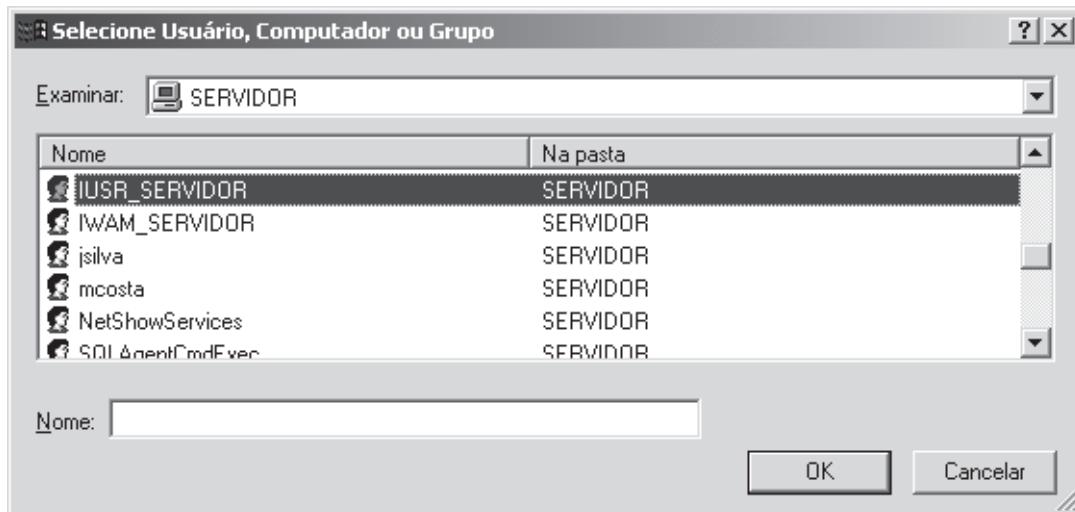


Figura 14.23: Adicionando novamente o usuário IUSR_SERVIDOR.

8. Dê um clique no botão OK. Surge uma janela pedindo para que você defina as permissões NTFS para o usuário IUSR_SERVIDOR.
 9. Defina as permissões conforme indicado na Figura 14.24 e dê um clique no botão OK.

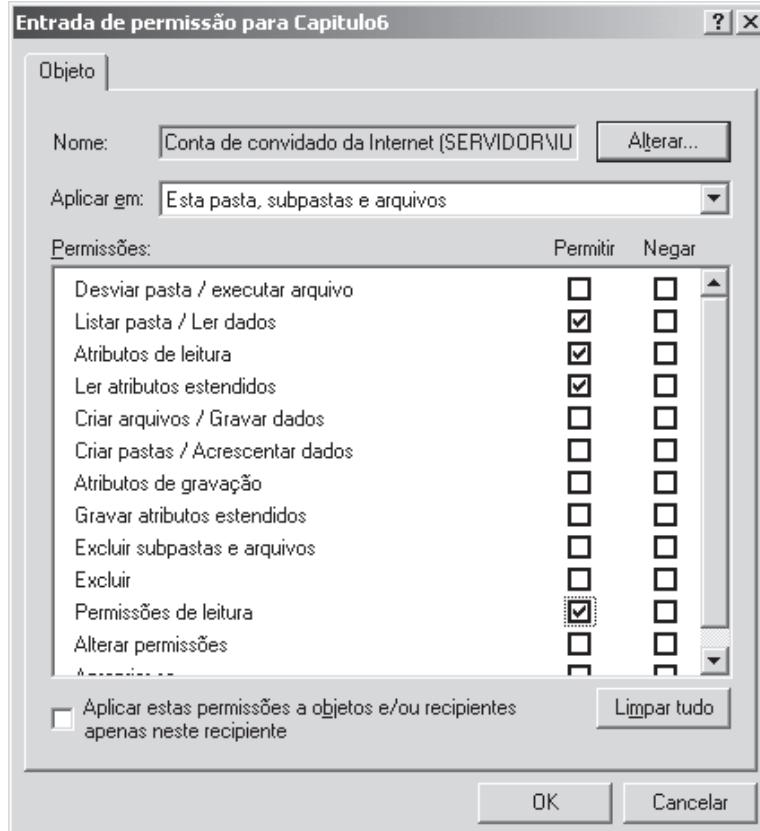


Figura 14.24: Restaurando as permissões para o usuário IUSR_SERVIDOR.

10. Você estará de volta à janela de opções avançadas. Certifique-se de que a opção “Redefinir permissões em todos os objetos filho e permitir a propagação das permissões herdadas”, esteja marcada.
11. Dê um clique no botão OK para voltar à janela de propriedades da pasta.
12. Surge uma janela pedindo confirmação, conforme indicado na Figura 14.25. Dê um clique em Sim para continuar.

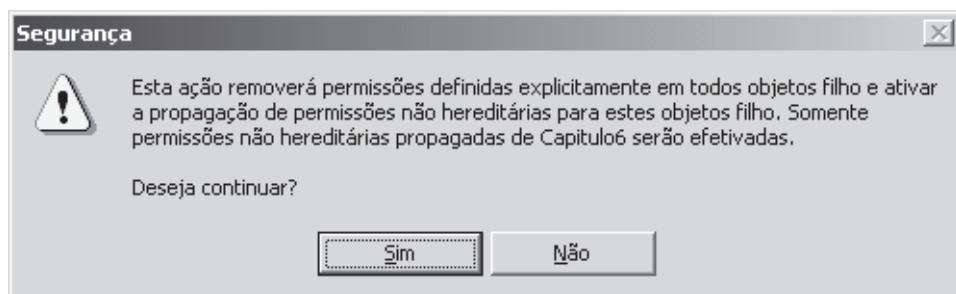


Figura 14.25: Confirmando as alterações.

13. Você estará de volta à guia Segurança, da janela de propriedades da pasta.
14. Vamos adicionar o usuário IUSR_SERVIDOR para que ele tenha permissões de acesso à pasta.
15. Dê um clique no botão Adicionar. Surge a janela “Selecione Usuários, Computadores ou Grupos”. Localize o usuário IUSR_SERVIDOR e dê um clique sobre o mesmo para marcá-lo. Depois dê um clique no botão Adicionar, para incluir o usuário na parte inferior da janela, conforme indicado na Figura 14.26.

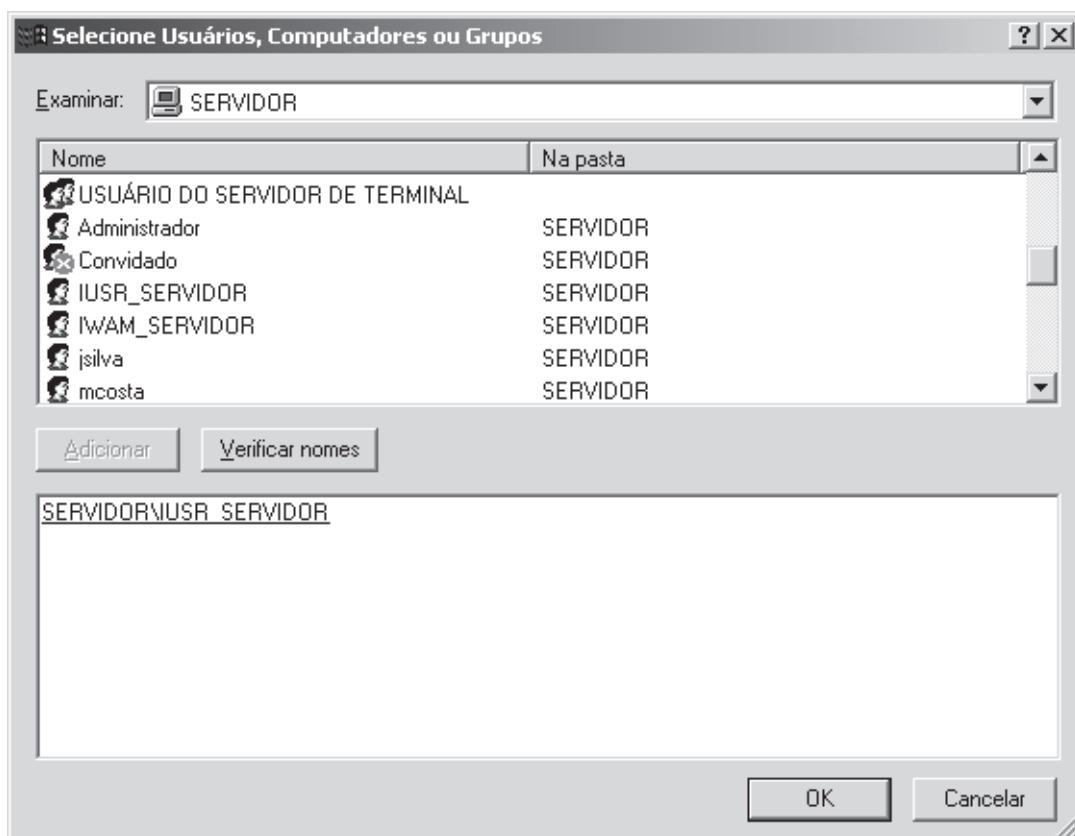


Figura 14.26: Adicionando o usuário IUSR_SERVIDOR na lista de usuários que têm permissão de acesso à pasta selecionada.

16. Dê um clique no botão OK. Observe que o usuário IUSR_SERVIDOR já consta na listagem de usuários.
17. Defina as permissões conforme indicado na Figura 14.27 e dê um clique no botão OK.

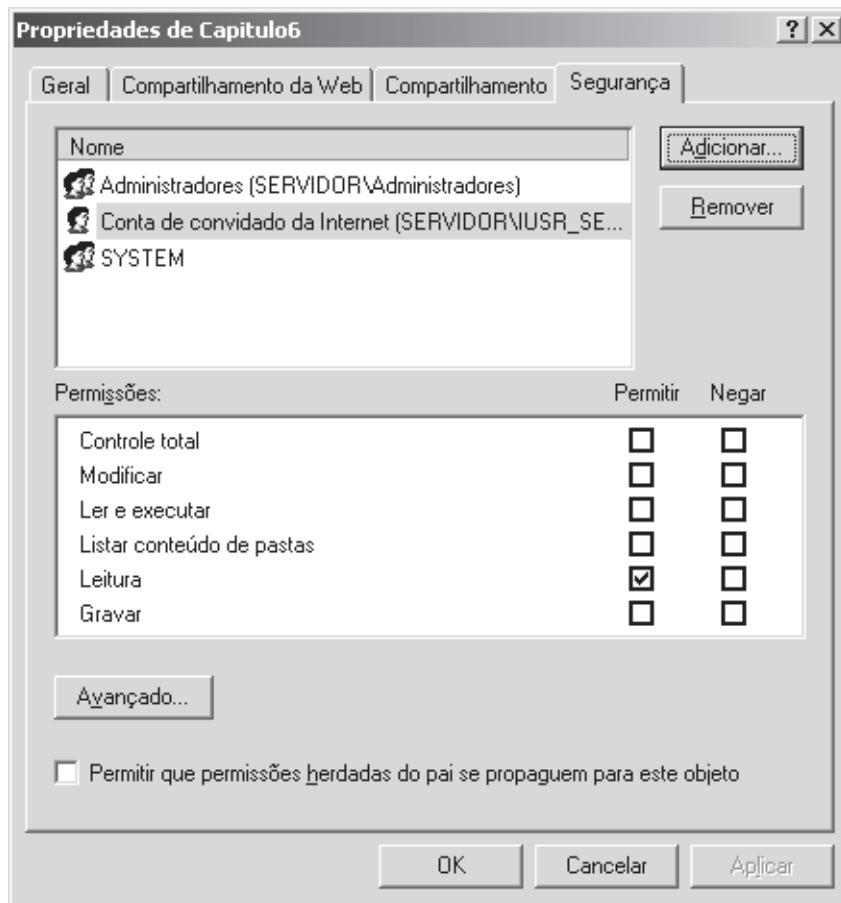


Figura 14.27: Redefinindo as permissões de acesso à pasta selecionada.

Feito isso foram reatribuídas as permissões NTFS originais e qualquer usuário volta a ter acesso à pasta (no nosso exemplo era a pasta Capítulo 8) e a todo o seu conteúdo, porém com permissão somente para leitura.

Agora o usuário já poderá acessar a página, pois não será mais retornada a mensagem de acesso negado.

Com este exemplo, podemos constatar que o servidor IIS trabalha em sintonia com o Windows 2000, de tal forma que os recursos de segurança do Sistema Operacional podem ser utilizados pelo IIS.

Nós detalhamos um pouco mais o primeiro tipo de acesso – Acesso anônimo, para explicar alguns conceitos importantes em detalhes. Agora passaremos a estudar outros tipos de autenticação possíveis com o IIS. Lembrando que a autenticação com o IIS é apenas um dos tantos níveis de segurança que podemos configurar.

Autenticação Básica

A autenticação básica é uma das mais antigas formas de autenticação que existem, desenvolvidas desde a época dos primeiros servidores Web como o NCSA e o Cern HTTP. Neste tipo de autenticação, o usuário precisa fornecer um

Username e uma senha. O método de autenticação básica é um padrão de mercado amplamente usado para coletar informações de nome de usuário e senha. A autenticação básica funciona da seguinte forma:

1. O navegador exibe uma caixa de diálogo na qual o usuário pode digitar seu Username e senha de conta do Windows 2000, previamente cadastrada. Por isso, é um pré-requisito da autenticação básica, que o usuário já possua uma conta cadastrada no Windows 2000.
2. O navegador tenta estabelecer uma conexão usando essas informações.
3. Se o servidor rejeitar as informações, o navegador da Web exibe repetidamente a caixa de diálogo até que o usuário digite um nome de usuário e uma senha válidos ou feche a caixa de diálogo.
4. Quando o servidor Web verifica que o nome de usuário e a senha correspondem a uma conta de usuário do Windows 2000 válida, a conexão é estabelecida e o acesso à página solicitada é liberado.

A autenticação básica apresenta, como principal requisito, o fato de que o usuário deve ter uma conta no Windows 2000. Para sites que são acessados por um grande número de usuários pode não ser uma boa opção. Além disso, o fato de o usuário ter que digitar um username e senha não é muito “simpático”.

Uma das grandes desvantagens deste método de autenticação é o fato de que a senha não é criptografada ao ser transmitida pela rede. A codificação que é feita é extremamente simples de ser quebrada; por isso este método de autenticação não é dos mais seguros.

A vantagem da autenticação básica é que ela faz parte da especificação do HTTP e tem suporte da maioria dos navegadores. A desvantagem é que, pelo fato de os navegadores que usam a autenticação básica transmitirem senhas de forma descriptografada, ao monitorar as comunicações na sua rede, alguém pode interceptar e decifrar facilmente essas senhas usando ferramentas disponíveis publicamente na Internet. Portanto, a autenticação básica não é recomendada a menos que você tenha certeza de que a conexão entre o usuário e seu servidor Web é segura, como uma conexão direta via cabo ou uma linha dedicada.

Autenticação Integrada do Windows

A autenticação integrada do Windows (chamada anteriormente NTLM ou autenticação de desafio/resposta do Windows NT) é uma forma segura de autenticação pois o nome de usuário e a senha não são enviados pela rede criptografados. Quando você ativa a autenticação integrada do Windows, o navegador do usuário verifica a validade da senha através de uma troca criptográfica com o servidor Web.

A autenticação integrada do Windows pode usar o protocolo de autenticação Kerberos versão 5 e o seu próprio protocolo de autenticação desafio/resposta. Se o Serviço de diretório – Active Directory, estiver instalado no servidor e o navegador for compatível com o protocolo de autenticação Kerberos versão 5, o protocolo Kerberos versão 5 e o

IMPORTANTE: A autenticação integrada do Windows (que veremos logo em seguida) tem prioridade sobre a autenticação básica. O navegador escolherá a autenticação integrada do Windows e tentará usar as informações de logon atuais do Windows antes de solicitar ao usuário um nome de usuário e uma senha. Atualmente, somente o Internet Explorer, versão 2.0 e posterior, oferece suporte à autenticação integrada do Windows.

protocolo desafio/resposta serão usados; caso contrário, somente o protocolo desafio/resposta será usado.

O protocolo de autenticação Kerberos versão 5 é um recurso da arquitetura do Windows 2000 Distributed Services. Para que a autenticação do Kerberos versão 5 seja bem-sucedida, o cliente e o servidor devem ter uma conexão confiável com um Key Distribution Center (KDC) e devem ser compatíveis com os Serviços do Active Directory. A situação ideal é onde o cliente utiliza o Windows 2000 Professional.

A autenticação integrada do Windows funciona da seguinte forma:

1. Diferentemente da autenticação básica, ela não solicita inicialmente um nome de usuário e uma senha. As informações atuais de usuário logado e sobre o computador cliente são usadas para a autenticação integrada do Windows.
2. No entanto, se a troca da autenticação não consegue identificar o usuário, o navegador solicita ao usuário um nome de usuário e uma senha de conta de usuário do Windows, que ele processa usando a autenticação integrada do Windows.
3. O Internet Explorer continuará a solicitar o usuário até que ele digite um nome de usuário e uma senha válidos ou feche a caixa de diálogo de solicitação.

Embora a autenticação integrada do Windows seja segura, ela tem duas limitações.

1. Somente o Microsoft Internet Explorer, versão 2.0 ou posterior, oferece suporte a esse método de autenticação.
2. A autenticação integrada do Windows não funciona em conexões feitas através de um Servidor Proxy.

NOTA: O Internet Explorer, versão 4.0 e posterior, pode ser configurado para solicitar inicialmente informações do usuário, se necessário. Para obter mais informações, consulte a documentação do Internet Explorer.

Portanto, a autenticação integrada do Windows é mais adequada para um ambiente de Intranet, no qual o usuário e o servidor Web estão no mesmo domínio e os administradores podem garantir que todos os usuários tenham o Microsoft Internet Explorer, versão 2.0 ou posterior.

Autenticação Utilizando Certificados

Este é um dos métodos de autenticação que mais vêm crescendo em termos de utilização. Inicialmente os Certificados digitais foram projetados como um instrumento de autenticação segura para a Internet, porém o seu uso apresentou tantas vantagens que hoje é bastante comum a utilização de Certificados digitais em Intranets e Extranets.

A tecnologia de certificados usa os recursos de segurança de Secure Sockets Layer (SSL, camada de soquetes de segurança) do servidor Web para dois tipos de autenticação.

É possível usar um certificado de servidor para permitir que os usuários façam a autenticação do seu site da Web antes de transmitir informações pessoais, como um número de cartão de crédito. Além disso, você pode usar certificados de cliente para autenticar os usuários que solicitam informações no seu site da Web. O SSL faz a autenticação verificando o conteúdo de uma identificação digital (O Certificado digital) criptografada submetida pelo navegador do usuário

durante o processo de logon (Os usuários obtêm certificados de cliente de uma organização independente mutuamente confiável – Autoridade Certificadora.). Os certificados de servidor contêm geralmente informações sobre sua empresa e a organização que emitiu o certificado. Os certificados de cliente contêm normalmente informações de identificação sobre o usuário e a organização que emitiu o certificado.

Mapeamento do Certificado Cliente

Você pode associar, ou mapear, certificados de cliente a contas de usuário do Windows no IIS. Depois que você cria e ativa um mapa de certificado, sempre que um usuário faz logon com um certificado de cliente, seu servidor Web associa automaticamente esse usuário à conta de usuário do Windows apropriada. Dessa forma, você pode autenticar automaticamente os usuários que fazem logon com certificados de cliente, sem exigir o uso da autenticação básica ou integrada do Windows. É possível mapear um certificado de cliente para uma conta de usuário do Windows ou muitos certificados de cliente para uma conta. Por exemplo, se você tivesse vários departamentos ou empresas diferentes no seu servidor, cada uma com seu próprio site da Web, seria possível usar o mapeamento vários-para-um para mapear todos os certificados de cliente de cada departamento ou empresa para o próprio site da Web. Dessa forma, cada site forneceria acesso somente aos próprios clientes.

O tipo de autenticação é apenas um dos aspectos que precisam ser definidos. Devem ser consideradas diversas questões. A seguir segue uma lista de questões que devem ser levadas em consideração na hora de decidir sobre o tipo de autenticação que iremos configurar no IIS, ou se devemos configurar mais do que um tipo de autenticação.

- ◆ Para um site público, ou áreas de acesso público, a autenticação utilizando Acesso anônimo é a mais indicada, pois evita que o usuário tenha que fornecer um username e senha.
- ◆ Para acesso ao conteúdo de uma Intranet, uma das primeiras opções a serem pensadas é a utilização da autenticação integrada do Windows. Pois sendo uma Intranet um ambiente controlado, é possível garantir que todos os clientes satisfaçam as condições exigidas pela autenticação Integrada.
- ◆ Para sites que trabalham com dados sensíveis como serviços bancários pela Internet e Comercio Eletrônico, sem dúvida que a utilização de Certificados Digitais é o mais indicado. Em Intranets também têm sido utilizados Certificados Digitais, pois, conforme descrevemos no início do capítulo, a maioria dos ataques parte de usuários da própria Intranet da empresa. Muitas vezes nos preocupamos muito com os ataques externos e esquecemos as ameaças que vêm de dentro da empresa.

Configurando o Tipo de Autenticação no IIS

Neste item veremos como configurar uma ou mais opções de autenticação no IIS, lembrando que podemos ter diferentes tipos de autenticação em diferentes partes de um site armazenado em um servidor IIS. Por exemplo, para uma área de acesso público podemos utilizar autenticação anônima, para uma área mais restrita podemos utilizar somente autenticação integrada do Windows. Podemos inclusive configurar o nível de autenticação para uma página HTML ou ASP, individualmente.

Para configurar o tipo de autenticação faça o seguinte:

1. Faça o logon no Windows 2000 Server, com permissões de administrador.

2. Abra o Gerenciador do Internet Services: Iniciar -> Programas -> Ferramentas administrativas -> Gerenciador do Internet Services.
3. É aberto o console de gerenciamento do IIS.
4. Dê um clique duplo no nome do computador. No nosso exemplo o nome é Servidor.
5. Surgem as opções indicadas na Figura 14.28.

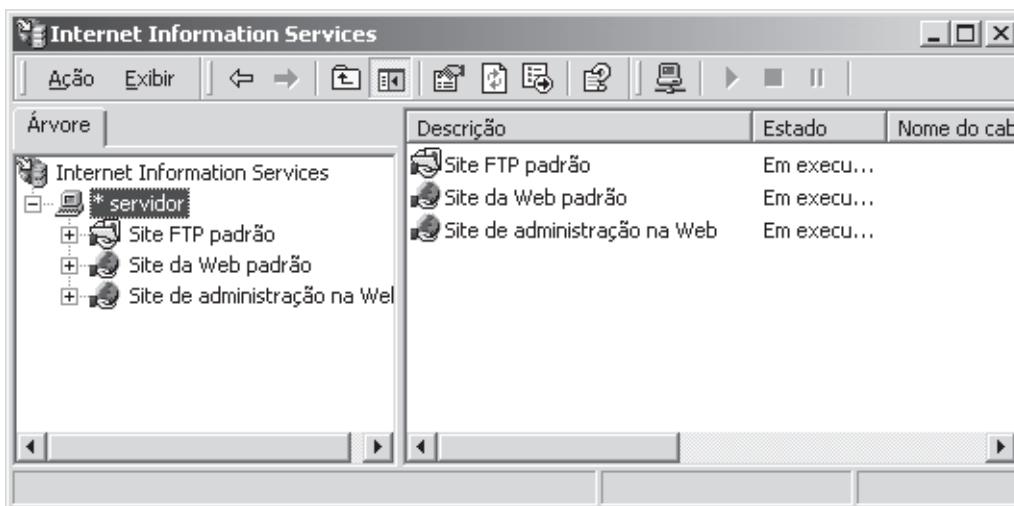


Figura 14.28: Opções de gerenciamento do IIS.

6. Neste momento podemos configurar o tipo de autenticação para todos os aplicativos Web contidos no Servidor ou para cada aplicativo individualmente.
7. A título de exemplo, vamos configurar o tipo de autenticação para o site Web padrão.
8. Clique com o botão direito do mouse sobre a opção “Site da Web padrão” (ou na opção correspondente, caso você tenha alterado este nome). No menu de opções que surge dê um clique em Propriedades.
9. Será exibida a janela “Propriedades de Site Web padrão”.
10. Dê um clique na guia Segurança de pasta. Serão exibidas as opções indicadas na Figura 14.29.
11. A primeira opção desta guia é Controle de acesso anônimo e autenticação. Dê um clique no botão Editar, ao lado desta opção. Surge a janela Métodos de autenticação, indicada na Figura 14.30.
12. Observe que, por padrão, estão definidas as opções de Acesso anônimo e Autenticação integrada do Windows.
13. Nesta janela você pode definir qual ou quais tipos de autenticação que o servidor IIS deverá suportar para o site Web padrão. Além disso, você pode configurar qual a conta que será utilizada para o acesso anônimo, conforme descrito anteriormente.

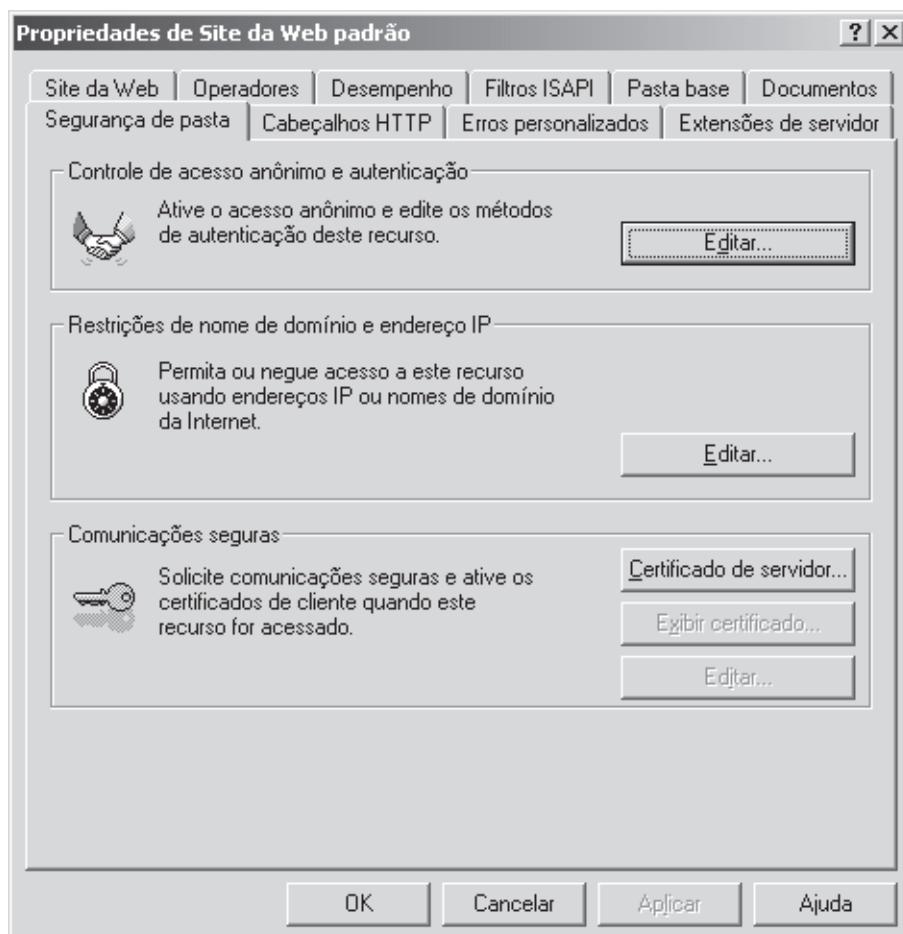


Figura 14.29: As opções da guia Segurança de pasta.

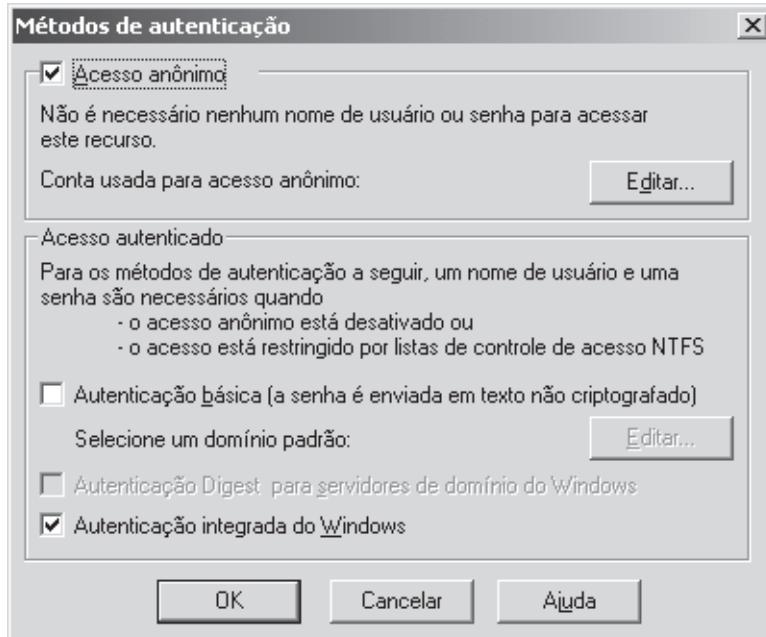


Figura 14.30: As opções para métodos de autenticação.

14. Selecione as opções desejadas. Se você clicar na opção Autenticação básica, o IIS emite um aviso de que para esta opção as senhas serão transmitidas sem criptografia, conforme indicado na Figura 14.31.

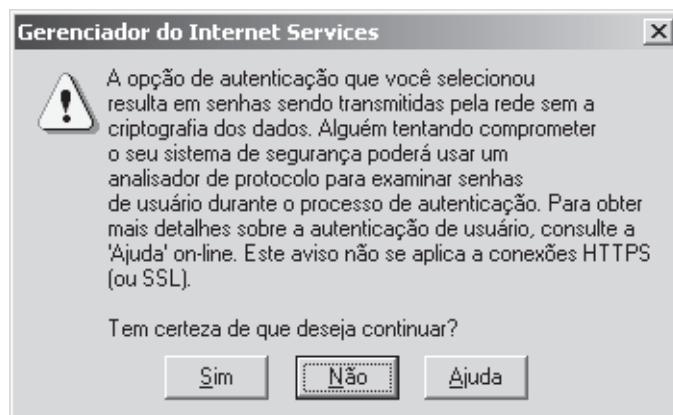


Figura 14.31: Aviso de que para a autenticação básica a senha é transmitida sem criptografia.

15. Dê um clique em Sim e a autenticação básica será habilitada.
 16. Dê um clique no botão OK para fechar a janela de configuração dos tipos de autenticação.
 17. Você estará de volta à janela de Propriedades do site Web padrão. Dê um clique no botão OK para fechá-la.
 18. Caso alguma aplicação Web ou pasta virtual do Servidor possua uma configuração diferente da definida para o site Web padrão, o IIS abre uma janela informando qual site possui uma configuração diferente e perguntando se você deseja estender as configurações do site Web padrão para as pastas virtuais e aplicativos Web internos, conforme indicado na Figura 14.32.

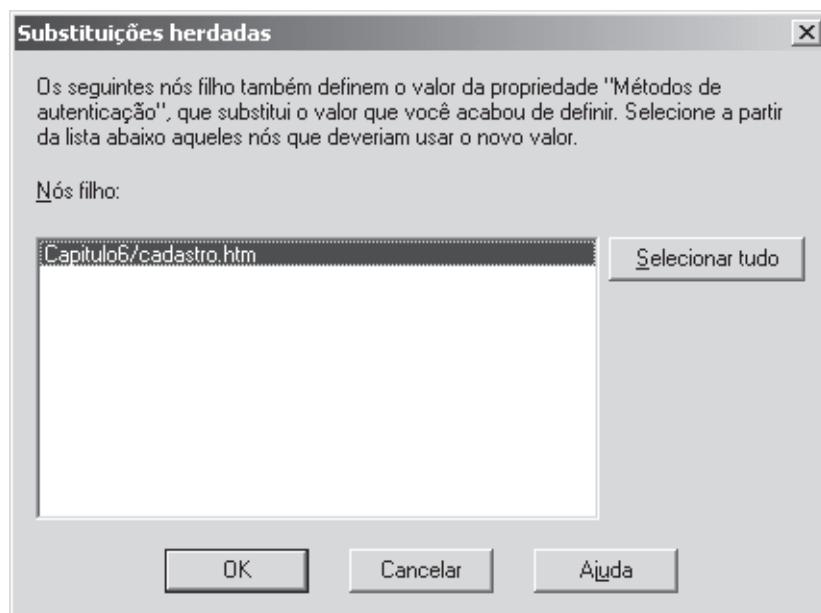


Figura 14.32: Estendendo as configurações para as aplicações e pastas virtuais.

19. Para estender as configurações basta selecionar uma ou mais das opções mostradas e clicar no botão OK. Você também pode utilizar o botão Selecionar tudo e depois clicar em OK.

- Você estará de volta ao Gerenciador do Internet Services. Feche-o.

Para configurar as opções para uma pasta Virtual em particular, basta localizá-la abaixo da opção Site Web padrão, clicar com o botão direito sobre a mesma e clicar na opção Propriedades. Depois é só seguir os passos indicados anteriormente.

Mais Configurações de Segurança do IIS

Podemos efetuar outras configurações relacionadas com segurança, no servidor IIS. Existem algumas opções que limitam o tipo de ação que o usuário pode tomar em uma determinada pasta virtual ou em uma página especificamente. Neste item iremos estudar diversas destas opções. Dividiremos as mesmas em dois grupos:

- ◆ Opções gerais de segurança.
- ◆ Opções relacionadas com uma aplicação Web.

Configurando Opções Gerais de Segurança

Estas configurações são definidas, normalmente, em nível de pasta virtual. Para configurar estas opções faça o seguinte:

- Faça o logon no Windows 2000 Server, com permissões de administrador.
- Abra o Gerenciador do Internet Services: Iniciar -> Programas -> Ferramentas administrativas -> Gerenciador do Internet Services.
- É aberto o console de gerenciamento do IIS.
- Dê um clique duplo no nome do computador. No nosso exemplo o nome é Servidor.

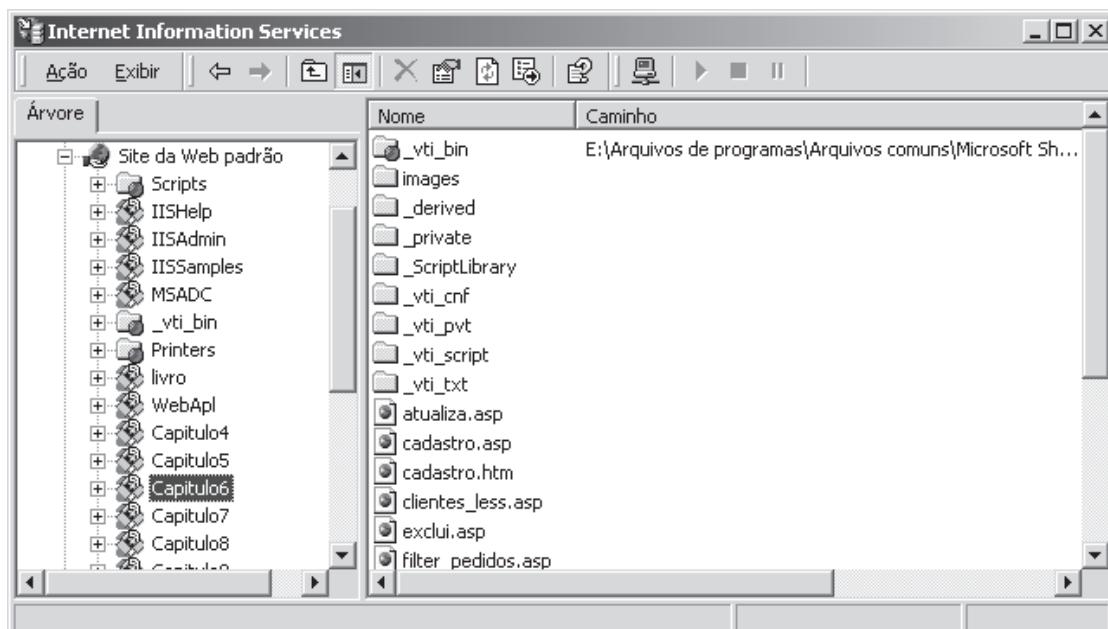


Figura 14.33: Configurando opções gerais de segurança para – Capítulo6.

5. Dê um clique no sinal de + ao lado de Site Web padrão. Serão exibidas as diversas pastas virtuais disponíveis no servidor.
6. A título de exemplo, vamos configurar as opções gerais de segurança para a pasta Capítulo6, conforme indicado na Figura 14.33.
7. Clique com o botão direito do mouse sobre a opção Capítulo6. No menu de opções que surge dê um clique em Propriedades.
8. Será exibida a janela “Propriedades de Capítulo6”.
9. Na guia Pasta (que já deve estar sendo exibida), existem diversas opções de configuração. Abaixo descrevemos cada uma destas opções.
 - ◆ Acesso ao código-fonte do Script: Selecione esta opção para permitir que os usuários accessem o código-fonte se a permissão de leitura ou gravação estiver definida. O código-fonte inclui scripts nos aplicativos ASP.
 - ◆ Leitura: Selecione esta opção para permitir que os usuários leiam ou façam o download dos arquivos ou diretórios e de suas propriedades associadas.
 - ◆ Gravação: Selecione esta opção para permitir que os usuários carreguem os arquivos e suas propriedades associadas no diretório ativado no servidor ou alterem o conteúdo de um arquivo ativado para gravação. A gravação só poderá ser feita com um navegador que dê suporte ao recurso PUT do protocolo padrão HTTP 1.1. Cuidado com esta permissão. Dificilmente você precisará habilitar permissão de Gravação para áreas em que é permitida a autenticação com usuário anônimo.
 - ◆ Pesquisa em Pasta: Selecione esta opção para permitir que o usuário veja uma listagem em hipertexto dos arquivos e subdiretórios deste diretório virtual. Os diretórios virtuais não aparecerão nas listagens de diretórios; os usuários devem saber o alias do diretório virtual. Caso o usuário digite o endereço para o caminho da pasta e não especifique um documento a ser carregado, será exibida uma listagem semelhante à indicada na Figura 14.34.

NOTA: Caso você não tenha criado uma pasta virtual Capítulo6, utilize qualquer pasta virtual disponível no seu servidor IIS.

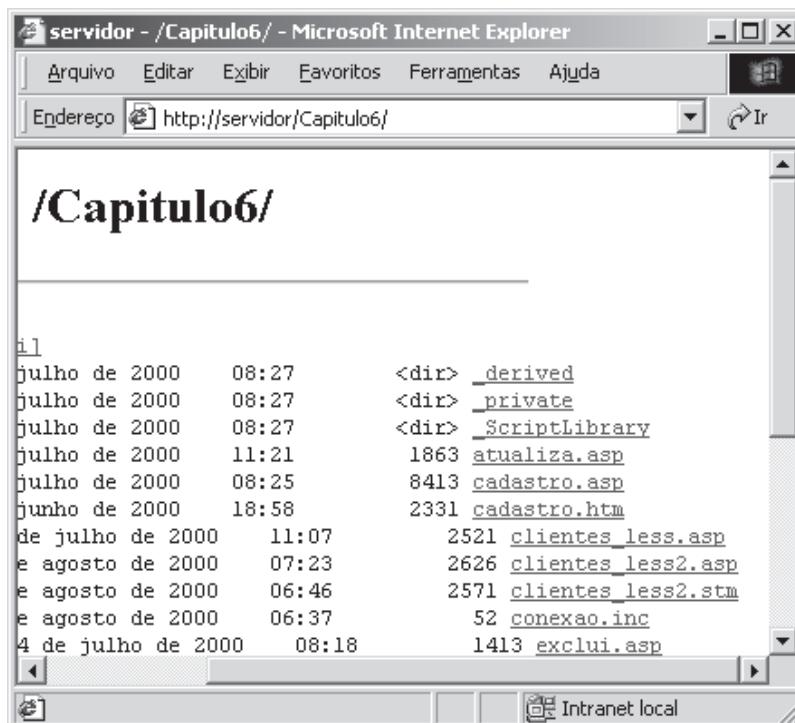


Figura 14.34: Listagem exibida quando a opção Pesquisa em Pasta está habilitada.

Caso esta opção não esteja marcada e o usuário digite o endereço para a pasta, sem especificar um arquivo a ser carregado, será retornada a mensagem de erro indicada na Figura 14.35.

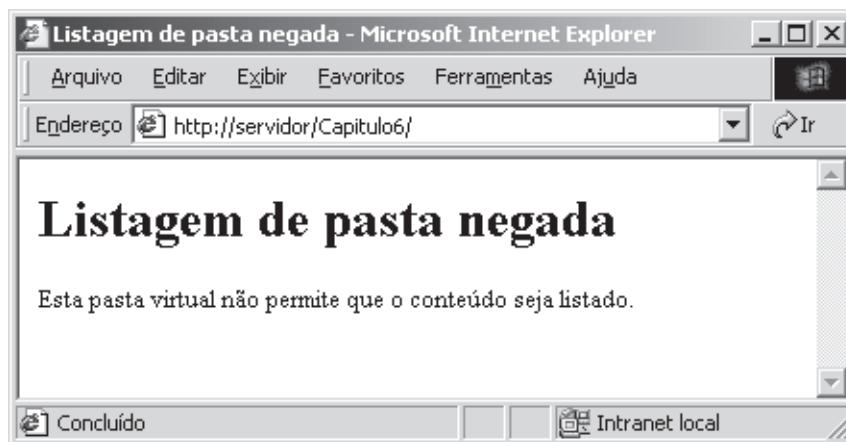


Figura 14.35: Mensagem de erro quando a opção Pesquisa em Pasta estiver desabilitada.

- ◆ Criar log de visitantes: Selecione esta opção para registrar as visitas feitas a este diretório em um arquivo de log. As visitas serão registradas somente se o log estiver ativado para este site da Web.
- ◆ Indexar este recurso: Para permitir que o Serviço de indexação da Microsoft (Index Services) inclua este diretório em um índice de texto completo do site da Web, selecione esta opção.

De uma maneira geral, recomendo que as opções Gravação e Pesquisa em pasta somente sejam marcadas em situações especiais e para áreas que não contenham dados confidenciais.

Uma questão importante é sobre a combinação entre as permissões definidas no IIS e as permissões NTFS. Por exemplo, vamos supor que o usuário tenha sido autenticado como usuário anônimo e está tentando gravar conteúdo em uma pasta virtual na qual o IIS possui permissão de gravação, porém as permissões NTFS não permitem que o usuário anônimo faça gravações. Como fica esta combinação?

Quando uma página é solicitada, o IIS segue a seqüência indicada na Figura 14.36.

Observe que primeiro o IIS verifica se o usuário tem permissões da Web para o recurso solicitado. Se o usuário não tiver, a solicitação falha e o usuário recebe uma mensagem “403 Acesso proibido”. O IIS verifica as permissões do NTFS para o recurso. Se o usuário não tiver permissões do NTFS para o recurso, a solicitação falha e o usuário recebe uma mensagem “401 Acesso negado”.

Então a resposta para o nosso exemplo é que o usuário não conseguiria gravar o conteúdo, pois o mesmo não teria as permissões NTFS necessárias e o mesmo acabaria recebendo a mensagem de erro “401 Acesso negado”.

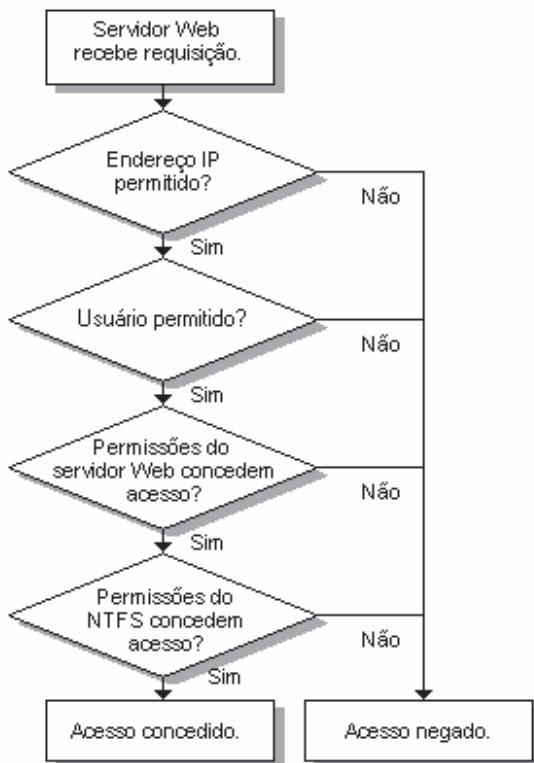


Figura 14.36: Seqüência de verificação do IIS.

Configurando de Segurança Para Aplicativos Web no IIS

Estas configurações são definidas, também utilizando a guia Pasta da janela de propriedades para a pasta virtual que representa a aplicação Web. Para configurar estas opções faça o seguinte:

1. Faça o logon no Windows 2000 Server, com permissões de administrador.
2. Abra o Gerenciador do Internet Services: Iniciar -> Programas -> Ferramentas administrativas -> Gerenciador do Internet Services.
3. É aberto o console de gerenciamento do IIS.
4. Dê um clique duplo no nome do computador. No nosso exemplo o nome é Servidor.
5. Dê um clique no sinal de + ao lado de Site Web padrão. Serão exibidas as diversas pastas virtuais disponíveis no servidor.
6. A título de exemplo, vamos configurar as opções de aplicação para a pasta Capítulo6.
7. Clique com o botão direito do mouse sobre a opção Capítulo6. No menu de opções que surge dê um clique em Propriedades.
8. Será exibida a janela “Propriedades de Capítulo6”.
9. Certifique-se de que estão sendo exibidas as opções da guia Pasta.
10. Na parte de baixo da guia Pasta, temos um grupo de opções chamado Configurações do aplicativo. Abaixo temos uma descrição de cada uma destas opções.

NOTA: Caso você não tenha criado uma pasta virtual Capítulo6, utilize qualquer pasta virtual disponível no seu servidor IIS.

Conforme descrito anteriormente, uma Aplicação Web do IIS é definida pela estrutura de diretórios em que está localizado (falaremos mais sobre Aplicações Web, mais adiante). Para obter mais informações, consulte o tópico “Aplicações Web”, mais adiante.

- ◆ **Nome do aplicativo:** Normalmente utilizamos o mesmo nome da pasta virtual.
- ◆ **Botão Remover:** Se clicarmos neste botão removemos todas as opções de aplicativo, e transformamos o aplicativo Web em uma simples pasta virtual, para a qual não se aplicam os conceitos de Aplicação Web. No capítulo sobre Web Services, tivemos que verificar se uma determinada pasta estava configurada como aplicação Web, para que pudéssemos testar o Web Service “CalculosLegais”, que criamos no Capítulo 13.
- ◆ **Permissões de execução:** Esta opção determina o nível de execução de programa permitido para recursos de diretórios virtuais ou deste site. Temos as seguintes opções:
 - ◆ **Nenhum:** Somente arquivos estáticos, como os arquivos HTML (Hypertext Markup Language, linguagem de marcação de hipertexto) ou os arquivos de imagem, podem ser acessados. Não permite que scripts ASP sejam executados.
 - ◆ **Somente scripts:** Somente scripts, como os scripts ASP, podem ser executados. Este é o padrão normalmente definido.
 - ◆ **Scripts e executáveis:** Todos os tipos de arquivos podem ser acessados ou executados. Cuidado com esta configuração. Muitos dos ataques conhecidos consistem em enviar e executar arquivos executáveis no servidor a ser atacado. Os executáveis enviados normalmente abrem portas de segurança que estavam fechadas pelo Administrador.
- ◆ **Proteção do aplicativo:** Temos as seguintes opções:
 - ◆ **Baixa:** Selecione esta opção para que os aplicativos sejam executados no mesmo processo que os serviços da Web (opção baixo); neste caso, se um dos aplicativos apresentar problema e desativar o processo do servidor Web, todos os aplicativos ficarão indisponíveis até que o servidor Web tenha sido normalizado.
 - ◆ **Média (em pool):** Selecione esta opção para que a aplicação Web seja executada em um processo em pool isolado em que outros aplicativos também são executados.
 - ◆ **Alta (isolada):** Neste caso a aplicação Web será executada em seu próprio espaço de endereçamento e se a mesma apresentar problemas, as outras aplicações Web, bem como o servidor IIS, continuam funcionando normalmente. A execução isolada aumenta a disponibilidade do servidor Web, pois problemas em uma aplicação não irão afetar todo o servidor, porém consomem mais recursos, como memória, pois cada aplicação é executada em seu próprio espaço de memória.

NOTA: Para determinar qual opção representa uma aplicação Web e qual é simplesmente uma pasta virtual é só observar o ícone ao lado do nome, no Gerenciador do Internet Services. Opções que são simplesmente uma pasta virtual são representadas por um envelope amarelo; já aplicações Web possuem um ícone que parece uma caixa aberta, conforme indicado na Figura 14.37.

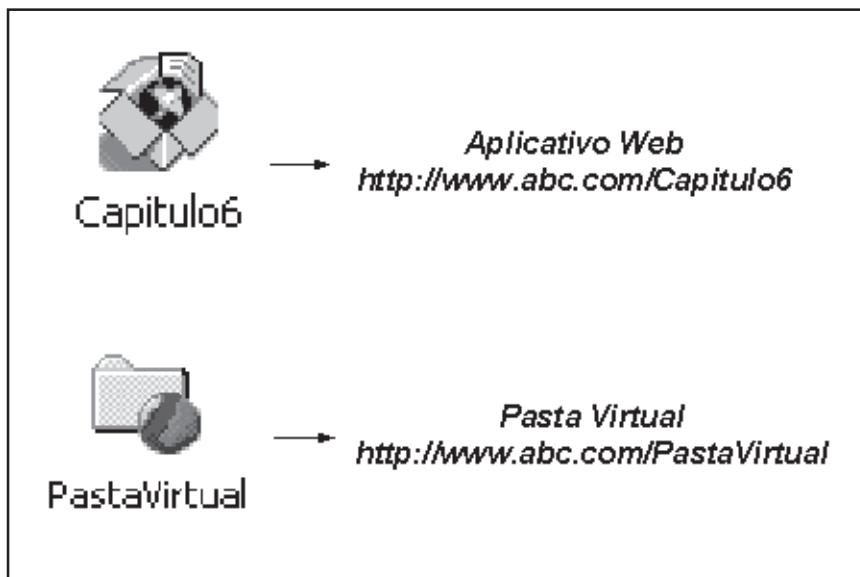


Figura 14.37: Ícones para uma pasta virtual e para uma aplicação Web.

Definindo Restrições de Acesso em Nível de Endereço IP

Podemos definir restrições de acesso em nível de endereço IP.

Por exemplo, vamos supor que existe uma área do site que está em desenvolvimento, de tal forma que a mesma não deva ser acessada, nem por usuários da Intranet da empresa, muito menos por usuários da Internet. Somente os participantes do grupo de desenvolvimento é que devem ter acesso a esta parte do site. Podemos, sem maiores problemas, limitar o acesso, de tal maneira que somente as estações de trabalho dos desenvolvedores tenham acesso à área de desenvolvimento do site.

Neste item aprenderemos a configurar uma aplicação Web ou uma pasta virtual do IIS, para limitar o acesso com base no endereço IP do usuário.

Para definir restrições de acesso em nível de endereço IP faça o seguinte:

1. Faça o logon no Windows 2000 Server, com permissões de administrador.
2. Abra o Gerenciador do Internet Services: Iniciar -> Programas -> Ferramentas administrativas -> Gerenciador do Internet Services.
3. É aberto o console de gerenciamento do IIS.
4. Dê um clique duplo no nome do computador. No nosso exemplo o nome é Servidor.
5. Nas opções que surgem dê um clique com o botão direito do mouse sobre a opção “Site da Web padrão” (ou na opção correspondente, caso você tenha alterado este nome). No menu de opções que surge dê um clique em Propriedades.
6. Será exibida a janela “Propriedades de Site Web padrão”.
7. Dê um clique na guia Segurança de pasta.

8. O segundo grupo de opções desta guia é “Restrições de nome de domínio e endereço IP”. Dê um clique no botão Editar, ao lado desta opção. Surge a janela, indicada na Figura 14.38.

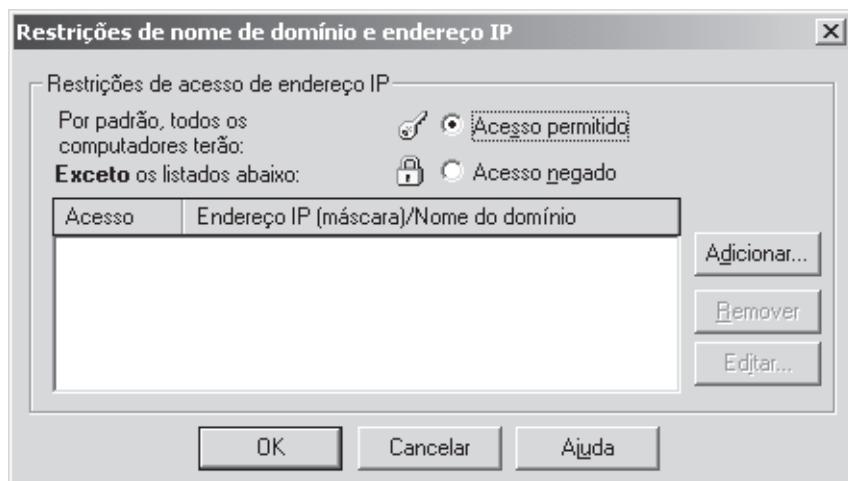


Figura 14.38: Janela para configuração das restrições de nome de domínio e endereço IP.

Observe que nós temos duas opções para configuração, conforme descrito a seguir:

- ◆ **Acesso permitido:** Se esta opção estiver marcada, todo mundo terá acesso ao site, com exceção dos endereços IP que estiverem indicados na listagem “Exceto os listados abaixo”. Este mecanismo é chamado de “Lista Negra”, ou seja, todo mundo tem acesso, com exceção de quem está na lista.
 - ◆ **Acesso negado:** Se esta opção estiver marcada, ninguém terá acesso ao site, com exceção dos endereços IP que estiverem indicados na listagem “Exceto os listados abaixo”. Este mecanismo é chamado de “Lista Branca”, ou seja, ninguém tem acesso, com exceção de quem está na lista.
9. A título de exemplo, deixe marcada a opção Acesso permitido. Agora vamos negar acesso para um computador específico.
10. Dê um clique no botão Adicionar. Surge a janela indicada na Figura 14.39.

NOTA: No exemplo do livro estarei negando acesso para o IP: 10.204.123.1, que é o endereço IP do meu computador. Utilize o endereço IP do equipamento para o qual você quer negar acesso, em nível de teste.

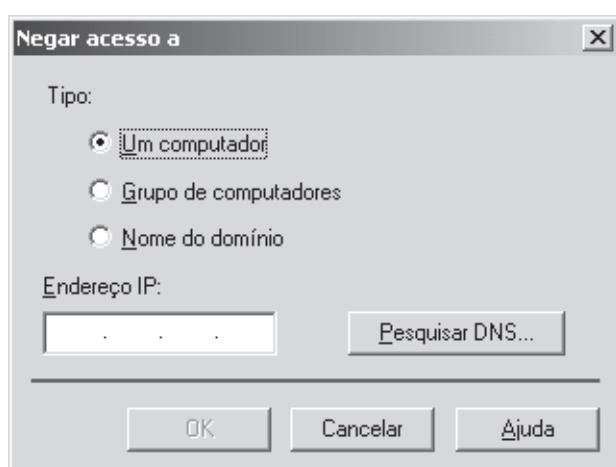


Figura 14.39: Definindo o alcance das restrições impostas.

Nesta janela temos três opções a serem escolhidas, conforme explicado a seguir:

- ◆ **Um computador:** Neste caso estamos negando acesso para um computador específico. Basta digitar o endereço IP do mesmo. No nosso exemplo, utilizaremos esta opção e iremos digitar o IP 10.204.123.1.
- ◆ **Grupo de computadores:** Se você marcar esta opção, surge, na parte de baixo da janela, mais um campo: Máscara de sub-rede, conforme indicado na Figura 14.40. Podemos utilizar esta opção para negar acesso a uma rede ou segmento de rede inteiro.

Por exemplo, podemos negar acesso a qualquer computador da rede 10.204.123; para isso preencheríamos os campos da seguinte maneira:

Identificação da rede: 10.204.123.0

Máscara de sub-rede: 255.255.255.0

Para negar acesso a todas as máquinas da rede 161.147, utilizariamos a seguinte configuração:

Identificação da rede: 161.147.0.0

Máscara de sub-rede: 255.255.0.0

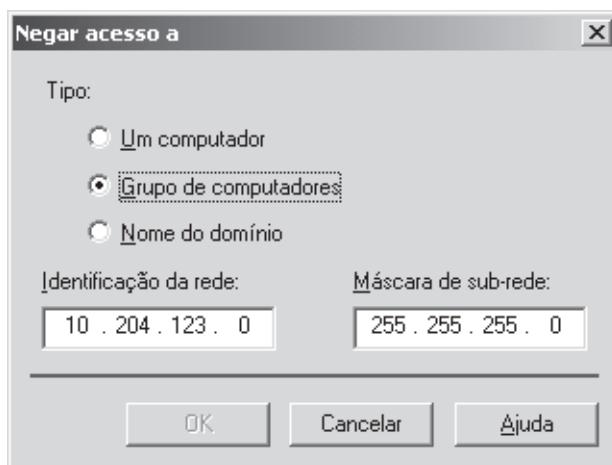


Figura 14.40: Negando acesso para um grupo de computadores.

- ◆ **Nome de domínio:** Esta opção permite que neguemos acesso com base no nome DNS de um grupo de computadores. Por exemplo, podemos negar acesso para setor de contabilidade empresa, negando acesso para o domínio: contabilidade.abc.com.br. Ao tentar utilizar esta opção, o IIS emite uma mensagem avisando que o desempenho do servidor pode ser prejudicado pela ativação desta opção, e perguntando se você realmente deseja ativá-la.
11. Vamos negar o acesso apenas para um computador – 10.204.123.1. Certifique-se de que a opção Um computador esteja marcada e no campo Endereço IP, digite 10.204.123.1, conforme indicado na Figura 14.41.

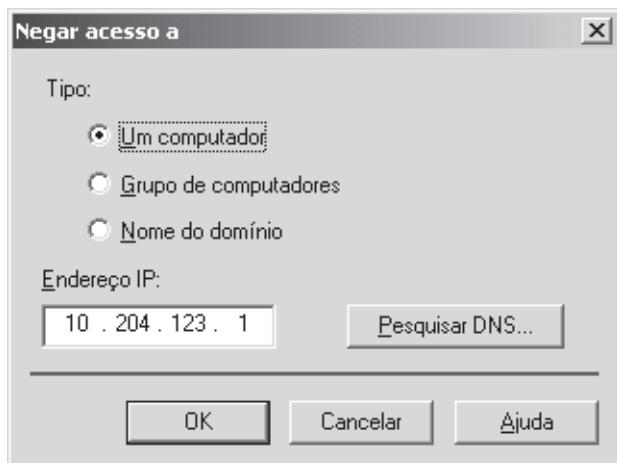


Figura 14.41: Negando acesso para o computador 10.204.123.1.

12. Dê um clique no botão OK para incluir o computador com endereço IP 10.204.123.1, na lista dos computadores com acesso negado, conforme indicado na Figura 14.42.
13. Dê um clique no botão OK e você estará de volta à janela de propriedades do site Web padrão.
14. Dê um clique no botão OK para fechar a janela de propriedades.
15. Caso alguma aplicação Web ou pasta virtual do Servidor possua uma configuração diferente da definida para o site Web padrão, o IIS abre uma janela informando qual site possui uma configuração diferente e perguntando se você deseja estender as configurações do site Web padrão para as pastas virtuais e aplicativos Web internos.

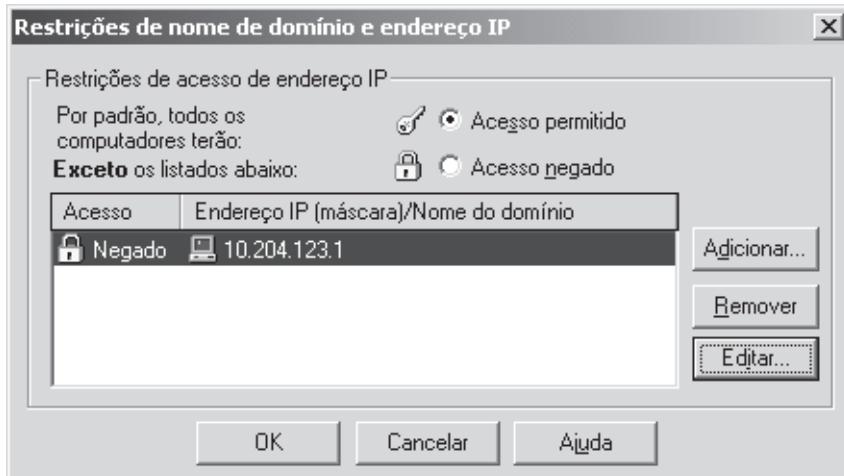


Figura 14.42: O computador com endereço IP 10.204.123.1 já aparece na lista.

16. Para estender as configurações basta selecionar uma ou mais das opções mostradas e clicar no botão OK. Você também pode utilizar o botão Selecionar tudo e depois clicar em OK.
17. Você estará de volta ao Gerenciador do Internet Services. Feche-o.

Agora vou tentar acessar uma página do servidor IIS para o qual o acesso foi negado para o IP do meu computador – 10.204.123.1. Ao tentar fazer o acesso, recebo a mensagem indicada na Figura 14.43.

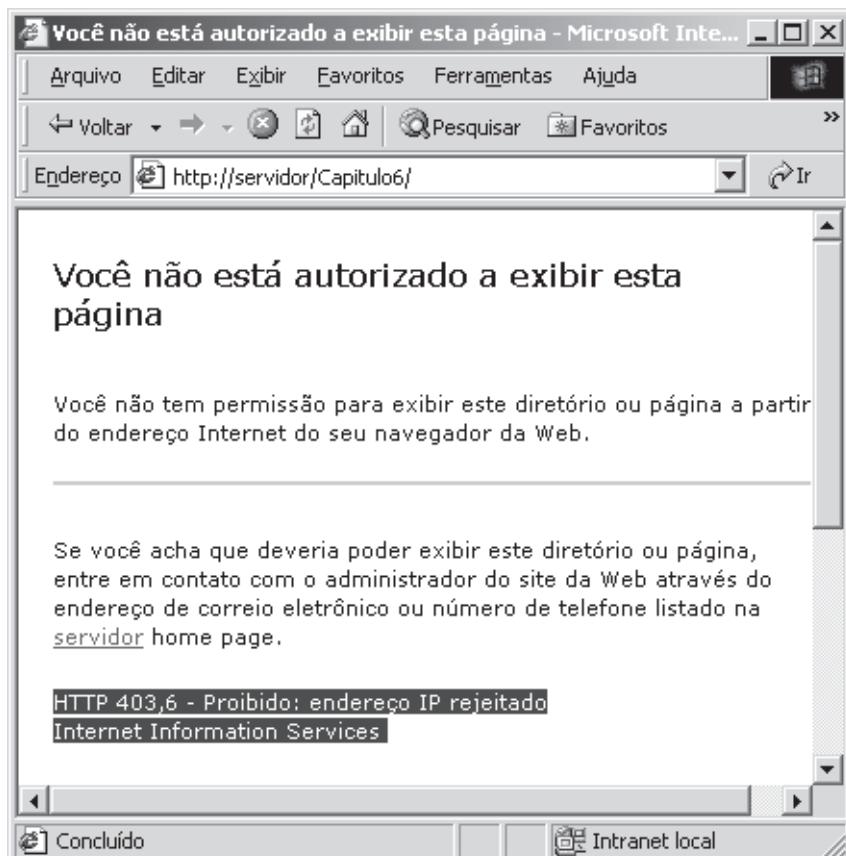


Figura 14.43: Mensagem de que o acesso foi negado.

Observe que a mensagem informa que o endereço IP foi rejeitado. Para que o endereço IP 10.204.123.1 volte a ter acesso ao site, é só seguir os passos indicados anteriormente e removê-lo da lista de endereços IP com acesso negado.

Mecanismos de Segurança do Banco de Dados

Um outro nível de segurança que pode ser configurado é no nível de banco de dados. Os Programas Gerenciadores de banco de dados, como o Microsoft SQL Server, Oracle, Microsoft Access, Sybase, etc., fornecem diversos mecanismos de segurança que, se adequadamente configurados, aumentam bastante o nível de segurança das informações.

No Microsoft SQL Server podemos atribuir níveis de permissão para os usuários do Windows 2000 e até mesmo para o usuário utilizado para acesso anônimo.

Na Figura 14.44, temos um exemplo onde o usuário IUSR_SERVIDOR está recebendo permissão somente para leitura na tabela Orders do Banco de dados Northwind em um Servidor com o Microsoft SQL Server 2000.

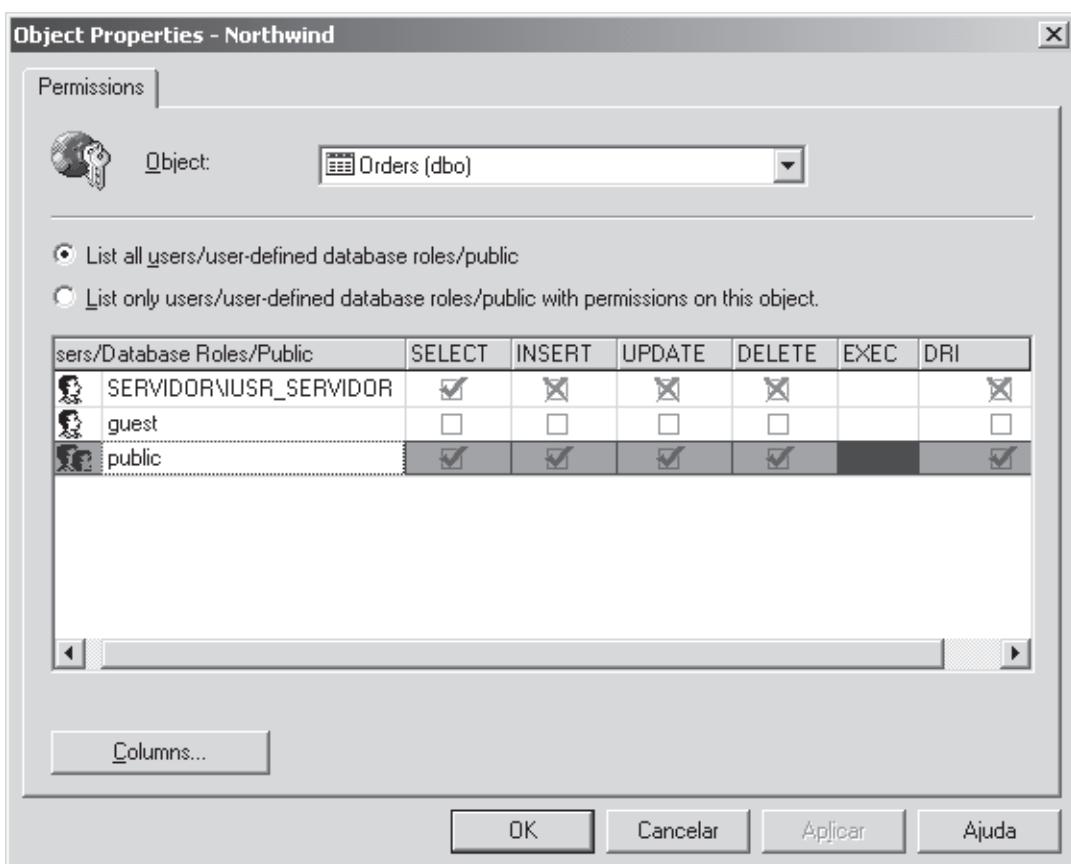


Figura 14.44: Definindo permissões de banco de dados no Microsoft SQL Server 2000.

Na maioria dos bancos de dados, pode ser necessário o fornecimento de um Username e senha para que o acesso ao banco de dados seja liberado. Podemos construir Web Form com ASP.NET, com dois campos, onde o usuário digita o Username e a senha. Ao clicar em um botão Logon, os valores digitados, a página monta a string de conexão de acordo com os dados fornecidos pelo usuário, incluindo o Username e senha.

Agora vamos falar um pouco mais sobre Aplicações Web, no IIS 5.0. Este é o último assunto de segurança relacionado com o IIS e o Windows 2000. Depois passaremos a tratar dos tópicos de segurança específicos do ASP.NET.

NOTA: Para maiores informações sobre configurações de segurança no Microsoft SQL Server 2000, consulte o Capítulo 6 – Segurança no SQL Server 2000”, do livro “SQL Server 2000 Administração & Desenvolvimento – Curso Completo”, 816 páginas, de minha autoria, publicado pela editora Axcel Books (www.axcel.com.br).

O que é uma Aplicação Web no IIS?

Antes de definirmos exatamente o que compõe uma aplicação Web no IIS, vamos fazer uma comparação/explanação entre as aplicações Cliente/Servidor tradicionais e as aplicações Web, com ênfase no conceito de conexão e estado.

Uma Aplicação Cliente/Servidor Tradicional

Em uma aplicação Cliente/Servidor tradicional, temos o cliente instalado em cada uma das estações da rede que farão acesso à aplicação; podemos ter uma camada de lógica instalada no Servidor de

Aplicações, através da qual é feito o acesso ao Banco de dados. Quando o usuário inicia a aplicação cliente, o mesmo se identifica, normalmente através da digitação de um nome de usuário (username ou login) e da digitação de uma senha.

As informações de identificação são utilizadas para estabelecer uma “conexão” com o Servidor. Esta conexão é mantida enquanto a aplicação cliente estiver sendo utilizada. Através desta conexão, o lado Servidor da aplicação consegue identificar o usuário. Com a identificação do usuário podem ser aplicadas regras de segurança, como níveis de permissão de acesso aos dados, níveis de permissão para as funcionalidades da aplicação, log das ações realizadas pelo usuário, etc. Por exemplo, as opções de menu da aplicação Cliente podem ser montadas com base nas permissões de acesso que o usuário possui. Se o usuário não possui permissões para alterar os dados, o menu com as opções de alteração não será exibido.

Através da manutenção da conexão e da respectiva possibilidade de identificar unicamente cada um dos usuários conectados, podemos implementar a maioria das funcionalidades fundamentais nas aplicações Cliente/Servidor tradicionais.

Porém existem diversos problemas na atualização e manutenção deste tipo de aplicação, dentre os quais podemos destacar:

- ◆ A atualização dos sistemas é problemática, pois neste modelo cada vez que houver alteração na camada de apresentação (no modelo de n camadas) ou até mesmo na camada de Lógica do negócio (no modelo mais antigo de 2 camadas), a aplicação terá que ser atualizada em todas as estações de trabalho que utilizam a aplicação.
- ◆ **Controle de versão:** Garantir que todas as estações de trabalho estejam com a última versão da aplicação é uma tarefa nada fácil, a qual, dependendo do número de estações da rede da empresa, pode exigir uma equipe de Help Desk (suporte) praticamente dedicada a esta tarefa.
- ◆ Problemas de conflitos de DLL e padronização de ambiente tornam a manutenção e o gerenciamento destas aplicações uma tarefa difícil e de custo elevado.

NOTA: Para maiores detalhes sobre os modelos de aplicação de 2 ou mais camadas, consulte os Capítulos 1 e 2.

Estes problemas surgem para cada aplicação Cliente/Servidor tradicional que estiver sendo usada. Agora imagine o caso de uma grande empresa, com milhares de computadores ligados em rede, com dezenas de aplicações. A situação torna-se insustentável, além do custo de manutenção e suporte atingir patamares impraticáveis.

Para resolver estes e outros problemas das aplicações Cliente/Servidor tradicionais é que começaram a ser desenvolvidas aplicações para a Web ou aplicações baseadas em tecnologia Web, como preferem alguns. Vamos falar um pouco mais sobre aplicações Web.

Aplicações Web – um Novo Paradigma

Para resolver os tradicionais problemas das aplicações Cliente/Servidor é que surge o conceito de aplicação Web.

Para acessar uma aplicação Web, o único programa de que o usuário precisa é um Navegador instalado na sua estação. Existem inclusive aplicações Web bem projetadas que não obrigam o usuário a utilizar um Navegador específico, o que aumenta mais ainda a flexibilidade.

Uma aplicação Web reside no servidor, no nosso caso no IIS. Toda e qualquer alteração que se fizer necessária na aplicação será feita diretamente no servidor Web, sem que seja necessária nenhuma modificação nas estações dos usuários. Na próxima vez que o usuário acessar a aplicação, utilizando o Navegador instalado na sua estação, já estarão disponíveis as alterações efetuadas. Com este modelo não é necessária a “atualização” (que muitas vezes significa uma reinstalação) da aplicação em todas as estações da rede, cada vez que forem feitas alterações na aplicação. Com isso, a equipe de suporte fica dispensada da tediosa tarefa de atualização da aplicação em cada estação da rede.

O modelo de aplicações Web traz inúmeras vantagens, dentre as quais podemos destacar:

- ◆ Atualização das aplicações centralizada no servidor Web, sem a necessidade de atualizar todas as estações da rede que fazem acesso à aplicação.
- ◆ Facilidade de manutenção e suporte, uma vez que o cliente somente precisa de um Navegador para acessar a aplicação.
- ◆ Redução do chamado TCO – Total Cost Ownership (Custo Total de Propriedade). O TCO é uma medida de quanto custa, por ano, a manutenção de uma estação de rede em funcionamento. O Cálculo do TCO leva em consideração diversos fatores. Para maiores informações sobre o cálculo do TCO acesse o site do Gardner Group na Internet.

Porém nem tudo são vantagens. Pela natureza e implementação do modelo Web, o qual faz uso do protocolo HTTP, as aplicações Web não têm como manter uma conexão permanente com o usuário. Por isso que o modelo Web, muitas vezes, é conhecido como “State Less”, isto é, “Sem estado”. Isso acontece porque, após ter enviado a página solicitada pelo usuário, a conexão é encerrada, não sendo possível continuar trocando informações com o usuário, a não ser que uma nova conexão seja estabelecida. Esta é uma característica intrínseca do protocolo HTTP.

Para vencer as limitações impostas por este modelo “State Less”, o ASP.NET deu passos importantes, como a possibilidade da manutenção do estado dos diversos controles de um formulário Web, entre uma chamada e outra da página, conforme estudamos nos capítulos anteriores.

A possibilidade de diferenciar cada usuário que está acessando uma aplicação Web é de fundamental importância, para que possamos criar conteúdos personalizados e interativos, além da possibilidade de aplicar conceitos como níveis de permissão de acesso aos dados e auditoria de acesso.

O que é uma Aplicação Web no IIS?

Agora que já sabemos que o modelo de desenvolvimento baseado na Web é o modelo dominante para o desenvolvimento de novas aplicações, vamos aprender a identificar quais os elementos que compõem uma aplicação Web no IIS.

De uma maneira simples e didática poderíamos definir uma aplicação Web desenvolvida com ASP ou ASP.NET, como sendo: “Uma pasta virtual e todas as suas subpastas, juntamente com um conjunto de páginas ASP ou ASP.NET, componentes COM ou COM+ e Web Services, projetados para executar uma ou mais tarefas específicas, como por exemplo um sistema para controle do departamento de Recursos Humanos.”

O nosso conjunto de páginas ASP pode fazer uso dos diversos componentes disponíveis para a aplicação Web.

No IIS, o primeiro passo para criar uma aplicação Web é criar uma pasta, a qual será o ponto de partida para a aplicação. O próximo passo é transformar esta pasta em uma “Pasta Virtual” do servidor IIS.

Vamos supor que você tenha criado uma pasta chamada WebApI no drive C:\WebApI, de um servidor IIS chamado www.abc.com.br. Agora você registrou esta pasta como uma pasta virtual chamada WebApI. Dentro deste diretório você criou uma página ASP.NET chamada index.aspx, a qual será a página inicial da aplicação Web. Você lembra como seria o endereço para acessar a página index.asp do diretório virtual WebApI do servidor www.abc.com.br?

NOTA: Para saber como transformar uma pasta em uma “Pasta Virtual” do IIS, consulte a Introdução e o Capítulo 1 deste livro.

O endereço seria o seguinte: <http://www.abc.com.br/WebApI/index.aspx>

Dentro da pasta WebApI poderíamos criar outras pastas, conforme a necessidade da nossa aplicação Web. As pastas criadas dentro de uma pasta virtual já passam a ser acessíveis para o servidor Web. Por exemplo, se dentro da pastas WebApI, criarmos uma pasta chamada Segurança e dentro da pasta Segurança colocarmos um arquivo chamado login.aspx. O endereço para acessar o arquivo login.asp seria o seguinte: <http://www.abc.com.br/WebApI/Segurança/login.aspx>

Todos as subpastas da pasta WebApI também farão parte da aplicação Web. Com isso podemos concluir, em um primeiro momento, que uma aplicação Web do IIS está ligada à criação de uma pasta virtual no servidor IIS.

O próximo passo é configurar as propriedades da aplicação Web. Para isso utilizamos o “Gerenciador do Internet Services”, conforme ilustrado nos passos a seguir.

Para configurar as propriedades de uma aplicação Web faça o seguinte:

1. Faça o logon no Windows 2000 Server, com permissões de administrador.
2. Abra o Gerenciador do Internet Services: Iniciar -> Programas -> Ferramentas administrativas -> Gerenciador do Internet Services.
3. Surge a janela indicada na Figura 14.45.

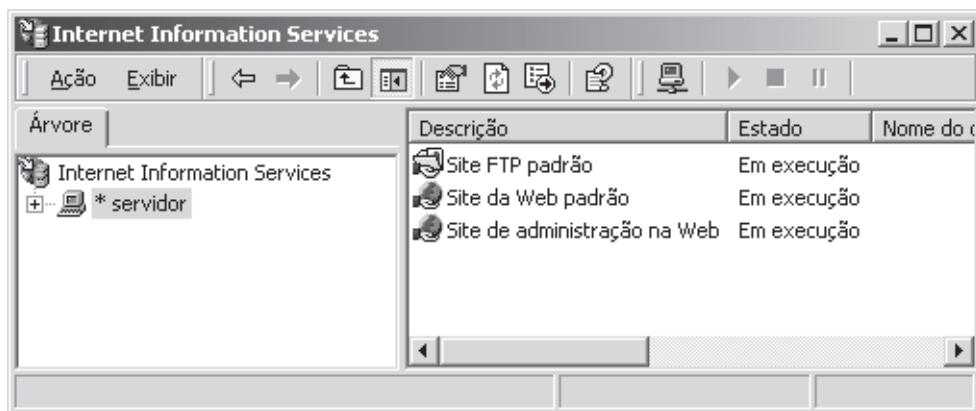


Figura 14.45: O Gerenciador do Internet Services.

4. Esta é a janela do console de administração do IIS 5.0.
5. Dê um clique duplo no nome do computador. No nosso exemplo o nome é Servidor.
6. Surgem as opções indicadas na Figura 14.46.

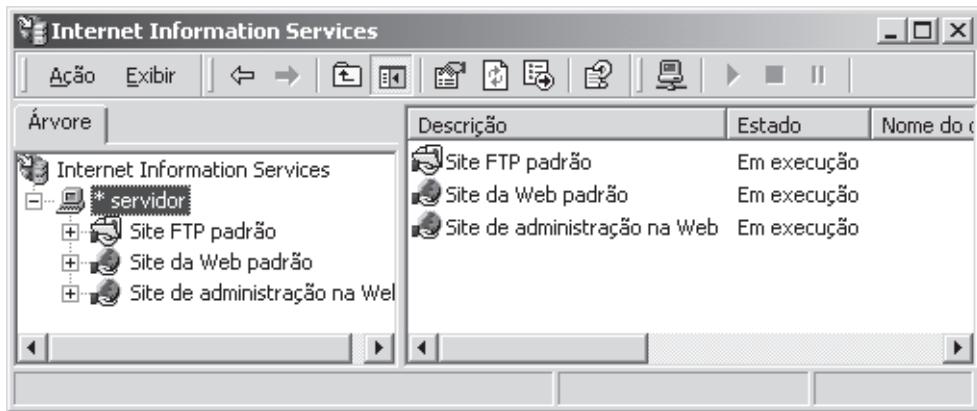


Figura 14.46: Opções de gerenciamento do IIS.

7. Todos os aplicativos Web estão disponíveis através da opção “Site da Web Padrão”. Dê um clique no sinal de + ao lado desta opção para abri-la.
8. Serão exibidas todas as pastas virtuais disponíveis no servidor IIS, conforme indicado na Figura 14.47.

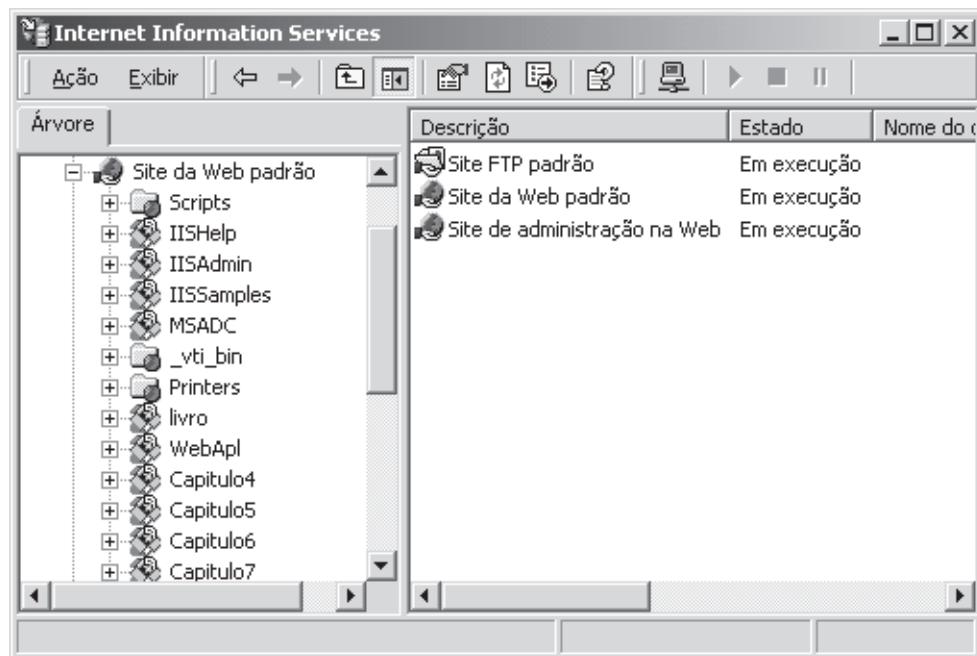


Figura 14.47: Pastas virtuais do servidor IIS.

9. No nosso exemplo queremos configurar as propriedades da pasta virtual WebApI. Neste ponto o conceito de pasta virtual confunde-se com o de aplicação Web. Na verdade a nossa aplicação WebApI está contida na pasta Virtual WebApI.

10. Dê um clique com o botão direito do mouse sobre WebApl.
11. No menu de opções que surge dê um clique em Propriedades.
12. Será exibida a janela “Propriedades de WebApl”, conforme indicado na Figura 8.4.

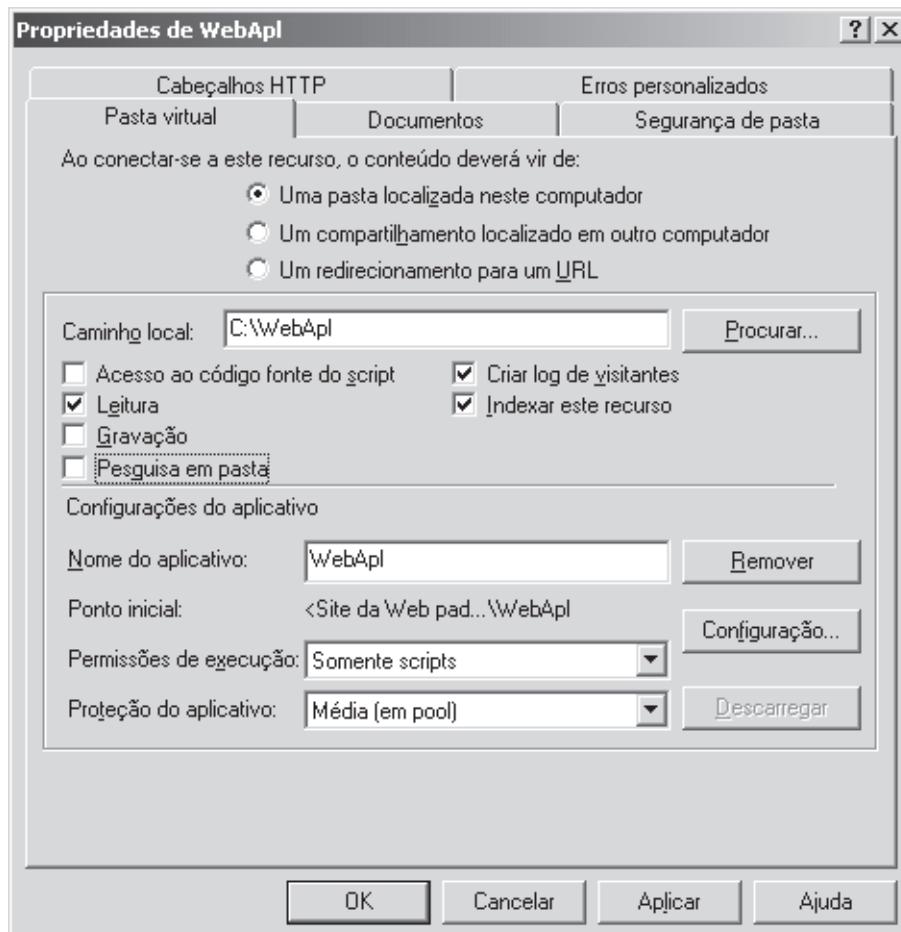


Figura 14.48: Propriedades da aplicação Web – WebApl.

Observe que nesta janela temos diversas informações, tais como:

- ◆ **Caminho da pasta:** C:\WebApl. Este é o caminho físico onde são gravados os elementos da aplicação Web. No nosso exemplo, é a pasta WebApl do drive C:
- ◆ **Permissões de acesso à pasta virtual da aplicação:** Observe que por padrão somente são marcadas as opções – Leitura, Criar log e Indexação.
- ◆ **Nome do aplicativo:** No nosso exemplo é WebApl.

A principal opção desta janela é o botão Configuração.

13. Dê um clique no botão Configuração.
14. Surge a janela Configuração de aplicativo, indicada na Figura 14.49.

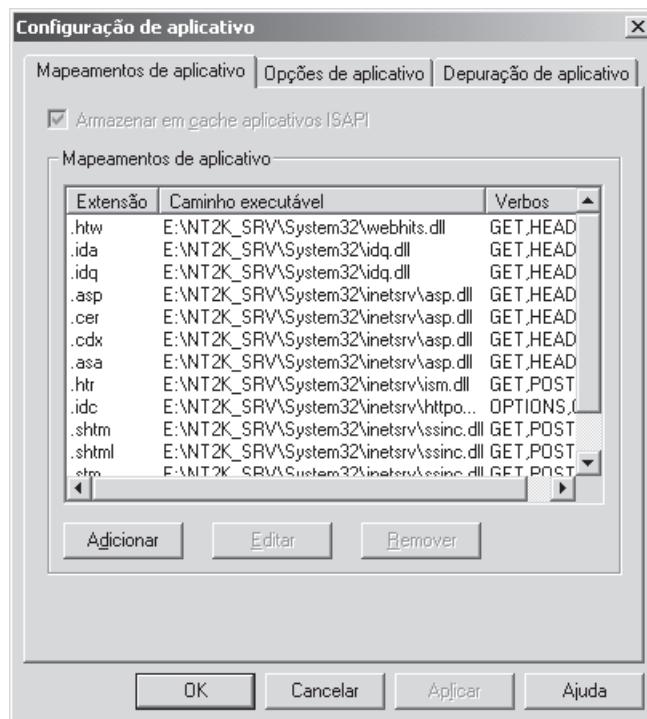


Figura 14.49: Configurando as propriedades da aplicação Web – WebApI.

Na primeira guia – Mapeamento de aplicativos, temos a indicação de qual componente do IIS irá processar cada requisição do usuário. Este mapeamento é baseado na extensão do arquivo. Por exemplo, toda página com a extensão .aspx será processada pela DLL aspnet_isapi.dll, conforme indicado pela Figura 14.50.

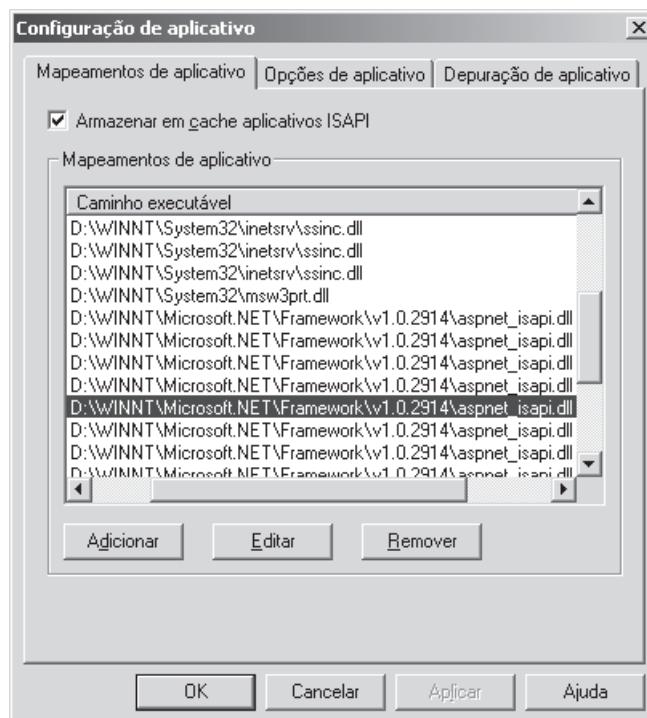


Figura 14.50: A DLL aspnet_isapi.dll é responsável pelo processamento das páginas .aspx.

15. Não iremos alterar nenhuma opção nesta guia.
16. Dê um clique na guia Opções de aplicativo.
17. Na segunda guia – Opções de aplicativo, temos diversas configurações importantes, conforme indicado na Figura 14.51. Estas opções afetam, principalmente, páginas ASP 3.0.

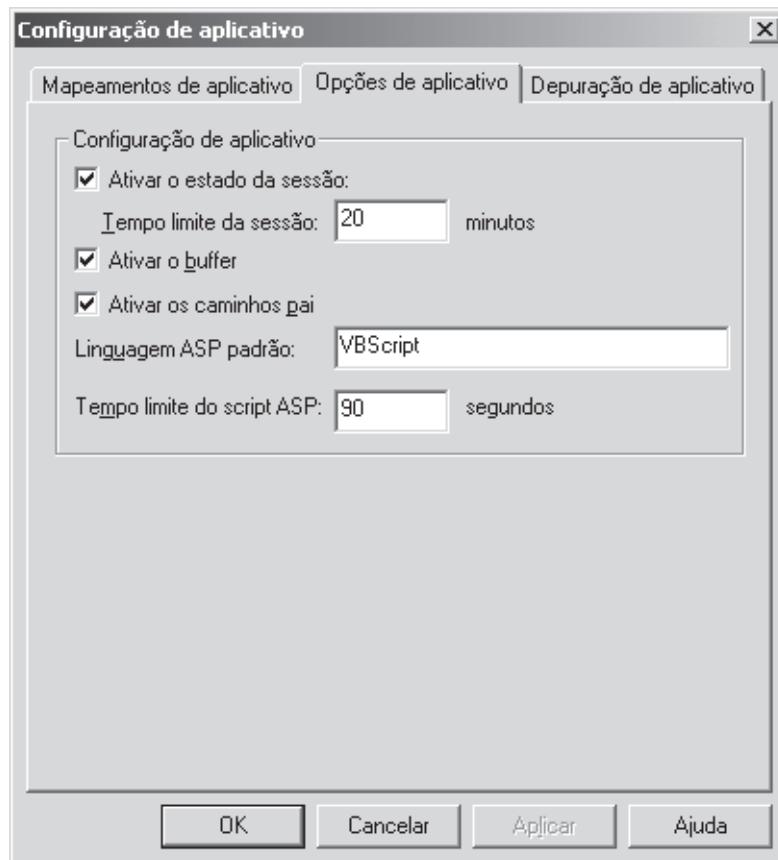


Figura 14.51: Opções da guia – Opções de aplicativo.

Abaixo temos a descrição destas opções:

- ◆ **Ativar o estado da seção:** Esta caixa afeta as páginas ASP 3.0. Use essa caixa de seleção para ativar ou desativar o estado da sessão. Quando o estado da sessão é ativado, o interpretador ASP cria uma sessão para cada usuário que acessa um aplicativo ASP, de modo que você possa identificar o usuário através das páginas do aplicativo. Quando o estado da sessão é desativado, o ASP não controla usuários e não permite que um script do ASP armazene informações no objeto Session ou use os eventos Session.OnStart ou Session.OnEnd. Uma sessão é finalizada automaticamente se o usuário não solicitou ou atualizou uma página em um aplicativo no fim do tempo limite – TimeOut.
- ◆ **Tempo limite da seção:** No nosso exemplo está definido em 20 minutos. Uma sessão é finalizada automaticamente se o usuário não solicitou ou atualizou uma página em um aplicativo por um tempo maior do que o tempo limite – TimeOut.
- ◆ **Ativar o Buffer:** Esta opção afeta as páginas ASP 3.0. Por padrão esta opção é ativada no IIS 5.0. Nas versões anteriores esta propriedade era desativada por padrão. Quando o Buffer está ativo, o resultado do processamento de uma página ASP somente é enviado para o Navegador do cliente, quando toda a página tiver sido processada, a menos que seja utilizado o método Flush do objeto Response.

- ◆ Ativar os caminhos Pai: Marque esta caixa de seleção para permitir que as páginas ASP usem os caminhos relativos do diretório pai do diretório atual (caminhos que usam a sintaxe ..), também conhecidos como endereços relativos.
- ◆ Linguagem ASP padrão: Por padrão é definida como sendo VBScript. Caso você deseje pode alterar para JScript. Também podemos especificar o nome de qualquer outra linguagem para a qual tenhamos o interpretador instalado no servidor IIS, como por exemplo Perl. Esta opção define a linguagem que será utilizada para interpretar os comandos dentro das tags <% e %>.
- ◆ Tempo limite do script ASP: Especifica o período de tempo durante o qual o ASP permitirá a execução de um script. Se a execução do script não terminar ao final do período de tempo limite, o ASP irá parar o script e gravar um evento no log de eventos do Windows 2000. Você pode definir o período de tempo limite como um valor entre 1 e 2147483647, além de poder substituir essa opção em método Server.ScriptTimeout.

IMPORTANTE: Se você ativar essa opção, não forneça acesso de execução aos diretórios pai; caso contrário, um script poderá tentar executar um programa não autorizado em um diretório pai.

18. Não iremos alterar nenhuma opção nesta guia.
19. Dê um clique na guia Depuração de aplicativo.
20. Na terceira guia – Depuração de aplicativo, temos diversas configurações relacionadas com a depuração de erros em páginas ASP, conforme indicado na Figura 14.52.

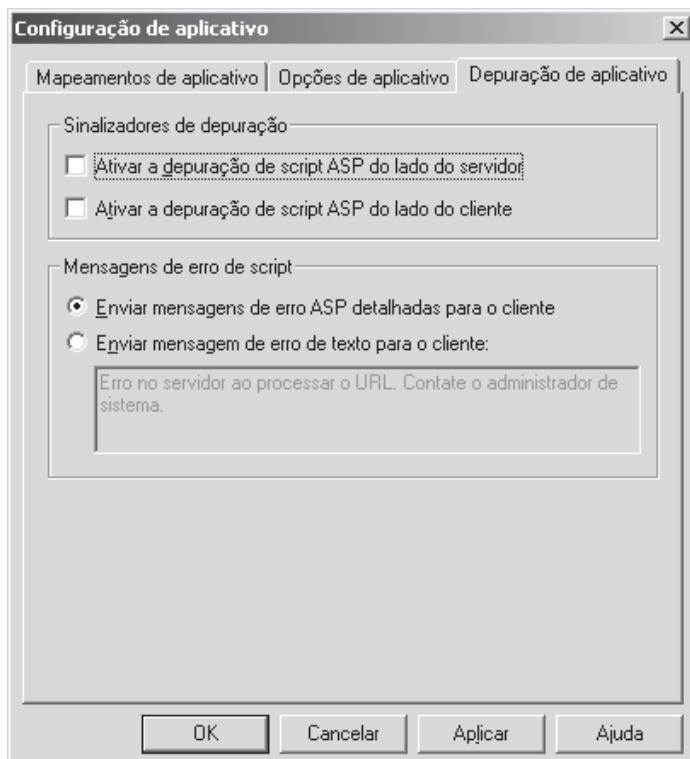


Figura 14.52: Opções da guia – Depuração de aplicativo.

Abaixo temos a descrição destas opções, que afetam, principalmente, páginas ASP 3.0:

- ◆ **Ativar a depuração de script ASP do lado do servidor:** Marque esta opção para permitir que o servidor Web entre no Depurador de scripts da Microsoft durante o processamento de páginas ASP. Em seguida, você pode usar o depurador para examinar seus scripts. Por questões de desempenho, a depuração do ASP não é recomendada em um ambiente de produção.
- ◆ **Ativar a depuração de script ASP do lado do cliente:** Essa caixa de seleção está reservada para uso futuro e não tem efeito sobre a versão atual do ASP – Esta é a informação contida na própria documentação do IIS 5.0.
- ◆ **Enviar mensagens de erro do ASP detalhadas para o cliente:** Selecione essa opção para enviar informações específicas de depuração (incluindo o nome do arquivo, a mensagem de erro e o número da linha) para o navegador. Principalmente quando estamos desenvolvendo nossas aplicações, é importante que esta opção esteja selecionada, pois ao testarmos uma página, se a mesma contiver um erro, o Navegador informa o número da linha onde está o erro, o que facilita a correção do script.
- ◆ **Enviar mensagem de erro de texto para o cliente:** Selecione esta opção para enviar uma mensagem de erro padrão ao navegador quando algum erro impedir o servidor Web de processar a página ASP. Uma mensagem de erro específica é gravada no log de erros. Você pode alterar a mensagem de erro padrão digitando uma nova mensagem na caixa de texto. Não é recomendada a utilização desta opção no ambiente de desenvolvimento, pelos motivos descritos no parágrafo anterior.

21. Não iremos alterar nenhuma opção nesta guia.
22. Dê um clique no botão OK para fechar a janela de configuração.
23. Dê um clique no botão OK para fechar a janela de propriedades da pasta virtual WebApL.
24. Você estará de volta ao Gerenciador do Internet Services.
25. Feche-o.

Com isso finalizamos os elementos de segurança ligados ao IIS e ao Windows 2000. Agora vamos estudar alguns aspectos de segurança, específicos das páginas ASP.NET.

Configurações de Segurança com o Arquivo Web.Config

Podem existir áreas de um site que não devam estar disponíveis para o público em geral; em outras palavras, existem situações em que pode ser necessário restringir o acesso a determinadas áreas de um site. Para definir tais restrições temos vários recursos, tanto do IIS, do Windows 2000 quanto do próprio ASP.NET. Neste tópico veremos algumas maneiras de restringir o acesso, utilizando um arquivo de configurações, para aplicações Web – Web.Config. Este é um arquivo que fica gravado na pasta raiz da aplicação Web e contém uma série de diretivas. Algumas destas diretivas são utilizadas para configurações de segurança.

Vamos iniciar o nosso estudo por um importante conceito: Impersonation.

Impersonation

Conforme descrevemos no início do capítulo, quando o usuário faz a requisição de uma página ASP.NET, a primeira coisa que acontece, antes de a página ser processada e retornada para o usuário, é a autenticação do usuário. Através da

autenticação, o IIS identifica o usuário que está fazendo a requisição. Para áreas de acesso público é utilizada a autenticação anônima, onde todos os usuários são autenticados utilizando a mesma conta: IUSR_NOME_DO_COMPUTADOR.

O mecanismo de Impersonation pode estar habilitado ou desabilitado. Por padrão este mecanismo está desabilitado. O mecanismo de Impersonation define se a requisição do usuário será executada utilizando a conta com a qual o usuário foi autenticado (IUSR_NOME_DO_COMPUTADOR para o caso de acesso anônimo, ou uma conta do Windows 2000 para outros tipos de autenticação), ou a conta System, que é uma conta local do servidor Windows 2000.

Este mecanismo pode parecer sem sentido prático, mas na verdade existe uma justificativa bastante plausível. Como as páginas ASP.NET são compiladas e mantidas em cache, pode acontecer situações onde o IIS precise gravar arquivos temporários, em áreas do disco nas quais a conta IUSR_NOME_DO_COMPUTADOR não tem permissões de acesso. Neste caso o IIS precisa “fazer de conta que está executando como se fosse o usuário System”, o qual tem permissões para as áreas para criação de arquivos temporários.

Se o mecanismo de Impersonation estiver habilitado e a página fizer parte de uma área do site, no qual o acesso anônimo está habilitado, a requisição será feita em nome do usuário configurado para o acesso anônimo, normalmente a conta IUSR_NOME_DO_COMPUTADOR. Se o mecanismo de Impersonation estiver desabilitado (que é o padrão), ao invés da conta configurada para o acesso anônimo, será utilizada a conta System.

Se o mecanismo de Impersonation estiver habilitado e a página fizer parte de uma área na qual o acesso anônimo não está habilitado, a requisição será feita em nome do usuário que está logado no Windows. No caso de uma rede com servidores Windows 2000 e clientes Windows 9x ou Windows 2000, o usuário, provavelmente, estará logado com uma conta de um domínio do Windows 2000. O mesmo acontece para o caso de o mecanismo de Impersonation estar habilitado. Observe que a mudança de contexto (Impersonation) para a conta System somente ocorre quando o acesso for feito a áreas nas quais o acesso anônimo é permitido.

Em qualquer uma das situações, após assumir a identidade de um determinado usuário, quer seja a conta de acesso anônimo, quer seja a conta System ou a conta com a qual o usuário está logado, será feita uma verificação se o usuário tem permissão para os recursos que ele requisitou. Se tiver permissão a página é compilada, executada e retornada para o cliente; caso contrário uma mensagem de erro será retornada. Quando falamos em permissões podem ser tanto permissões NTFS quanto as permissões configuradas no IIS, conforme descrevemos no início do capítulo.

O Arquivo Web.Config

O arquivo Web.Config é um arquivo de configuração que fica gravado na pasta raiz da aplicação Web. Por exemplo, no Capítulo 13, utilizamos o Visual Studio .NET, para criar a aplicação AppWebChap13. Na pasta raiz desta aplicação temos um arquivo chamado Web.Config, o qual contém diversas configurações para a aplicação. Neste tópico estaremos modificando algumas opções deste arquivo e testando o efeito das modificações.

Na Listagem 14.1 temos o arquivo Web.Config para a aplicação AppWebChap13.

Listagem 14.1 Arquivo Web.Config para a aplicação Web criada no Capítulo 13.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

<system.web>

    <!-- DYNAMIC DEBUG COMPILATION

        Set compilation debug="true" to enable ASPX debugging. Otherwise, setting
        this value to
            false will improve runtime performance of this application.

        Set compilation debug="true" to insert debugging symbols (.pdb information)
        into the compiled page. Because this creates a larger file that executes
        more slowly, you should set this value to true only when debugging and to
        false at all other times. For more information, refer to the documentation
        about
            debugging ASP.NET files.

    -->

    <compilation
        defaultLanguage="c#"
        debug="true"
    />

    <!-- CUSTOM ERROR MESSAGES

        Set mode="on" or "remoteonly" to enable custom error messages, "off" to
        disable. Add
            <error> tags for each of the errors you want to handle.

    -->

    <customErrors
        mode="Off"
    />

    <!-- AUTHENTICATION

        This section sets the authentication policies of the application. Possible modes
        are
            "Windows", "Forms",
            "Passport" and "None"
    -->

    <authentication mode="None" />
```

```
<!-- APPLICATION-LEVEL TRACE LOGGING

Application-level tracing enables trace log output for every page within an
application.

Set trace enabled="true" to enable application trace logging. If
pageOutput="true", the

    trace information will be displayed at the bottom of each page. Otherwise,
you can view the

    application trace log by browsing the "trace.axd" page from your web applica-
tion root.

-->

<trace

    enabled="false"
    requestLimit="10"
    pageOutput="false"
    traceMode="SortByTime"
    localOnly="true"
/>

<!-- SESSION STATE SETTINGS

By default ASP.NET uses cookies to identify which requests belong to a
particular session.

If cookies are not available, a session can be tracked by adding a session
identifier to the URL.

To disable cookies, set sessionState cookieless="true".

-->

<sessionState

    mode="InProc"
    stateConnectionString="tcpip=127.0.0.1:42424"
    sqlConnectionString="data source=127.0.0.1;user id=sa;password="
    cookieless="false"
    timeout="20"
/>

<!-- PREVENT SOURCE CODE DOWNLOAD

This section sets the types of files that will not be downloaded. As well
as entering

    a httpHandler for a file type, you must also associate that file type with
the aspnet_isapi.dll

    in the App Mappings property of the web site, or the file can be
downloaded.
```

It is recommended that you use this section to prevent your sources being downloaded.

```
-->

<httpHandlers>

    <add verb="*" path="*.vb"
type="System.Web.HttpNotFoundHandler, System.Web" />

    <add verb="*" path="*.cs"
type="System.Web.HttpNotFoundHandler, System.Web" />

    <add verb="*" path="*.vbproj"
type="System.Web.HttpNotFoundHandler, System.Web" />

    <add verb="*" path="*.csproj"
type="System.Web.HttpNotFoundHandler, System.Web" />

    <add verb="*" path="*.webinfo"
type="System.Web.HttpNotFoundHandler, System.Web" />

</httpHandlers>

<!-- GLOBALIZATION

      This section sets the globalization settings of the application.

-->

<globalization

    requestEncoding="utf-8"
    responseEncoding="utf-8"

/>

</system.web>
</configuration>
```

A estrutura básica do arquivo Web.Config é a seguinte:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

    <system.web>

        Configuração 1
        Configuração 2
        ...
        Configuração n

    </system.web>
</configuration>
```

Onde temos diversos formatos para as cláusulas Configuração 1, Configuração 2, ... , Configuração n. Neste tópico aprenderemos a utilizar cláusulas referentes a configurações de segurança.

Definindo o Tipo de Autenticação

Uma das configurações que podemos definir como arquivo Web.Config é o tipo de autenticação que será utilizado pela aplicação Web. Podemos definir um dos seguintes tipos:

- ◆ **Windows built-in authentication:** ASP.NET utiliza este tipo de autenticação em conjunto com a autenticação do IIS. Primeiro a autenticação é feita pelo IIS, utilizando um dos seguintes tipos: autenticação básica, digest, ou autenticação Integrada do Windows (NTLM). Quando a autenticação do IIS é finalizada ASP.NET utiliza a identidade do usuário autenticado para autorizar ou negar acesso às páginas e demais recursos da aplicação Web.
- ◆ **Passport-based authentication:** Este tipo de autenticação utiliza um serviço pago, disponibilizado pela Microsoft. Com este tipo de autenticação, o usuário se identifica uma única vez e, automaticamente, estará autenticado para todos os sites e aplicações que utilizam a autenticação Passport-based. O inconveniente é que o serviço é pago; maiores informações em www.passport.com/business.
- ◆ **Forms authentication:** Quando uma requisição é recebida e ainda não foi autenticada, isto é, o IIS não conhece o usuário que fez a requisição, esta é redirecionada para um formulário HTML, através da utilização de redireção HTTP client-side. Neste formulário o usuário fornece um username e senha para login e envia o formulário para o servidor, normalmente clicando em um botão “Login”. Se a aplicação autentica a requisição, o sistema cria um cookie que contém as credenciais necessárias à autenticação do usuário. O navegador do cliente envia o cookie em todas as futuras requisições; desta forma o usuário pode continuar acessando a aplicação, sem digitar um username e senha, enquanto o cookie estiver gravado no seu computador.

Para definir o tipo de autenticação, no arquivo Web.Config, utilizamos a seguinte sintaxe:

```
<authentication mode="" [Windows/Forms/Passport/None]>  
    outras opções de autenticação utilizadas para a aplicação  
</authentication>
```

No exemplo da Listagem 14.1, observe que foi definido o tipo None:

```
<authentication mode="None" />
```

Além disso foi utilizada a sintaxe alternativa, ou seja, ao invés do fechamento `</authentication>`, utilizamos simplesmente `</>`.

Definindo os Usuários e Grupos que Têm Permissão de Acesso à Aplicação

Para definir uma lista de usuários e grupos com permissão de acesso aos recursos da aplicação, utilizamos uma cláusula `<authorization> </authorization>`, para definir uma lista de usuários ou grupos, com permissão de acesso. A sintaxe para esta cláusula é a seguinte:

```

<authorization>

    <allow users='[lista de usuários, separados por vírgula]'

        roles='[lista de grupos, separados por vírgula]'

        verb=" [GET/POST/HEAD]">

    />

    <deny users='[lista de usuários, separados por vírgula]'

        roles='[lista de grupos, separados por vírgula]'

        verb=" [GET/POST/HEAD]">

    />

</authorization>

```

A definição de uma lista de usuários e grupos do Windows, com permissão de acesso, é mais comum em uma Intranet, onde temos que definir permissões de acesso somente para um grupo específico de usuários. Vamos supor que exista uma aplicação Web, com informações gerenciais sobre o desempenho dos funcionários, planos de promoção e carreira. Imagine que somente o Gerente de RH, diretores, vice-presidentes e o presidente devam ter acesso a essa aplicação. Esta é uma situação em que devemos utilizar a autenticação Windows built-in authentication e utilizar uma cláusula <authorization> </authorization> para definir quais os usuários que têm permissão de acesso.

Neste caso a primeira coisa a ser feita é definir o tipo de autenticação como Windows:

```
<authentication mode="Windows" />
```

- ◆ “allow users” contém uma lista de usuários e grupos com permissão de acesso à aplicação.
- ◆ “deny users” contém uma lista de usuários e grupos, para os quais a permissão de acesso é negada.

Ao especificarmos o nome de um usuário, devemos utilizar a nomenclatura completa, na qual é especificado o nome do domínio. Na máquina que estou utilizando, está instalado um domínio Windows 2000 chamado GROZA. Para especificar uma conta deste domínio utilizo a seguinte nomenclatura:

GROZA\jsilva

GROZA\user1

GROZA\user2

GROZA\Administrador

NOTA: Se a conta pertencer a um Member Server, isto é, um servidor que não faz parte de um domínio, basta substituir o nome do domínio pelo nome do servidor.

Podemos utilizar alguns caracteres especiais na lista de usuários ou grupos:

- ◆ * : Significa todos os usuários ou todos os grupos.
- ◆ ? : Significa acesso anônimo. Para o caso de estarmos utilizando Windows authentication, será interpretada como a conta configurada para o acesso anônimo normalmente a conta IUSR_NOME_DO_COMPUTADOR. Somente pode ser utilizada na lista de usuários – user.

Vamos considerar um exemplo simples, onde é permitido o acesso para três usuários do domínio GROZA e negado o acesso para todos os demais usuários:

```
<authorization>
    <allow users="GROZA\Usuário1,GROZA\Usuário2" />
    <deny users="*"/>
</authorization>
```

Neste exemplo os usuários “Usuário1” e “Usuário2” poderão acessar a aplicação e todos os demais terão o acesso negado.

Porém existem situações mais complexas. Pode acontecer de existir um arquivo Web.Config no diretório-raiz do site, um arquivo Web.Config na pasta-raiz da aplicação e um outro arquivo Web.Config em uma pasta dentro da aplicação Web. Podemos perfeitamente criar arquivos Web.Config nos locais descritos no exemplo. Neste caso temos a seguinte questão: “Quais as configurações que são efetivamente aplicadas?”. Se em um dos arquivos o usuário tem permissão e em outro não, qual será a permissão que irá prevalecer?

Para responder a estas questões, devemos levar em consideração algumas regras:

- ◆ As cláusulas `<allow>` e `<deny>` são mescladas, a partir de todos os arquivos Web.Config existentes no caminho de uma aplicação. Considere o exemplo da Figura 14.53.

Neste caso, quando a página “pagina1.aspx” for acessada, serão mescladas as cláusulas `<allow>` e `<deny>`, dos três arquivos Web.Config existentes, desde a pasta raiz do site, até chegar na pasta-onde está a página pagina1.aspx. Em caso de conflito é dada preferência para as configurações do arquivo Web.Config que está em um nível mais alto, isto é, mais próximo da pasta-raiz do site. Após processados todos os arquivos Web.Config do caminho, é montada uma lista única de usuários e grupos para as cláusulas `<allow>` e `<deny>`.

Uma vez montada a lista única vamos entender como são aplicadas as permissões. O processador do ASP.NET vai lendo a lista de cima para baixo e aplica a configuração que melhor se encaixar; em outras palavras, a configuração mais específica. Você pode ter notado no exemplo que apresentamos anteriormente, onde a permissão é atribuída a dois usuários:

```
<allow users="GROZA\Usuário1,GROZA\Usuário2" />
```

IMPORTANTE: As configurações de segurança do ASP.NET somente são aplicadas e verificadas para recursos associados ao ASP.NET; em outras palavras, a arquivos associados para processamento da DLL: `aspnet_isapi.dll`. As configurações de segurança do ASP.NET não se aplicam para recursos não associados a DLL `aspnet_isapi.dll`, como por exemplo arquivos .txt, html, gif, jpg, asp e outros tipos de arquivos. As limitações de acesso para este tipo de arquivo dependem das configurações de segurança do IIS e do Windows 2000. Poderíamos contornar esta situação, mapeando os demais recursos para a DLL `aspnet_isapi.dll`, porém isso teria um forte impacto em termos de performance, o que iria piorar muito o desempenho da aplicação.

e negada para todos:

```
<deny users="*"/>
```

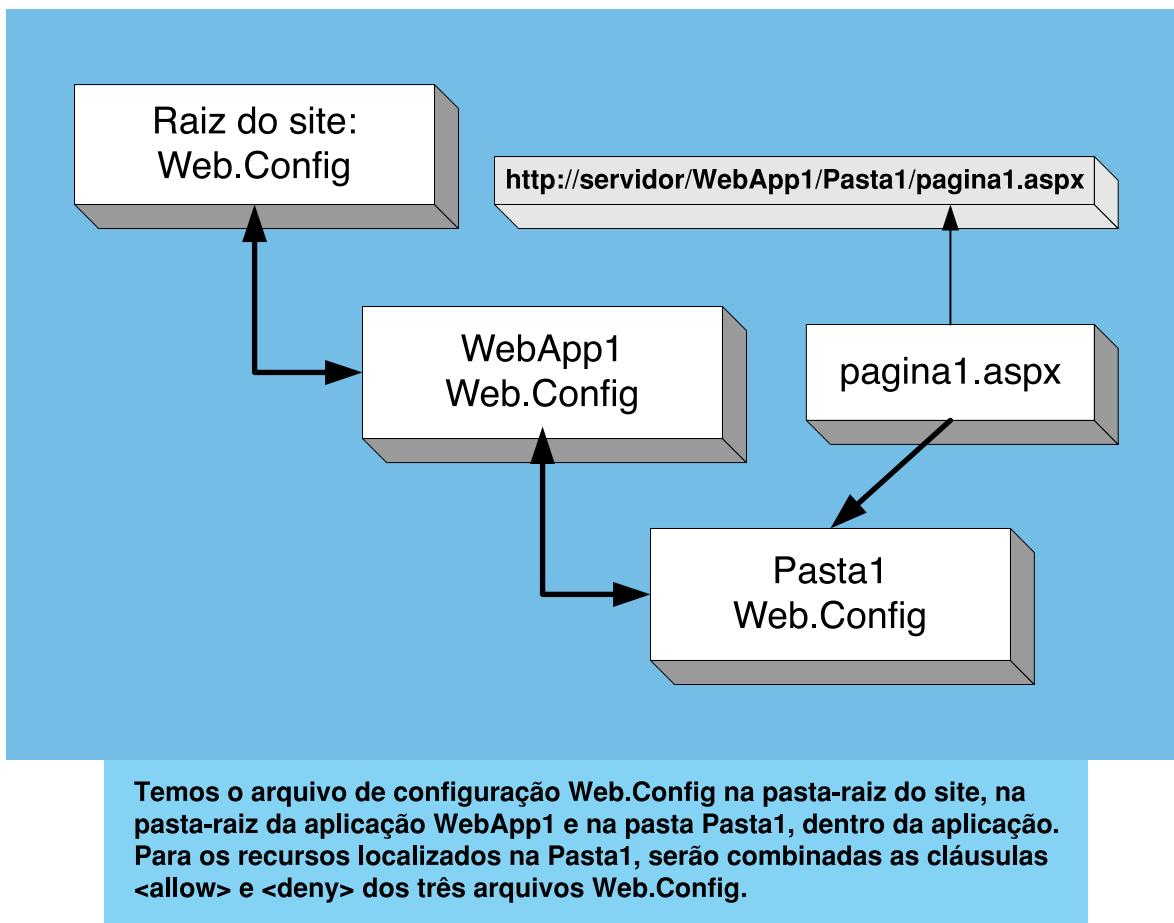


Figura 14.53: Vários arquivos Web.Config no caminho de uma aplicação.

Aqui precisamos fazer alguns comentários para que não haja confusão entre a maneira como as permissões são aplicadas com as cláusulas `<allow>` e `<deny>`, no arquivo Web.Config e a maneira como o Windows 2000 aplica permissões NTFS em pastas e arquivos.

Com as permissões NTFS, negar tem precedência sobre qualquer outra permissão. Então, se negarmos a permissão de acesso a uma pasta ou arquivo, para o grupo Todos (Everyone no Windows 2000 em inglês), ninguém terá acesso a pasta ou arquivo, independente de quantos usuários tenham permissões explícitas de acesso. Em resumo, nas permissões NTFS, negar tem precedência sobre permitir e as permissões são cumulativas; por exemplo, se um usuário tem permissão de acesso e o grupo ao qual ele pertence tem negada a permissão de acesso, o usuário herda o negar do grupo e, como negar tem precedência sobre permitir, o acesso será negado.

Já para as configurações do arquivo Web.Config, o comportamento é um pouco diferente. Será aplicada a permissão mais específica, ou seja, aquela que estiver mais especificamente definida. Vamos novamente nos reportar ao exemplo anterior. Os usuários:

GROZA\Usuário1

GROZA\Usuário2

têm permissão <allow> de acesso definida. O grupo todos (*) tem permissão negada - <deny>. Como os usuários têm permissão específica de acesso, estes poderão acessar a aplicação. Todos os demais usuários (* - GROZA\Usuário1 - GROZA\Usuário2) terão permissão de acesso negada. Mais uma vez cabe ressaltar que este comportamento é diferente do que temos com as permissões NTFS em relação a pastas e arquivos em partições NTFS.

Outro uso comum, onde precisamos combinar permissões <allow> e <deny>, é para casos em que um determinado grupo deve ter permissões de acesso e um ou mais elementos do grupo devem ter a permissão negada. Vamos considerar a seguinte situação de usuários e grupos para um domínio chamado GROZA:

```
Grupo gerentes Membros do grupo: user1
                                    user2
                                    user3
                                    user4
                                    user5
```

Nós queremos que o grupo gerente tenha permissão de acesso a uma aplicação Web, com exceção dos usuários: user2 e user5. Para implementar estas configurações de segurança, utilizamos a seguinte cláusula <authorization> </authorization>, no arquivo Web.Config da aplicação Web:

```
<authorization>
    <allow roles="GROZA\gerentes" />
    <deny users="GROZA\user2,GROZA\user5" />
</authorization>
```

Neste caso o grupo gerentes tem permissão de acesso e, para os usuários user2 e user5, tenho negado, explicitamente, o acesso. O resultado prático é que os usuários user1, user3 e user4 têm permissão de acesso e os usuários user2 e user5 não têm permissão de acesso. Observe que para permitir acesso a grupos utilizamos a cláusula <allow roles=.../> e para negar permissão a grupos, utilizamos a cláusula <deny roles=.../>.

A utilização da cláusula verb é opcional. Esta cláusula pode ser utilizada para definir o tipo de requisição que o usuário pode fazer. Conforme descrevemos anteriormente existem diferentes métodos de enviar os dados de um formulário para processamento no servidor. Os tipos mais comuns são “GET” e “POST”. Com a cláusula verb podemos limitar a forma de envio permitida para usuários ou grupos. Vamos a um exemplo prático.

Vamos definir que os usuários GROZA\user1 e GROZA\user2 devem ter permissão para utilizar o método GET e todos os demais usuários somente podem utilizar o método POST. Para implementar estas configurações, utilizamos a seguinte cláusula de autorização, no arquivo Web.Config:

```
<authorization>
    <allow verb="GET"      users="GROZA\user1,GROZA\user2" />
    <allow verb ="POST"   users="*"   />
    <deny verb = "GET"    users="*"   />
</authorization>
```

Observe que negamos o verbo GET para todos os usuários, de tal maneira que somente os usuários user1 e user2 tenham a permissão para utilizar GET. Para os demais usuários demos a permissão para utilizar POST.

Podem existir situações onde precisamos definir diferentes configurações de acesso, para uma página ou pasta da aplicação Web. Neste caso podemos utilizar a cláusula <location></location> para especificar a qual arquivo ou pasta as configurações devem ser aplicadas. No exemplo a seguir, aplicamos as configurações de acesso que se aplicam somente à página dados.aspx.

```
<location path="dados.aspx">
<system.web>
<authorization>
    <allow roles="GROZA\gerentes" />
    <deny users="GROZA\user2,GROZA\user5" />
</authorization>
</system.web>
</location>
```

Estas configurações somente se aplicam à página dados.aspx.

No Capítulo 15 veremos mais alguns detalhes sobre as configurações de segurança no ASP.NET.

Conclusão

Neste capítulo aprendemos sobre aspectos básicos de segurança, tais como:

- ◆ Tipos de autenticação.
- ◆ Permissões NTFS do Windows 2000.
- ◆ Opções de segurança do IIS.
- ◆ Permissões em nível de Banco de dados.

Muito existe a ser tratado sobre segurança. Livros inteiros já foram escritos sobre o assunto. Pela experiência de anos trabalhando com ambientes de Rede Locais e acesso a dados críticos posso afirmar com convicção: “O primeiro passo para estabelecer um ambiente seguro é a definição de uma política de segurança e a ampla divulgação da mesma, para que todos estejam conscientes de suas responsabilidades em relação à segurança.”

Muitas vezes cuida-se muito da segurança de acesso lógico aos dados, com a otimização das configurações de segurança do Sistema Operacional, do Servidor Web, das aplicações Web e do Servidor de Banco de dados. São gastos milhares de dólares em equipamentos e programas sofisticados para atuarem como Firewall. Investe-se em roteadores e switches modernos, com capacidades de filtragem de pacotes, detecção de tentativas de invasão e assim por diante. E muitas vezes a segurança física é esquecida. De que adianta toda esta segurança no acesso lógico se um desconhecido pode, facilmente, invadir a sala dos Servidores e sair com uma “meia dúzia” de fitas de Backup embaixo do braço?

Obviamente que o investimento em segurança de acesso lógico aos dados é necessário, porém o aspecto da segurança no acesso físico é igualmente importante. Pontos como estes devem ser definidos na política de segurança da empresa, a qual deve ser continuamente revisada para se adaptar às freqüentes mudanças no mundo da tecnologia.

Na parte final do capítulo aprendemos a utilizar o arquivo Web.Config para definir permissões de acesso a aplicações Web ou partes específicas da aplicação.

Como o próprio título sugere, este capítulo apresenta assuntos variados sobre o ASP.NET. Veremos uma série de exemplos e técnicas úteis na criação de páginas e aplicações Web com ASP.NET. Veremos assuntos que variam de classes do Framework .NET que tratam da requisição do usuário, passando por alguns exemplos práticos que fazem conexão com banco de dados, seguindo com o conceito de Code-Behind até questões relacionadas ao controle de segurança através do código ASP.NET.

Vamos iniciar o capítulo apresentando três exemplos práticos:

- ◆ Como limitar o controle de uma lista, com base no valor selecionado em outra lista.
- ◆ Utilizando o controle DataGridView para editar dados.
- ◆ Apresentaremos mais um exemplo prático, no qual estaremos exibindo, em uma página ASP.NET, dados de uma planilha do Excel. Este exemplo é importante, pois salienta a possibilidade de acessarmos dados das mais variadas fontes. Esta é uma situação comum nas empresas, nos dias atuais. Os dados estão “espalhados” nos mais diversos formatos. Com o .NET podemos criar aplicações que acessam dados das mais variadas fontes.

Em seguida apresentaremos o conceito de “Code Behind”, que é uma técnica utilizada para facilitar a separação entre o código responsável pela lógica e o código responsável pela apresentação da página.

Seguindo na apresentação da nossa “Caixa de Ferramentas”, falaremos sobre dois importantes objetos: `HttpRequest` e `HttpResponse`. Com estes objetos podemos ter um controle mais eficaz sobre a requisição que é enviada pelo navegador do cliente e sobre a resposta que é enviada de volta pelo servidor.

Também iremos tratar sobre as diretivas de página. Uma diretiva é, geralmente, incluída no início da página e é utilizada para instruir o servidor sobre como efetuar o processamento da página. Por exemplo, podemos utilizar uma diretiva `@OutputCache`, para controlar a maneira como o servidor fará o cache das páginas ASP.NET.

O nosso próximo e final assunto será sobre as configurações de segurança através da utilização de código. Veremos uma série de pontos sobre segurança, utilizando código ASP.NET e classes do Framework .NET.

Este capítulo apresenta assuntos variados, mas de grande utilização na construção de aplicações Web. O objetivo dos exemplos apresentados é salientar técnicas que você provavelmente utilizará nas aplicações que estiver construindo.

Para os exemplos apresentados, utilizarei diversos conceitos e técnicas que foram apresentados nos capítulos anteriores, de tal forma que não irei explicá-los novamente. Caso você tenha alguma dúvida em relação aos comandos ou técnicas utilizados consulte os capítulos iniciais do livro, principalmente os Capítulos 10, 11 e 12 quando tratamos do acesso a bancos de dados, utilizando as classes do ADO.NET.

Criação de Listas Dinâmicas

Neste exemplo iremos construir um formulário com três controles do tipo DropDownList, conforme indicado a seguir:

- ◆ **LlistaPaises:** Este controle irá exibir uma lista dos países para os quais existem pedidos, a partir da tabela Pedidos do banco de dados NorthWind.mdb.
- ◆ **LlistaCidades:** Este controle irá exibir uma lista das cidades para as quais existem pedidos, a partir da tabela Pedidos do banco de dados NorthWind.mdb. Quando a página é carregada pela primeira vez, esta lista está vazia. Quando o usuário seleciona um país na lista de países, a página é recarregada e são exibidas as cidades para o país selecionado.
- ◆ **LlistaPedidos:** Este controle irá exibir uma lista dos pedidos para a cidade selecionada na lista de cidades. Quando a página é carregada pela primeira vez, esta lista está vazia. Quando o usuário seleciona um país, a página é recarregada e são exibidas as cidades para o país selecionado. A lista de pedidos continua vazia. Quando o usuário seleciona uma cidade, na lista de cidades, a página é recarregada e são exibidos os pedidos para a cidade selecionada. Quando o usuário seleciona um pedido na lista, são exibidas informações sobre o pedido. Para exibir as informações sobre o pedido, utilizamos diversos controles do tipo TextBox.

Na Listagem 15.1 temos o código para o exemplo proposto. Após a listagem apresentaremos mais alguns comentários sobre o exemplo.

Listagem 15.1 – Estrutura básica para a criação de um Web Service.

```
<%@ Page Language="C#" Debug="true" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>

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

// Declaro variáveis que serão globais para a página.

OleDbDataAdapter MeuComando;
String auxSQL1;
String auxSQL2;
String comandoSQL;
```

```

DataSet ds = new DataSet();
OleDbConnection MinhaConexão;
String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
    "DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";

protected void Page_Load(Object Src, EventArgs E )
{
    if (!Page.IsPostBack)
    {

        OcultaControles();
        String auxSQL1;
        String auxSQL2;
        String comandoSQL;

        auxSQL1 = "Select PaísDeDestino From Pedidos";
        auxSQL2 = " Group By PaísDeDestino Order By PaísDeDestino";

        comandoSQL = auxSQL1+auxSQL2;
        MinhaConexão = new OleDbConnection(DefineConexão);

        MeuComando = new OleDbDataAdapter(comandoSQL, MinhaConexão);
        MeuComando.Fill(ds);

        DataView source = new DataView(ds.Tables[0]);

        ListaPaises.DataSource = source ;
        ListaPaises.DataBind();
        ListaPaises.Items.Insert(0,"");

    }
}

void OcultaControles()
{
    txtNúmeroDoPedido.Visible=false;
}

```

```
txtCódigoDoCliente.Visible=false;
txtDataDoPedido.Visible=false;
txtFrete.Visible=false;
txtNomeDoDestinatário.Visible=false;
txtPaísDeDestino.Visible=false;
txtCidadeDeDestino.Visible=false;
}

void PaísSelecionado(Object sender, EventArgs e)
{
    if (ListaPaises.SelectedItem.Value!=>>>)
    {
        OcultaControles();
        auxSQL1 = «Select CidadeDeDestino From Pedidos Where PaísDeDestino='>>>';
        auxSQL2 = ListaPaises.SelectedItem.Value + `` Group By CidadeDeDestino
Order
By CidadeDeDestino»;

        comandoSQL = auxSQL1+auxSQL2;

        MinhaConexão = new OleDbConnection(DefineConexão);
        MeuComando = new OleDbDataAdapter(comandoSQL, MinhaConexão);

        MeuComando.Fill(ds);

        DataView source = new DataView(ds.Tables[0]);
        ListaCidades.DataSource=source;
        ListaCidades.DataBind();
        ListaCidades.Items.Insert(0,"");
        ListaPedidos.SelectedIndex=0;
    }
}

void CidadeSelecionada(Object sender, EventArgs e)
```

```

{
    if (ListaCidades.SelectedItem.Value!="")
    {
        OcultaControles();
        auxSQL1 = "Select * From Pedidos Where CidadeDeDestino=''";
        auxSQL2 = ListaCidades.SelectedItem.Value + " Order By NúmeroDoPedido";

        comandoSQL = auxSQL1+auxSQL2;

        MinhaConexão = new OleDbConnection(DefineConexão);
        MeuComando = new OleDbDataAdapter(comandoSQL, MinhaConexão);

        MeuComando.Fill(ds);
        DataView source = new DataView(ds.Tables[0]);

        ListaPedidos.DataSource=source;
        ListaPedidos.DataBind();
        ListaPedidos.Items.Insert(0,"");
    }
}

void PedidoSelecionado(Object sender, EventArgs e)
{
    if (ListaCidades.SelectedItem.Value!="")
    {
        auxSQL1 = "Select * From Pedidos Where NúmeroDoPedido=";
        auxSQL2 = ListaPedidos.SelectedItem.Value + " Order By NúmeroDoPedido";

        comandoSQL = auxSQL1+auxSQL2;

        MinhaConexão = new OleDbConnection(DefineConexão);
        MeuComando = new OleDbDataAdapter(comandoSQL, MinhaConexão);

        MeuComando.Fill(ds);
    }
}

```

```
        DataView source = new DataView(ds.Tables[0]);
        DataTable Pedidos = ds.Tables[0];

        DataRow Linha = Pedidos.Rows[0];

        txtNúmeroDoPedido.Visible=true;
        txtCódigoDoCliente.Visible=true;
        txtDataDoPedido.Visible=true;
        txtFrete.Visible=true;
        txtNomeDoDestinatário.Visible=true;
        txtPaísDeDestino.Visible=true;
        txtCidadeDeDestino.Visible=true;

        txtNúmeroDoPedido.Text      = Linha["NúmeroDoPedido"].ToString();
        txtCódigoDoCliente.Text     = Linha["CódigoDoCliente"].ToString();
        txtDataDoPedido.Text        = Linha["DataDoPedido"].ToString();
        txtFrete.Text               = Linha["Frete"].ToString();
        txtNomeDoDestinatário.Text  = Linha["NomeDoDestinatário"].ToString();
        txtPaísDeDestino.Text       = Linha["PaísDeDestino"].ToString();
        txtCidadeDeDestino.Text     = Linha["CidadeDeDestino"].ToString();
    }

}

</script>

<body>

<h3>

<font face="Verdana">

Selecione um País na lista de países.

<BR>

Em seguida uma cidade na lista de cidades.

<BR>

E por último, um pedido na lista de Pedidos.

</font>

</h3>
```

<HR>

```
<form runat=server>

<div align="left">
    <table border="0">

        <tr>
            <td>
                <p align="right">
                    <B>Selecione um País ->></B>
                </p>
            </td>

            <td>
                <asp:DropDownList
                    id="ListaPaises"
                    runat="server"
                    BackColor="#c0c0c0"
                    Font-Bold="True"
                    DataTextField = "PaísDeDestino"
                    DataValueField = "PaísDeDestino"
                    AutoPostBack = "True"
                    onSelectedIndexChanged = "PaísSeleccionado"
                >
                </asp:DropDownList>
            </td>
        </tr>
        <tr>
            <td>
                <p align="right">
                    <B>Selecione uma Cidade ->></B>
                </p>
            </td>

            <td>
                <asp:DropDownList
                    id="ListaCidades"
                    runat="server"

```

```
        BackColor="#c0c0c0"
        Font-Bold="True"
        DataTextField = "CidadeDeDestino"
        DataValueField = "CidadeDeDestino"
        AutoPostBack = "True"
        onSelectedIndexChanged = "CidadeSelecionada"

    >
</asp:DropDownList>
</td>
</tr>
<tr>
<td>
<p align="right">
<B>Selecione um Pedido ->></B>
</td>

<td>
<asp:DropDownList
    id="ListaPedidos"
    runat="server"
    BackColor="#c0c0c0"
    Font-Bold="True"
    DataTextField = "NúmeroDoPedido"
    DataValueField = "NúmeroDoPedido"
    AutoPostBack = "True"
    onSelectedIndexChanged = "PedidoSelecionado"
    >
</asp:DropDownList>
</td>
</tr>
<tr>
<td>
<p align="right">
<B>Número do Pedido:</B>
</td>
<td>
<asp:TextBox
```

```
        runat=server
        id="txtNúmeroDoPedido"
        Text=""
        TextMode="SingleLine"
        Font_Face="Arial" Font_Size="3"
        Height="20"
        Width="200"
        Enabled="False"
    />
</td>
</tr>
<tr>
<td>
    <p align="right">
        <B>Código do Cliente:</B>
    </td>
    <td>
        <asp:TextBox
            runat=server
            id="txtCódigoDoCliente"
            Text=""
            TextMode="SingleLine"
            Font_Face="Arial" Font_Size="3"
            Height="20"
            Width="200"
            Enabled="False"
        />
    </td>
</tr>
<tr>
    <td>
        <p align="right">
            <B>Data do Pedido:</B>
        </td>
        <td>
            <asp:TextBox
```

```
        runat=server
        id="txtDataDoPedido"
        Text=""
        TextMode="SingleLine"
        Font_Face="Arial" Font_Size="3"
        Height="20"
        Width="200"
        Enabled="False"
    />
</td>
</tr>

<tr>
    <td>
        <p align="right">
            <B>Frete:</B>
        </td>
        <td>
            <asp:TextBox
                runat=server
                id="txtFrete"
                Text=""
                TextMode="SingleLine"
                Font_Face="Arial" Font_Size="3"
                Height="20"
                Width="200"
                Enabled="False"
            />
        </td>
    </tr>
    <tr>
        <td>
            <p align="right">
                <B>Nome do Destinatário:</B>
            </td>
        <td>
            <asp:TextBox
```

```
        runat=server
        id="txtNomeDoDestinatário"
        Text=""
        TextMode="SingleLine"
        Font_Face="Arial" Font_Size="3"
        Height="20"
        Width="200"
        Enabled="False"
    />
</td>
</tr>
<tr>
    <td>
        <p align="right">
            <B>País de Destino:</B>
        </td>
        <td>
            <asp:TextBox
                runat=server
                id="txtPaísDeDestino"
                Text=""
                TextMode="SingleLine"
                Font_Face="Arial" Font_Size="3"
                Height="20"
                Width="200"
                Enabled="False"
            />
        </td>
    </tr>
    <tr>
        <td>
            <p align="right">
                <B>Cidade de Destino:</B>
            </td>
            <td>
                <asp:TextBox
                    runat=server
```

```
        id="txtCidadeDeDestino"  
  
        Text=""  
  
        TextMode="SingleLine"  
  
        Font_Face="Arial" Font_Size="3"  
  
        Height="20"  
  
        Width="200"  
  
        Enabled="False"  
  
    />  
  
    </td>  
  
</tr>  
  
</table>  
</div>  
  
</form>  
  
</body>  
</html>
```

Digite o código da Listagem 15.1 e salve o mesmo em um arquivo chamado chap15ex1.aspx, na pasta chap15, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap15/chap15ex1.aspx>

Ao carregar a página você obtém o resultado indicado na Figura 15.1. Inicialmente apenas a lista de países está preenchida, conforme destacado na Figura 15.1.

Esta lista é preenchida com o auxílio do evento Page_Load, onde estabelecemos uma conexão com o banco de dados NorthWind.mdb e retornamos a lista de países da tabela Pedidos. Por último adicionamos um elemento em branco (“”), como primeiro elemento da lista de Países. Este elemento é adicionado para que nenhum país esteja selecionado por padrão, ou seja, o usuário deve abrir a lista e selecionar um país:

```
ListaPaises.Items.Insert(0,"");
```

Na lista de países selecione Brasil. Ao selecionar um país na lista, será disparado o evento onSelectedIndexChanged, do controle ListaPaises. Em resposta a este evento, configuramos o procedimento “PaísSelecionado”:

```
onSelectedIndexChanged = "PaísSelecionado"
```

O procedimento PaísSelecionado faz uma conexão com o banco de dados NorthWind.mdb e retorna apenas as cidades para o país selecionado. Os dados retornados são atribuídos ao controle ListaCidades, através do uso da propriedade DataSource e do método DataBind. Por último adicionamos um elemento em branco (“”), como primeiro elemento da lista de cidades. Este elemento é adicionado para que nenhuma cidade seja selecionada por padrão, ou seja, o usuário deve abrir a lista e selecionar uma cidade. Abra a lista de cidades e selecione São Paulo, conforme indicado na Figura 15.2.

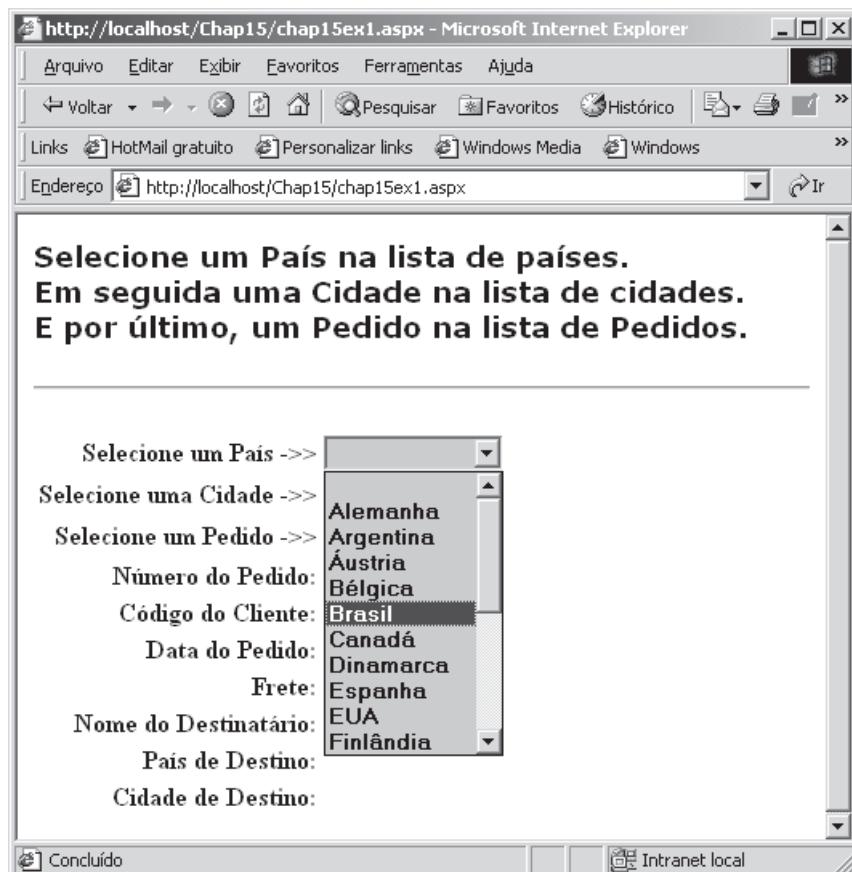


Figura 15.1: Lista de países já preenchida.

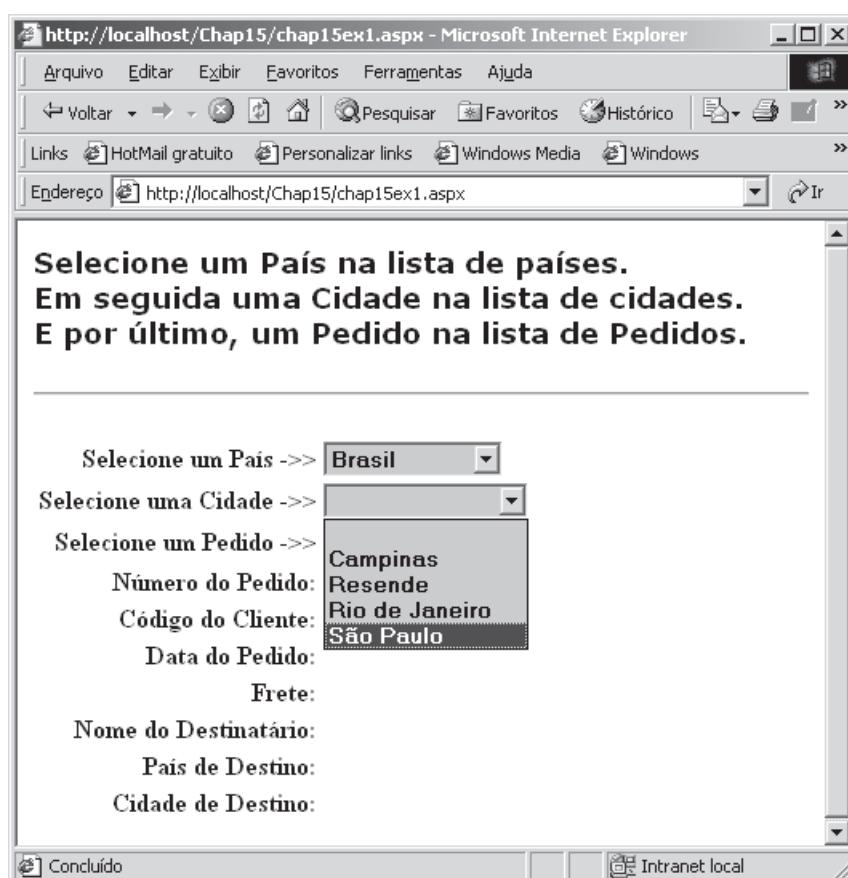


Figura 15.2: Selezionando uma cidade.

Ao selecionar uma cidade na lista de cidades, será disparado o evento `onSelectedIndexChanged`, do controle `ListaCidades`. Em resposta a este evento, configuramos o procedimento “`CidadeSelecionada`”:

```
onSelectedIndexChanged = "CidadeSelecionada"
```

O procedimento `CidadeSelecionada` faz uma conexão com o banco de dados `NorthWind.mdb` e retorna apenas os pedidos para a cidade selecionada. Os dados retornados são exibidos nos respectivos controles `TextBox` da página. O procedimento `CidadeSelecionada` também torna a propriedade `Visible`, dos controles `TextBox`, igual a `True`, de tal forma que estes controles sejam exibidos na página. Selecione o pedido número: 10494; os dados para o pedido selecionado serão exibidos, conforme indicado na Figura 15.3.

Se você selecionar uma outra cidade, os controles `TextBox` são ocultados e a lista de pedidos será redefinida para que sejam exibidos os pedidos para a cidade selecionada. Isto é feito pelo procedimento `CidadeSelecionada`, o qual é disparado toda vez que uma nova cidade for selecionada. Se você selecionar um novo país, a lista de cidades será redefinida para exibir as cidades do país selecionado e o controle `ListaPedidos` será desabilitado. Quando você seleciona uma cidade, na lista de cidades, o controle `ListaPedidos` é habilitado e passa a exibir os pedidos para a cidade selecionada.

Neste exemplo vimos algumas técnicas interessantes, as quais possibilitam a criação de formulários para pesquisas com um ou mais critérios. O grande “segredo” deste exemplo é a correta utilização da propriedade `onSelectedIndexChanged`, do controle `DropDownList`.

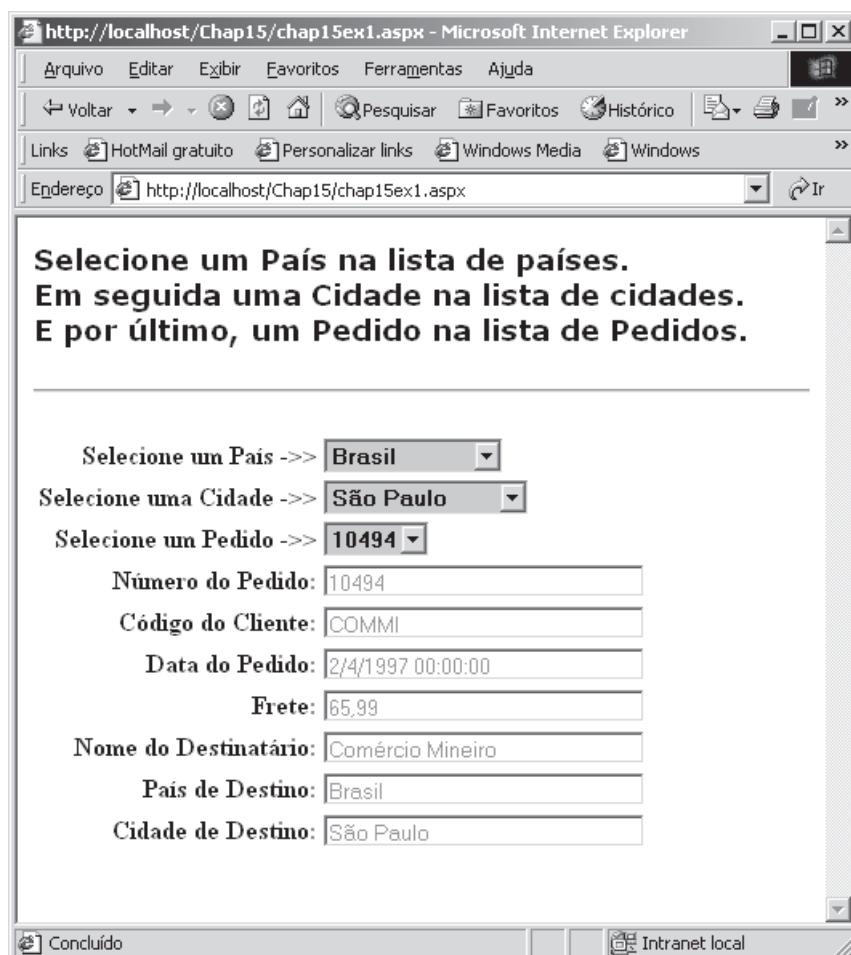


Figura 15.3: Exibindo os dados do pedido selecionado.

Edição de Dados com o Controle DataGrid

Nos exemplos dos capítulos anteriores, utilizamos o controle DataGrid para exibir dados em uma página ASP.NET. Também podemos utilizar o controle DataGrid para fazer a edição dos dados. No exemplo que apresentaremos neste tópico, utilizaremos um controle DataGrid que acessa dados da tabela authors, do banco de dados pubs do SQL Server 2000. Veremos como permitir que os dados sejam editados e que estas edições sejam enviadas de volta para o banco de dados.

Na Listagem 15.2 temos o código para o exemplo proposto. Após a listagem apresentaremos mais alguns comentários.

Listagem 15.2 – Edição de dados como DataGrid.

```
<%@ Page Language="C#" Debug="true" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

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

void BuscaDados()
{
    // Crio uma conexão com o banco de dados pubs localizado no servidor local.
    // Vamos acessar a instância SERVIDOR\NETSDK.

    SqlConnection myConnection = new SqlConnection("server=SERVIDOR\NETSDK;" +
        " uid=sa;pwd=;database=pubs");

    // Conectamos com o banco de dados utilizando um comando SQL,
    // o qual retorna todos os dados da tabela "Authors", do banco de
    // dados pubs.

    SqlDataAdapter myCommand = new SqlDataAdapter("SELECT " +
        " au_id,au_fname,au_lname,phone FROM Authors Order By au_fname",
        myConnection);

    // Criamos e preenchemos um objeto DataSet.

    DataSet ds = new DataSet();
    myCommand.Fill(ds);
}
```

```
// Conectamos um controle DataGrid com o DataSet criado anteriormente.  
// MyDataGrid é o id (nome) de um controle do tipo  
// DataGrid que está na seção de apresentação da página.  
  
DataView source = new DataView(ds.Tables[0]);  
MinhaGrade.DataSource = source ;  
MinhaGrade.DataBind();  
  
}  
  
void Page_Load(Object sender, EventArgs e)  
{  
if (!Page.IsPostBack)  
{  
    txtTitulo.Text="Clique em Editar para alterar os dados de uma linha!";  
    BuscaDados();  
}  
}  
  
void FazEdicao(Object sender, DataGridCommandEventArgs e)  
{  
    txtTitulo.Text="Clique em Atualizar para Confirmar as alterações!";  
    txtTitulo2.Text="ou Clique em Cancelar para descartar as alterações!";  
    MinhaGrade.EditRowIndex = e.Item.ItemIndex;  
    BuscaDados();  
}  
  
void CancelaAlteracoes(Object sender, DataGridCommandEventArgs e)  
{  
    txtTitulo.Text="Clique em Editar para alterar os dados de uma linha!";  
    txtTitulo2.Text="";  
    MinhaGrade.EditRowIndex = -1;  
    BuscaDados();  
}  
  
void FazUpdate(Object sender, DataGridCommandEventArgs e)  
{  
    // Crio uma conexão com o banco de dados pubs localizado no servidor local.  
    // Vamos acessar a instância SERVIDOR\NETSDK.
```

```
SqlConnection myConnection = new SqlConnection("server=SERVIDOR\\NETSDK;" +  
    " uid=sa;pwd=;database=pubs");  
  
// Conectamos com o banco de dados utilizando um comando SQL,  
// o qual retorna todos os dados da tabela "Authors", do banco de  
// dados pubs.  
  
SqlDataAdapter MeuDataAdapter = new SqlDataAdapter("SELECT " +  
    " au_id,au_fname,au_lname,phone FROM Authors Order By au_fname",  
myConnection);  
  
// Criamos e preenchemos um objeto DataSet.  
  
DataSet ds = new DataSet();  
MeuDataAdapter.Fill(ds,"authors");  
  
DataView source = new DataView(ds.Tables[0]);  
  
DataTable authors = ds.Tables[0];  
  
// Vamos definir o campo au_id como sendo a chave  
// primária da tabela Autores.  
  
authors.PrimaryKey = new DataColumn[] {authors.Columns["au_id"]};  
  
// Atribuo o valor dos campos da linha que está sendo editada,  
// a variáveis do tipo String, pois todos os campos são do tipo String.  
  
    string txtCodAutor = ((TextBox)e.Item.Cells[0].Controls[0]).Text;  
    string txtNome      = ((TextBox)e.Item.Cells[1].Controls[0]).Text;  
    string txtSobreNome = ((TextBox)e.Item.Cells[2].Controls[0]).Text;  
    string txtTelefone  = ((TextBox)e.Item.Cells[3].Controls[0]).Text;  
  
DataRow Linha = authors.Rows.Find(txtCodAutor);  
  
// Agora vamos editar o valor da linha.
```

```
Linha.BeginEdit();

    Linha["au_id"] = txtCodAutor;
    Linha["au_fname"] = txtNome;
    Linha["au_lname"] = txtSobreNome;
    Linha["phone"] = txtTelefone;

Linha.EndEdit();

// Agora crio os comandos necessários para enviar
// as alterações/inclusões/exclusões para o banco
// de dados pubs, no servidor SQL Server 2000.

// Em primeiro lugar crio um objeto do tipo SqlCommandBuilder

SqlCommandBuilder CriaComando = new SqlCommandBuilder(MeuDataAdapter);

// Agora defino a propriedade UpdateCommand do objeto MeuDataAdapter.

MeuDataAdapter.UpdateCommand= CriaComando.GetUpdateCommand();

// Chamo o método Update do objeto DataAdapter.

MeuDataAdapter.Update(ds,"authors");

MinhaGrade.DataSource = source ;
MinhaGrade.DataBind();
}

</script>

<body>

<form runat=server>

<h3><font face="Verdana">DataGrid para edição de dados.</font></h3>
<HR>

<asp:Label
```

```
    id="txtTitulo"
    Font-Bold="True"
    runat="server"
    >
</asp:Label>

<BR>

<asp:Label
    id="txtTitulo2"
    Font-Bold="True"
    runat="server"
    >
</asp:Label>

<HR>

<asp:DataGrid
    id="MinhaGrade"
    BorderColor="black"
    BorderWidth="1"
    CellPadding="3"
    AutoGenerateColumns="false"
    runat="server"
    EditItemStyle-BackColor="#c0c0c0"
    EditItemStyle-ForeColor="blue"
    OnEditCommand="FazEdicao"
    OnUpdateCommand="FazUpdate"
    OnCancelCommand="CancelaAlteracoes"
    >

    <HeaderStyle
        BackColor="#00aaaa"
        Font-Bold="True"
        Font-Name="Courier New"
        ForeColor="White"
        >
```

```
</HeaderStyle>

<Columns>

    <asp:BoundColumn
        HeaderText="Código"
        DataField="au_id"
        ItemStyle-BackColor="#c0c0c0"
    >

    </asp:BoundColumn>

    <asp:BoundColumn
        HeaderText="Nome"
        DataField="au_fname"
        ItemStyle-Font-Italic="True"
        ItemStyle-Font-Name="Courier New"
    >

    </asp:BoundColumn>

    <asp:BoundColumn
        HeaderText="Sobrenome"
        DataField="au_lname"
        ItemStyle-Font-Italic="True"
        ItemStyle-Font-Name="Courier New"
    >

    </asp:BoundColumn>

    <asp:BoundColumn
        HeaderText="Telefone"
        DataField="phone"
        ItemStyle-BackColor="#c0ffff"
        ItemStyle-Font-Bold="True"
    >
```

```

</asp:BoundColumn>

<asp:EditCommandColumn
    EditText="Editar"
    CancelText="Cancelar"
    UpdateText="Atualizar"
/>

</Columns>

</asp:DataGrid>

</form>

</body>
</html>

```

Digite o código da Listagem 15.2 e salve o mesmo em um arquivo chamado chap15ex2.aspx, na pasta chap15, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap15/chap15ex2.aspx>

Ao carregar a página você obtém uma tabela com os dados sobre os autores. Observe a última coluna onde temos um link Editar, conforme indicado na Figura 15.4.

Dê um clique no link Editar, da primeira linha. Observe que os dados da linha passaram a ser exibidos em controles do tipo TextBox, nos quais podemos fazer as alterações necessárias. Altere o sobrenome do autor para Battisti, conforme indicado na Figura 15.5.

Observe que na última coluna, da primeira linha, ao invés do link Editar, passaram a ser exibidos dois links. Se você clicar no link Atualizar, as alterações serão enviadas para o banco de dados. Se você clicar no link Cancelar, as alterações serão descartadas e o DataGrid sairá do modo de edição e voltará a ser exibido somente o link Editar.

Dê um clique no botão Atualizar para confirmar as alterações. Observe que o sobrenome foi atualizado para Battisti e o DataGrid continua no modo de edição. Agora altere o sobrenome do autor da primeira linha, de volta para Bennet. Dê um clique no botão Cancelar. Observe que a última alteração é cancelada, isto é, o sobrenome continua sendo Battisti e o DataGrid saiu do modo de edição, conforme indicado na Figura 15.6.



Figura 15.4: Link para colocar o DataGrid no modo de edição.

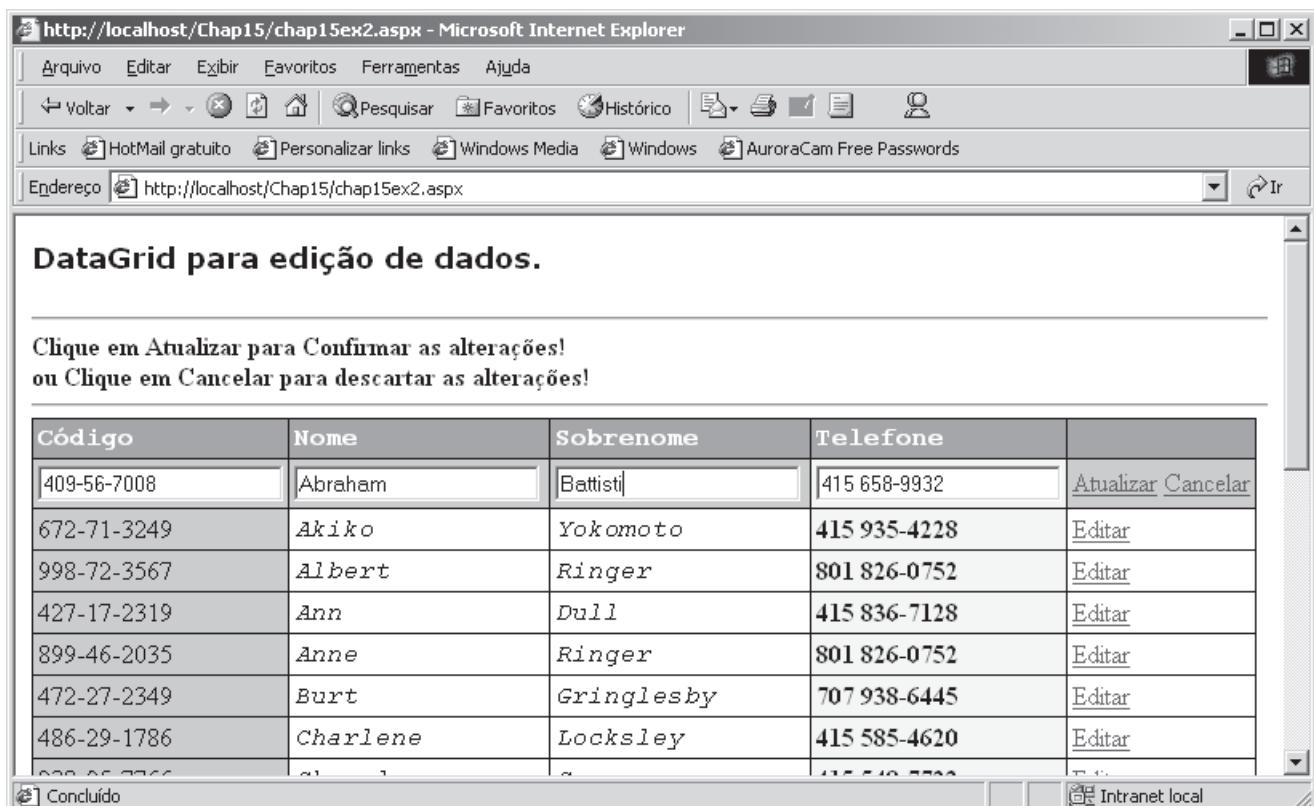


Figura 15.5: Editando dados com o controle DataGrid.

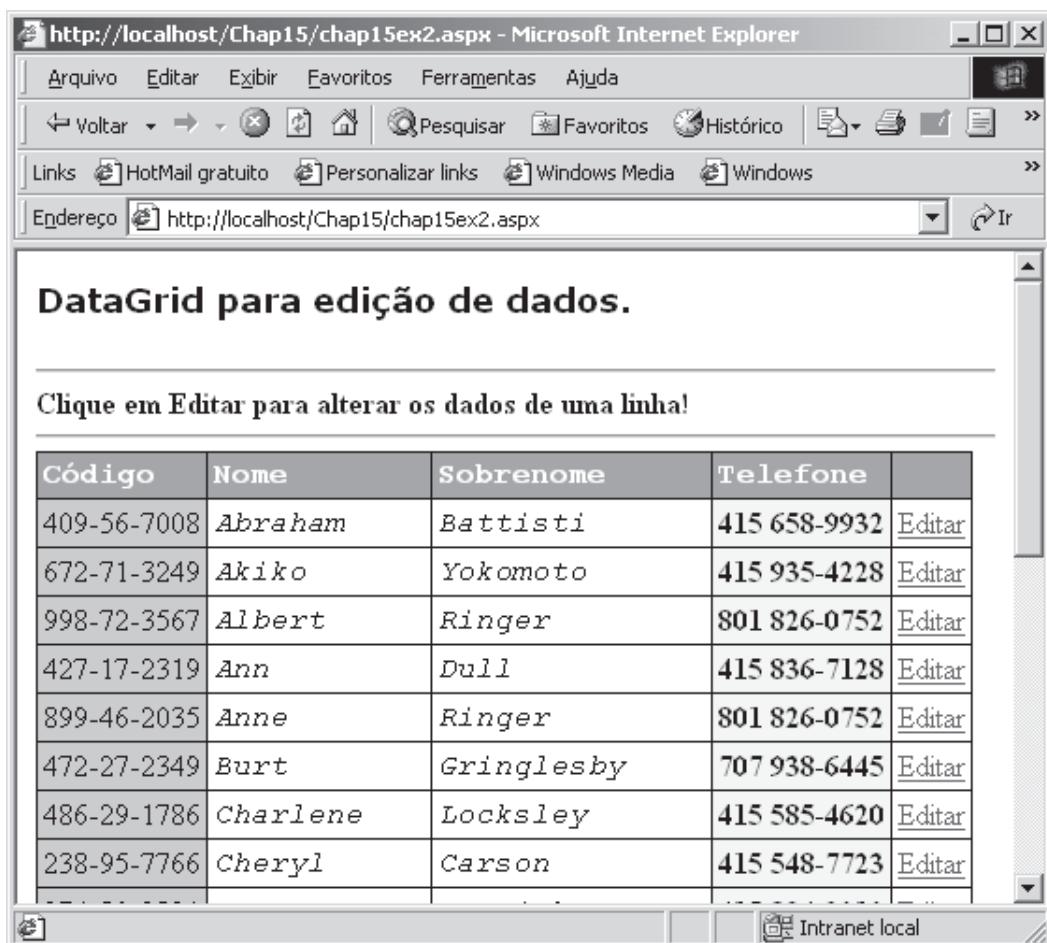


Figura 15.6: DataGrid no modo de visualização.

Comentários sobre o código do exemplo.

- ◆ Criamos um DataGrid, onde as colunas foram definidas manualmente, utilizando as tags `<columns>` `</columns>`. Para cada campo a ser exibido criamos uma coluna do tipo BoundColumn. O nome do campo associado com a coluna é definido pela propriedade DataField, como para a primeira coluna, que irá exibir dados do campo au_id:

```

<asp:BoundColumn
    HeaderText="Código"
    DataField="au_id"
    ItemStyle-BackColor="#c0c0c0"
>

```

Para exibir a última coluna, onde são exibidos os links Editar, Atualizar ou Cancelar, dependendo do modo em que se encontra o DataGrid, utilizamos uma coluna do tipo EditCommandColumn:

```

<asp:EditCommandColumn
    EditText="Editar"
    CancelText="Cancelar"
    UpdateText="Atualizar"
/>

```

Poderíamos definir uma série de propriedades desta coluna, porém definimos apenas o texto para o link de cada um dos comandos: Editar, Cancelar e Atualizar.

- ♦ Na definição do DataGrid definimos que as colunas não serão geradas automaticamente – AutoGenerateColumns="false". A parte mais importante da definição do DataGrid é a especificação dos procedimentos que serão executados quando o usuário clicar nos links Editar, Atualizar e Cancelar. Isto é feito através da definição das propriedades OnEditCommand (executado quando o usuário clica no link Editar), OnUpdateCommand (executado quando o usuário clica no link Atualizar) e OnCancelCommand (executado quando o usuário clica no link Cancelar). A seguir temos a definição do DataGrid:

```
<asp:DataGrid
    id="MinhaGrade"
    BorderColor="black"
    BorderWidth="1"
    CellPadding="3"
    AutoGenerateColumns="false"
    runat="server"
    EditItemStyle-BackColor="#c0c0c0"
    EditItemStyle-ForeColor="blue"
    OnEditCommand="FazEdicao"
    OnUpdateCommand="FazUpdate"
    OnCancelCommand="CancelaAlteracoes"
    >
```

- ♦ Quando a página é carregada pela primeira vez, isto é, quando não for um PostBack, o texto do início da página é definido e o procedimento BuscaDados é chamado. Este procedimento faz a conexão com o banco de dados e exibe as informações obtidas no controle DataGrid. Neste momento está habilitado apenas o link Editar.

```
if (!Page.IsPostBack)
{
    txtTitulo.Text="Clique em Editar para alterar os dados de uma linha!";
    BuscaDados();
}
```

- ♦ Quando o usuário clica no link Editar, dispara o evento OnEditCommand e é executado o procedimento FazEdicao. Este procedimento altera o texto dos labels txtTitulo e txtTitulo1 e coloca a linha onde o usuário clicou no link Editar, no modo de edição:

```
void FazEdicao(Object sender, DataGridCommandEvent e)
{
    txtTitulo.Text="Clique em Atualizar para Confirmar as alterações!";
    txtTitulo2.Text="ou Clique em Cancelar para descartar as alterações!";
    MinhaGrade.EditItemIndex = e.Item.ItemIndex;
    BuscaDados();
}
```

O índice da linha onde o usuário clicou é retornado pela propriedade ItemIndex do argumento e (do tipo DataGridCommandEventArgs), passado como parâmetro para o procedimento. Este índice é atribuído à propriedade EditRowIndex do DataGrid. O efeito prático é colocar a respectiva linha no modo de edição.

- ◆ Quando o usuário clica no link Cancelar, as alterações são abandonadas, o texto dos labels é atualizado e o DataGrid é retirado do modo de edição e o procedimento BuscaDados é chamado para reexibir os dados no DataGrid:

```
void CancelaAlteracoes(Object sender, DataGridCommandEventArgs e)
{
    txtTitulo.Text = "Clique em Editar para alterar os dados de uma linha!";
    txtTitulo2.Text = "";
    MinhaGrade.EditRowIndex = -1;
    BuscaDados();
}
```

Para retirar a linha do modo de edição atribuímos o valor –1 à propriedade EditRowIndex.

- ◆ Agora vamos analisar o procedimento FazUpdate, que é o procedimento mais complexo. Na parte inicial deste procedimento criamos os objetos para conexão com o banco de dados e preenchimento de um objeto DataView, com as informações originais da tabela authors.

O próximo passo é acessar os valores que foram digitados nos controles TextBox, da linha que está sendo editada. O ASP.NET cria uma coleção de controles do tipo TextBox, para a linha que está sendo editada. O primeiro controle possui o 0, o segundo o índice 1 e assim por diante. Para atribuir estes valores a variáveis do tipo string, utilizamos a seguinte sintaxe:

```
string txtCodAutor = ((TextBox)e.Item.Cells[0].Controls[0]).Text;
string txtNome      = ((TextBox)e.Item.Cells[1].Controls[0]).Text;
string txtSobreNome = ((TextBox)e.Item.Cells[2].Controls[0]).Text;
string txtTelefone  = ((TextBox)e.Item.Cells[3].Controls[0]).Text;
```

O argumento e é do tipo DataGridCommandEventArgs. Este argumento tem uma propriedade chamada Item, a qual é uma referência à linha atual do DataGrid, ou seja, a linha que estamos editando. A propriedade Item é baseada na classe DataGridItem, a qual possui uma coleção chamada Cells, coleção esta que representa todas as colunas (cada coluna é considerada uma célula) do DataGrid. Então considere a seguinte referência:

- ◆ e.Item.Cells[0] é uma referência à primeira coluna da linha que está sendo editada.
- ◆ e.Item.Cells[1] é uma referência à segunda coluna da linha que está sendo editada e assim por diante.

Em cada célula do DataGrid posso ter um ou mais controles. Estes controles fazem parte da coleção Controls da célula. O primeiro controle é acessado pelo índice 0 – Controls[0], o segundo pelo índice 1 – Controls[1] e assim por diante. No nosso caso temos então:

- ◆ e.Item.Cells[0].Controls[0] é uma referência ao primeiro controle da primeira coluna da linha que está sendo editada.

- ◆ e.Item.Cells[1].Controls[0] é uma referência ao segundo controle da segunda coluna da linha que está sendo editada e assim por diante.

Cada controle, no nosso exemplo, é um objeto do tipo TextBox. Para acessar o valor contido neste controle, utilizamos a sua propriedade Text. Finalmente chegamos à seguinte conclusão:

- ◆ e.Item.Cells[0].Controls[0].Text retorna o valor contido no primeiro (e único) controle da primeira coluna da linha que está sendo editada.
- ◆ e.Item.Cells[1].Controls[0].Text retorna o valor contido no primeiro (e único) controle da segunda coluna da linha que está sendo editada.
- ◆ e.Item.Cells[2].Controls[0].Text retorna o valor contido no primeiro (e único) controle da terceira coluna da linha que está sendo editada.
- ◆ e.Item.Cells[3].Controls[0].Text retorna o valor contido no primeiro (e único) controle da quarta coluna da linha que está sendo editada.

Além disso, tivemos que fazer uma conversão explícita, para o tipo TextBox. Isto é feito colocando-se o tipo TextBox entre parênteses na frente de cada um dos comandos.

De posse destes valores, utilizamos os passos aprendidos no Capítulo 12, para enviar as atualizações para o banco de dados. Em resumo:

1. Criamos um objeto DataTable.
2. Definimos uma chave primária para este objeto.
3. Localizamos a linha correspondente ao código do autor que está sendo editado.
4. Atualizamos a linha no objeto DataTable.
5. Enviamos as atualizações para o banco de dados.

Para maiores detalhes sobre estes passos, consulte o Capítulo 12, onde apresentamos explicações detalhadas sobre cada um destes passos.

Sem dúvida a parte mais complexa deste procedimento é o entendimento de como é feita a referência aos valores contidos em cada coluna da linha que está sendo editada. Podemos ver que existe uma hierarquia bem definida. O DataGridView é formado por uma coleção de linhas. Cada linha é formada por uma coleção de colunas. Cada coluna pode conter um ou mais controles, os quais fazem parte da coleção Controls da coluna. Cada controle é um objeto com uma série de propriedades e métodos. Utilizamos a propriedade Text do controle, para acessar o valor contido em cada controle.

Com o ASP 3.0, para implementarmos a mesma funcionalidade vista neste exemplo, temos que utilizar um componente COM vendido separadamente. Este componente simula as funcionalidades de edição que o DataGridView nos fornece apenas com a configuração de umas poucas propriedades.

Acessando Dados de uma Planilha do Excel

Vamos apresentar um exemplo, onde utilizaremos um controle DataGrid, para exibir dados de uma planilha do Excel: C:\Meus documentos\Clientes.xls. Esta planilha contém dados que foram importados da tabela Clientes, do banco de dados NorthWind.mdb. Não importamos todas as colunas. Para simplificar o exemplo, foram importadas apenas as seguintes colunas:

- ◆ CódigoDoCliente
- ◆ NomeDaEmpresa
- ◆ Endereço
- ◆ Cidade
- ◆ País
- ◆ Telefone

Na Listagem 15.3 temos o código para o exemplo proposto. Após a listagem apresentaremos mais alguns comentários sobre o exemplo.

Listagem 15.3 – Acessando dados de uma planilha do Microsoft Excel.

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data.OleDb" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System" %>

<script language="C#" runat="server">
protected void Page_Load(Object Src, EventArgs E)
{
    string strConn;
    strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" +
    "Data Source=C:\\Meus documentos\\Clientes.xls;" +
    "Extended Properties=Excel 8.0;";

    //You must use the $ after the object you reference in the spreadsheet

    OleDbDataAdapter myCommand = new OleDbDataAdapter("SELECT * FROM [Plan1$]", strConn);

    DataSet myDataSet = new DataSet();
}
```

```
myCommand.Fill(myDataSet, "Clientes");

MinhaGrade.DataSource = myDataSet.Tables[0].DefaultView;
MinhaGrade.DataBind();
}

</script>

<body>
<B>
<asp:Label
    id=Label1
    runat="server"
>
<HR>
    Dados da planilha C:\Clientes.xls
<HR>

</asp:Label>
</B>

<asp:DataGrid
    id=MinhaGrade
    HeaderStyle-BackColor="#c0c0c0"
    HeaderStyle-Font-Bold="True"
    runat="server"
/>

</body>
</html>
```

Digite o código da Listagem 15.3 e salve o mesmo em um arquivo chamado chap15ex3.aspx, na pasta chap15, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap12/chap15ex3.aspx>

Ao carregar a página você obtém o resultado indicado na Figura 15.7, onde são exibidos os dados da planilha Clientes.xls.

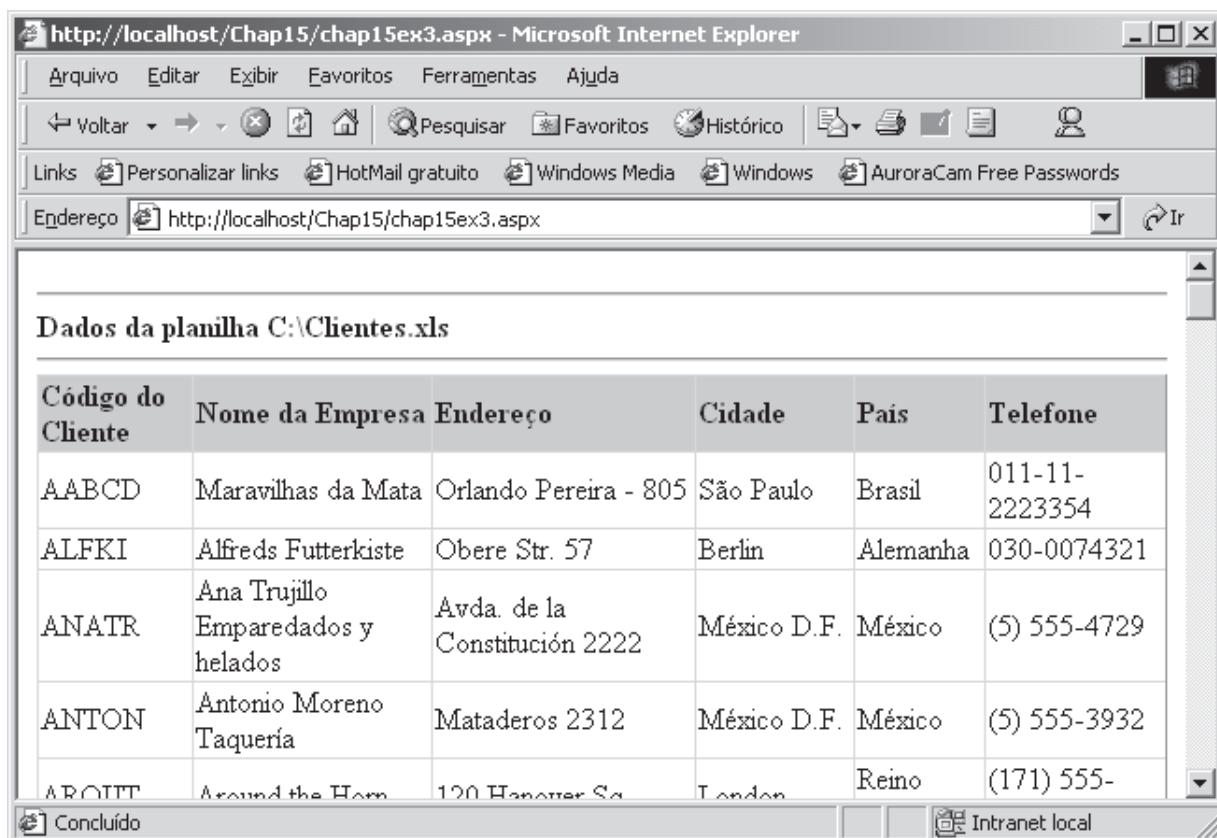


Figura 15.7: Acessando dados de uma planilha do Excel.

Comentários sobre o código do exemplo:

- Para fazer a conexão com uma planilha do Excel, utilizamos um objeto OleDbConnection; porém na definição da String de Conexão, utilizamos um atributo a mais:

```
string strConn;
strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" +
"Data Source=C:\\Meus documentos\\Clientes.xls;" +
"Extended Properties=Excel 8.0;";
```

O atributo “Extended Properties=Excel 8.0” é que informa que o arquivo está no formato do Excel.

Para acessar os dados da primeira planilha – Plan1, definimos um comando SQL:

```
OleDbDataAdapter myCommand = new OleDbDataAdapter("SELECT * FROM [Plan1$]", strConn);
```

Observe que, após o nome da planilha, devemos colocar um sinal de cifrão – [Plan1\$]

Poderíamos, também, selecionar colunas individuais da planilha. Para isso basta alterar o comando SQL. Por exemplo, se quiséssemos exibir apenas as colunas “Código do Cliente” e “Nome da Empresa”, utilizariam o seguinte comando:

```
OleDbDataAdapter myCommand = new OleDbDataAdapter("SELECT [Código do Cliente], [Nome da Empresa] FROM [Plan1$]", strConn);
```

Com este comando obteríamos o resultado indicado na Figura 15.8.

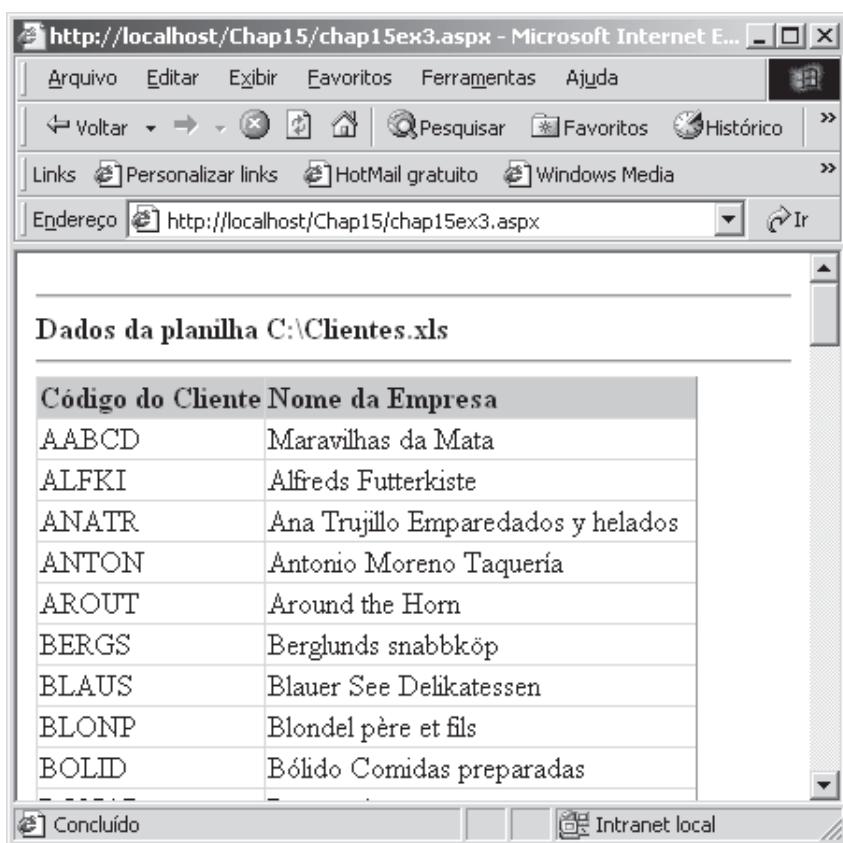


Figura 15.8: Acessando apenas algumas colunas.

- ◆ O restante do código não tem maiores novidades. O único detalhe diferente dos exemplos anteriores é que, ao invés de criar explicitamente um objeto do tipo DataView, utilizamos a propriedade DefaultView, da primeira (e única) tabela, da coleção de tabelas do objeto DataSet, para definir a propriedade DataSource do objeto DataGrid:

```
MinhaGrade.DataSource = myDataSet.Tables[0].DefaultView;
```

```
MinhaGrade.DataBind();
```

Este exemplo demonstra o poder e flexibilidade das classes do namespace System.Data.OleDb. Poderíamos, sem maiores problemas, exibir na mesma página ASP.NET dados de um banco de dados do SQL Server, de uma tabela de um arquivo do Microsoft Access, de uma planilha do Microsoft Excel e de uma planilha do Lotus 123. Poderíamos inclusive fazer cálculos com dados das diferentes fontes e salvar os resultados em um banco de dados do ORACLE. Esta flexibilidade e possibilidade de trabalhar com diferentes formatos de dados é que torna o Framework .NET uma opção muito interessante como plataforma para o desenvolvimento de aplicações empresariais, quer sejam aplicações Web, quer sejam aplicações tradicionais.

Um Conceito Importante: “Code Behind”

A idéia básica do conceito de Code Behind é fazer a separação entre o código responsável pela lógica de processamento da página ASP.NET e o código responsável pela apresentação da página. A apresentação é o que o usuário recebe de

volta no seu navegador. Para que os resultados estejam corretos, é necessário que a lógica da página tenha sido implementada corretamente. Esta lógica pode ser implementada na própria página, na seção de código ou em um arquivo separado, quando então estamos utilizando o conceito de Code Behind. Para ter acesso ao código gravado em um arquivo separado, fazemos uma referência ao arquivo, nas páginas onde iremos utilizá-lo.

O arquivo com o código responsável pela lógica pode ser criado com qualquer linguagem habilitada ao .NET. Por exemplo, pode ser um arquivo com a extensão .vb, criado com o VB.NET ou um arquivo com a extensão .cs, criado com o C#. Na página, onde o arquivo .vb ou .cs deve ser utilizado, devemos incluir uma referência ao mesmo, utilizando as cláusulas Inherits e Src, na diretiva @Page, conforme exemplo a seguir:

```
<% @Page Language="C#" Inherits="NomeDaClasse" Src="Caminho para o arquivo" %>
```

Considere o exemplo:

```
<% @Page Language="C#" Inherits="Conecta" Src="Conecta.cs" %>
```

Vamos a um exemplo prático, onde colocaremos o código responsável por fazer a conexão com o banco de dados NorthWind.mdb em um arquivo chamado conecta.cs. Neste arquivo criaremos uma classe chamada Conecta. Em seguida criaremos uma página ASP.NET que faz uso do código contido no arquivo conecta.cs.

Na Listagem 15.4 temos o código para o arquivo conecta.cs.

Listagem 15.4 – Arquivo com o código de conexão.

```
using System;
using System.Data;
using System.Data.OleDb;
using System.Web.UI;
using System.Web.UI.WebControls;

public class Conecta : Page

{
    // Variável pública que faz referência ao controle DataGrid.

    public DataGrid MinhaGrade1;

    public void Page_Load(Object Src, EventArgs E )
    {
        String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +

```

```

    "DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";

OleDbConnection MinhaConexão = new OleDbConnection(DefineConexão);

String auxSQL1;
String auxSQL2;
String auxSQL3;
String ComandoSQL;

auxSQL1= "SELECT CódigoDoCliente,NomeDaEmpresa,Endereço,Telefone,Páis";
auxSQL2= " FROM Clientes Where País='Brasil'";
auxSQL3= " Order By NomeDaEmpresa";

ComandoSQL= auxSQL1+auxSQL2+auxSQL3;

OleDbDataAdapter MeuComando = new OleDbDataAdapter(ComandoSQL, MinhaConexão);

DataSet ds = new DataSet();
MeuComando.Fill(ds,"Clientes");

DataView source = new DataView(ds.Tables[0]);

MinhaGrade1.DataSource = source ;
MinhaGrade1.DataBind();

}

}

```

O mecanismo de Code Behind é baseado na idéia de herança. Observe, pelo código da Listagem 15.4, que criamos uma nova classe chamada Conecta. A classe Conecta é baseada na classe Page. Isto é feito porque a classe Conecta será herdada pelas páginas ASP.NET onde a classe for utilizada. Como será herdada por páginas ASP.NET, a classe Conecta é definida como do tipo Page, que é a classe básica para todas as páginas ASP.NET. Além disso utilizamos uma série de diretivas using, para fazer referência aos namespaces utilizados pela classe:

```

using System;
using System.Data;
using System.Data.OleDb;
using System.Web.UI;
using System.Web.UI.WebControls;
public class Conecta: Page

```

Em seguida declaramos uma variável pública do tipo DataGrid. A declaração desta variável é necessária para que possamos fazer a ligação dos dados do objeto DataView, criado no procedimento Page_Load, com o controle DataGrid, da página que irá utilizar o código da classe Conecta.

```
public DataGrid MinhaGrade1;
```

Em seguida criamos um procedimento para o evento Page_Load. O código deste procedimento simplesmente faz a conexão com o banco de dados NorthWind.mdb e retorna alguns campos da tabela Clientes. Em seguida estes dados são exibidos em um controle DataGrid.

Agora vamos criar uma página ASP.NET que utiliza o código da classe Conecta, contido no arquivo conecta.cs. Considere o código da Listagem 15.5.

Listagem 15.5 – Utilizando a classe Conecta.

```
<%@ Page Language="C#" Inherits="Conecta" Src="conecta.cs" %>

<html>

<body>
    <HR>
    <B> Exemplo de utilização de "Code Behind".</B>
    <HR>
    <ASP:DataGrid
        id="MinhaGrade1"
        runat="server"
        Width="500"
        BackColor="#c0c0c0"
        BorderColor="black"
        ShowFooter="false"
        CellPadding=3
        CellSpacing="0"
        Font-Name="Verdana"
        Font-Bold="True"
        Font-Size="7pt"
        HeaderStyle-BackColor="#aaaadd"
        MaintainState="false"
    />
    <HR>
</body>
</html>
```

Digite o código da Listagem 15.5 e salve o mesmo em um arquivo chamado chap15ex4.aspx, na pasta chap15, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap15/chap15ex4.aspx>

Ao carregar a página você obtém o resultado indicado na Figura 15.9.

Para acessarmos a classe Conecta, utilizamos a diretiva @Page, conforme indicado a seguir:

```
<%@ Page Language="C#" Inherits="Conecta" Src="conecta.cs" %>
```

Como o arquivo conecta.cs está na mesma pasta da página Chap15ex4.aspx, informamos somente o nome do arquivo. Caso o arquivo .cs estivesse em uma subpasta, digamos Códigos, da pasta onde está a página .aspx, teríamos que informar o caminho completo, conforme indicado no exemplo a seguir:

```
<%@ Page Language="C#" Inherits="Conecta" Src="Códigos\conecta.cs" %>
```

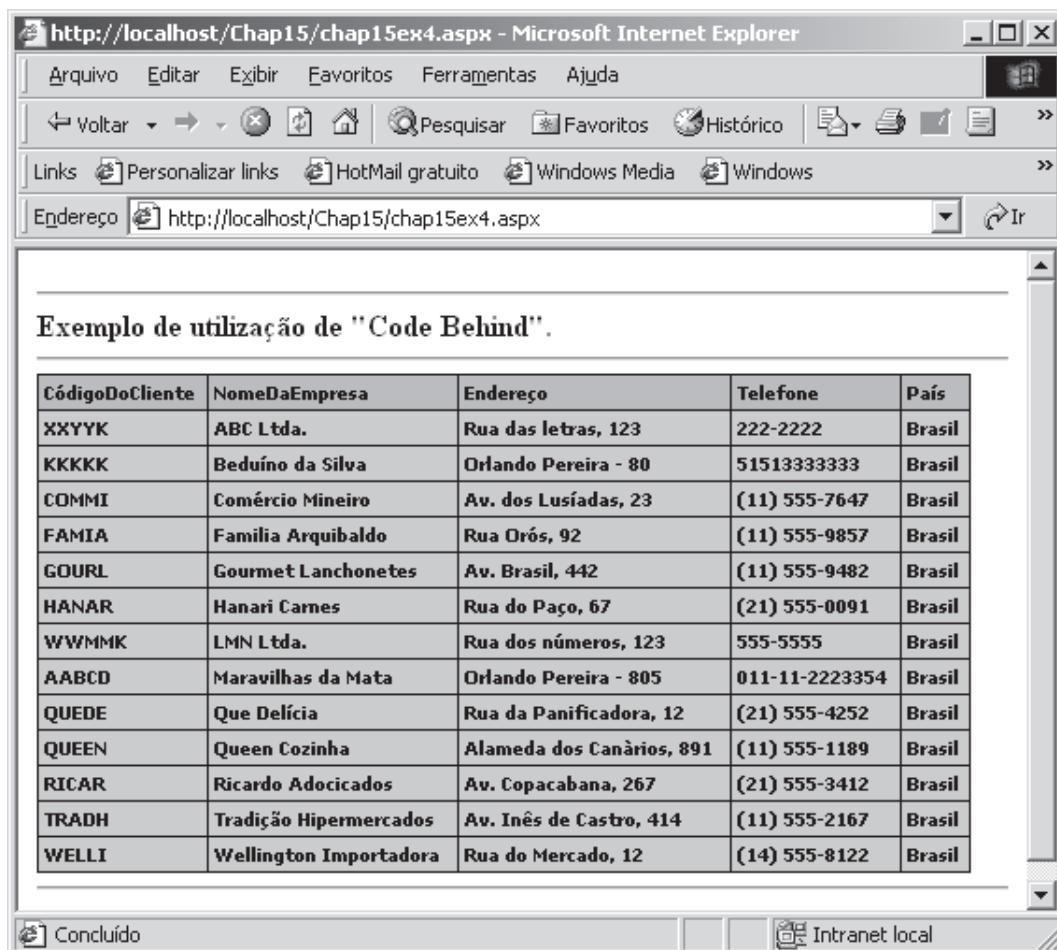


Figura 15.9: Utilizando Code Behind.

A utilização de Code Behind traz inúmeras vantagens, dentre as quais podemos destacar:

- ◆ Separação entre a lógica e a apresentação da página.

- ◆ Reaproveitamento de código, através do mecanismo de herança.
- ◆ Facilidade de manutenção. Podemos codificar uma funcionalidade, que será utilizada por diversas páginas, em um arquivo de código separado, como no nosso exemplo criamos a classe Conecta, definida no arquivo conecta.cs. Quando uma página ASP.NET precisa utilizar a lógica contida na classe Conecta, basta fazer referência ao arquivo conecta.cs, utilizando a diretiva @Page, conforme exemplificado anteriormente. Quando forem necessárias alterações na lógica da classe Conecta, basta que façamos as alterações no arquivo conecta.cs, e todas as páginas que utilizam a classe Conecta passarão a acessar a versão modificada. Se tivéssemos colocado o código da classe Conecta, na seção de código de cada página, no momento de fazer alterações teríamos que fazê-las em cada uma das páginas, o que tornaria o processo de manutenção e atualização bem mais complexo e oneroso.

O mecanismo de Code Behind demonstra, mais uma vez, o fato de os conceitos de Orientação a Objetos estarem presentes em todos os aspectos do Framework .NET. A possibilidade de codificar a lógica em uma classe separada e poder utilizar esta lógica em uma ou mais páginas ASP.NET nada mais é do que a implementação prática do conceito de reaproveitamento de código em aplicações Web.

O Objeto HttpRequest

Quando o usuário digita um endereço e pressiona Enter, ou clica no botão Enviar de um formulário, o navegador monta uma requisição e envia esta requisição, via HTTP, para o servidor. Na requisição enviada pelo navegador, existe uma série de informações. O objeto HttpRequest nos dá acesso às informações contidas na requisição enviada pelo navegador do cliente. Para acessarmos este objeto utilizamos simplesmente Request, conforme veremos nos exemplos no final deste tópico.

O objeto HttpRequest é baseado na classe HttpRequest, do namespace System.Web. A propriedade Request, do objeto Page, retorna um objeto do tipo HttpRequest, o qual contém informações sobre a requisição feita pelo cliente. Por isso podemos utilizar diretamente Request, em uma página ASP.NET, que esta propriedade estará fazendo referência a um objeto HttpRequest.

Com o ASP 3.0, para acessarmos as chamadas “Server Variables” (número IP do cliente, tipo de navegador, etc.), tínhamos que utilizar a coleção ServerVariables. Com o objeto HttpRequest, do ASP.NET, as variáveis de servidor estão disponíveis como propriedades do objeto HttpRequest.

Na Tabela 15.1, temos uma descrição das principais propriedades do objeto HttpRequest.

Tabela 15.1 Propriedades do objeto HttpRequest.

Propriedade	Descrição
AcceptTypes	Retorna um Array de strings, com todos os Mime Types aceitos pelo navegador do cliente.
ApplicationPath	Retorna uma String, indicando o caminho virtual para a pasta raiz da aplicação.
Browser	Retorna um objeto do tipo HttpBrowserCapabilities. As propriedades deste objeto fornecem informações sobre o navegador do cliente.

Propriedade	Descrição
Browser	Retorna um objeto do tipo <code>HttpBrowserCapabilities</code> . As propriedades deste objeto fornecem informações sobre o navegador do cliente.
ClientCertificate	Retorna um objeto do tipo <code>HttpClientCertificate</code> . As propriedades deste objeto fornecem informações sobre o certificado de segurança do Cliente.
ContentLength	Retorna o tamanho, em bytes, da requisição enviada pelo cliente.
ContentType	Retorna uma string indicando o tipo do conteúdo requisitado pelo cliente.
Cookies	Retorna uma coleção de Cookies, enviados na requisição do cliente.
HttpMethod	Retorna o método HTTP de transferência utilizado pelo cliente. Pode ser: GET, POST ou HEAD.
IsSecureConnection	Retorna True se a conexão está utilizando HTTPS (sockets seguros) e False, caso contrário.
Path	Retorna o caminho virtual da requisição atual.
PhysicalPath	Retorna o caminho físico correspondente a URL contida na requisição.
QueryString	Nos dá acesso às variáveis passadas na própria URL, quando é utilizado o método GET para enviar as informações contidas no campo de um formulário.
Url	Retorna informações sobre a URL da requisição atual.
UserAgent	Retorna uma string que identifica o navegador do Cliente. Por exemplo: "MSIE" identifica o Internet Explorer da Microsoft.
UserHostName	Retorna o número IP da estação do cliente.
UserHostAddress	Retorna o nome DNS da estação do cliente.

A seguir temos um exemplo, onde utilizamos a propriedade Browser para exibir as capacidades do navegador do cliente.

Listagem 15.6 – A propriedade Browser do objeto `HttpRequest`.

```
<html>
<script language="C#" runat="server">
protected void Page_Load(Object Src, EventArgs E )
{
    // Utilizo a propriedade Browser para exibir as capacidades
    // do navegador do cliente.

    HttpBrowserCapabilities bc = Request.Browser;
    Response.Write("<HR>");
}
```

```

Response.Write("<B>Propriedades do seu navegador.</B>");

Response.Write("<HR>");

Response.Write("Tipo = " + bc.Type + "<br>");

Response.Write("Nome = " + bc.Browser + "<br>");

Response.Write("Versão = " + bc.Version + "<br>");

Response.Write("Maior Versão = " + bc.MajorVersion + "<br>");

Response.Write("Menor Versão = " + bc.MinorVersion + "<br>");

Response.Write("Plataforma = " + bc.Platform + "<br>");

Response.Write("É versão Beta ? " + bc.Beta + "<br>");

Response.Write("É Crawler = " + bc.Crawler + "<br>");

Response.Write("É AOL = " + bc.AOL + "<br>");

Response.Write("É Win16 = " + bc.Win16 + "<br>");

Response.Write("É Win32 = " + bc.Win32 + "<br>");

Response.Write("Suporta Frames = " + bc.Frames + "<br>");

Response.Write("Suporta Tabelas = " + bc.Tables + "<br>");

Response.Write("Suporta Cookies = " + bc.Cookies + "<br>");

Response.Write("Suporta VB Script = " + bc.VBScript + "<br>");

Response.Write("Suporta JavaScript = " + bc.JavaScript + "<br>");

Response.Write("Suporta Applets Java = " + bc.JavaApplets + "<br>");

Response.Write("Suporta ActiveX Controls = " + bc.ActiveXControls + "<br>");

Response.Write("CDF = " + bc.CDF + "<br>");

Response.Write("<HR>");

}

</script>

<body>

</body>

</html>

```

Digite o código da Listagem 15.6 e salve o mesmo em um arquivo chamado chap15ex6.aspx, na pasta chap15, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap15/chap15ex6.aspx>

Ao carregar a página você obtém um resultado semelhante ao indicado na Figura 15.10.

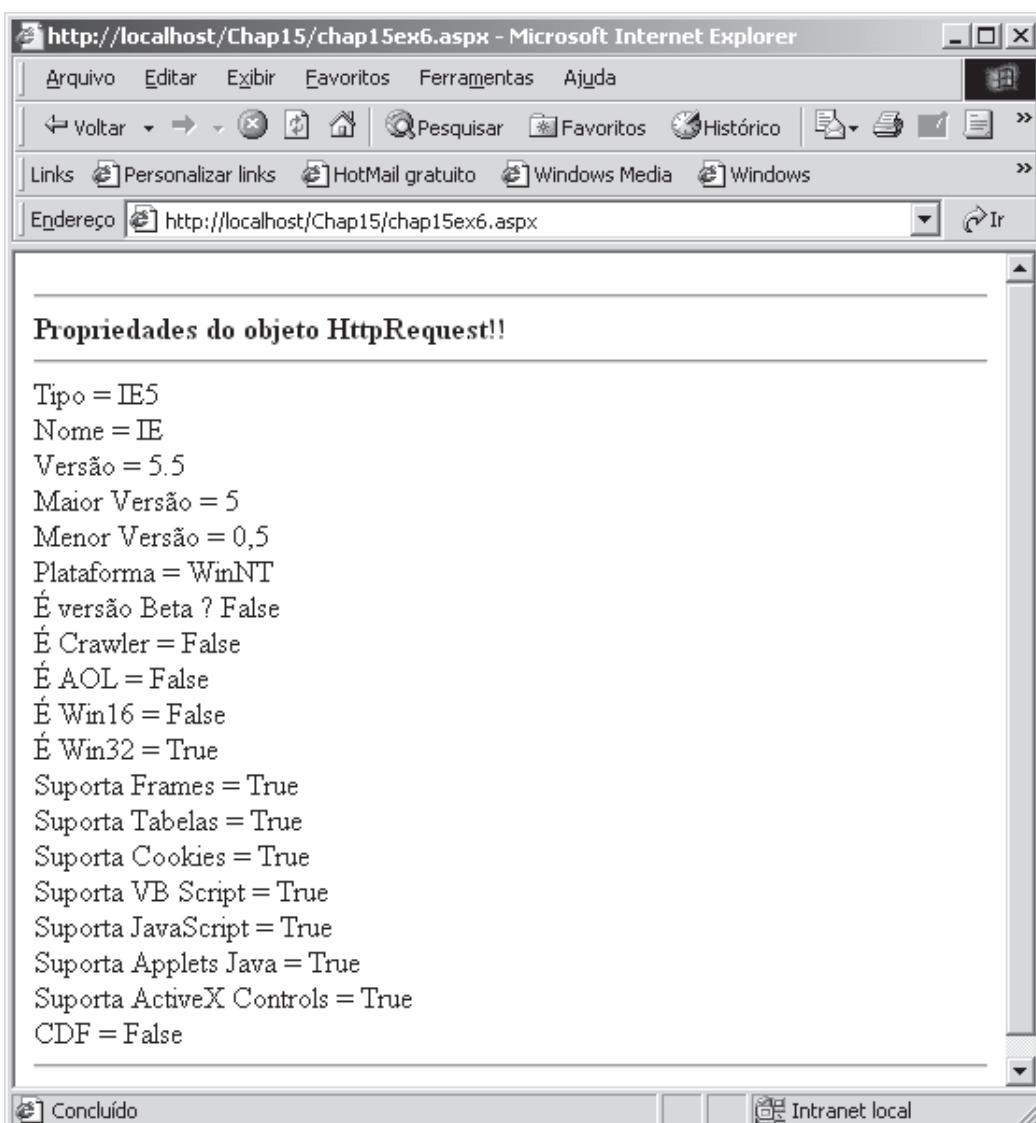


Figura 15.10: A propriedade Browser do objeto HttpRequest.

Utilizamos o evento PageLoad da página, onde criamos um objeto do tipo HttpBrowserCapabilities. Para criar este objeto utilizamos a propriedade Browser do objeto Request:

```
HttpBrowserCapabilities bc = Request.Browser;
```

Depois utilizamos uma série de comando Response.Write, para retornar o valor das diversas propriedades do navegador do cliente.

Vamos a mais um exemplo, onde utilizaremos outras propriedades do objeto HttpRequest.

Listagem 15.7 – Outras propriedades do objeto HttpRequest.

```
<html>
<script language="C#" runat="server">
    protected void Page_Load(Object Src, EventArgs E )
    {

```

```
// Utilizo a propriedade Browser para exibir as capacidades
// do navegador do cliente.

HttpClientCertificate cs = Request.ClientCertificate;

Response.Write("ClientCertificate Settings:<br>");
Response.Write("Certificate = " + cs.Certificate + "<br>");
Response.Write("Cookie = " + cs.Cookie + "<br>");
Response.Write("Flags = " + cs.Flags + "<br>");
Response.Write("IsPresent = " + cs.IsPresent + "<br>");
Response.Write("Issuer = " + cs.Issuer + "<br>");
Response.Write("IsValid = " + cs.IsValid + "<br>");
Response.Write("KeySize = " + cs.KeySize + "<br>");
Response.Write("SecretKeySize = " + cs.SecretKeySize + "<br>");
Response.Write("SerialNumber = " + cs.SerialNumber + "<br>");
Response.Write("ServerIssuer = " + cs.ServerIssuer + "<br>");
Response.Write("ServerSubject = " + cs.ServerSubject + "<br>");
Response.Write("Subject = " + cs.Subject + "<br>");
Response.Write("ValidFrom = " + cs.ValidFrom + "<br>");
Response.Write("ValidUntil = " + cs.ValidUntil + "<br>");
Response.Write("What's this = " + cs.ToString() + "<br>");

// Informações sobre o número IP do cliente.
String ClientIP;
ClientIP = Request.UserHostAddress;
Response.Write("<HR>");
Response.Write("Número IP =" + ClientIP + "<br>");

String Caminho;
Caminho = Request.Path;
Response.Write("<HR>");
Response.Write("Propriedade Path = " + Caminho + "<br>");

String FileCaminho;
FileCaminho = Request.FilePath;
Response.Write("<HR>");
Response.Write("Propriedade FilePath = " + FileCaminho + "<br>");
String FisicalFileCaminho;
FisicalFileCaminho = Request.PhysicalApplicationPath;
Response.Write("<HR>");
Response.Write("Propriedade PhysicalFilePath = " + FisicalFileCaminho + "<br>");

String FisicalCaminho;
```

```

FisicalCaminho = Request.PhysicalPath;
Response.Write("<HR>");
Response.Write("Propriedade PhysicalPath = " + FisicalCaminho + "<br>");
}

</script>

<body>

</body>
</html>

```

Digite o código da Listagem 15.7 e salve o mesmo em um arquivo chamado chap15ex7.aspx, na pasta chap15, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap15/chap15ex7.aspx>

Ao carregar a página você obtém um resultado semelhante ao indicado na Figura 15.11.

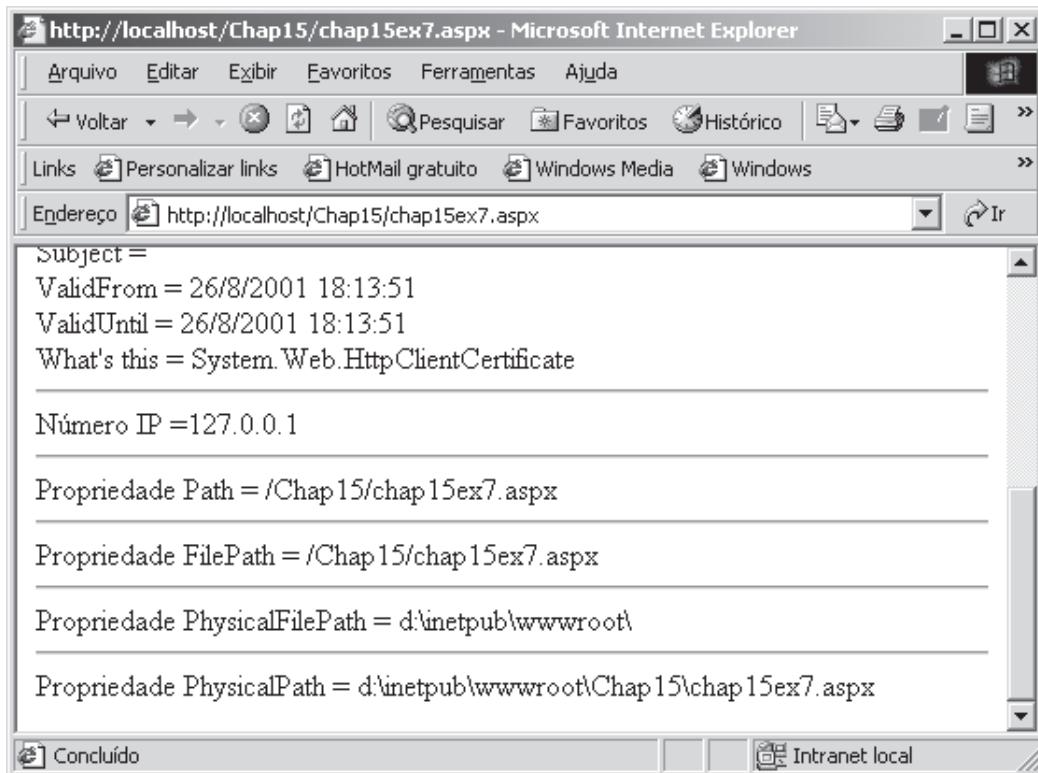


Figura 15.11: Outras propriedades do objeto HttpRequest.

Utilizamos o evento PageLoad da página, onde exibimos informações sobre o certificado de segurança do cliente e informações sobre as diversas variáveis Path do objeto HttpRequest.

Também exibimos informações sobre o endereço IP do cliente. Como estou utilizando o endereço http://localhost, observe que está sendo retornado o IP 127.0.0.1, que é o IP associado com o nome local – localhost.

O Objeto HttpResponse

O objeto `HttpResponse` é utilizado para enviar informações para o navegador do cliente, em resposta a uma requisição. A propriedade `Response`, do objeto `Page`, retorna um objeto do tipo `HttpResponse`. Este objeto é derivado da classe `HttpResponse`, do namespace `System.Web`. Por isso este objeto, suas propriedades e métodos podem ser utilizados diretamente em uma página ASP.NET.

Quando uma página é requisitada, é criada uma instância da classe `page` na memória do servidor, conforme descrito no Capítulo 6. Em qualquer local desta página podemos utilizar a propriedade `Response`; na prática é como se tivéssemos criado uma instância da classe `HttpResponse`. Por isso que, nos exemplos anteriores, utilizamos diretamente `Response.Write`. Quando fizemos isso, estamos utilizando o método `Write`, da classe `HttpResponse`, classe essa acessada através da propriedade `Response` da página, ou diríamos melhor, da classe `Page`.

Na Tabela 15.2, temos uma descrição das principais propriedades do objeto `HttpResponse`.

Tabela 15.2 Propriedades do objeto `HttpResponse`.

Propriedade	Descrição
<code>Buffer</code>	Pode ser definido em <code>True</code> ou <code>False</code> . Se for <code>True</code> , toda a resposta será processada, antes de ser enviada para o navegador do cliente. Caso seja <code>False</code> , à medida que a resposta for sendo processada, será enviada para o navegador do cliente.
<code>BufferOutput</code>	Pode ser definido em <code>True</code> ou <code>False</code> . Se for <code>True</code> , toda a página será processada, antes de ser enviada para o navegador do cliente. Caso seja <code>False</code> , à medida que a página for sendo processada, será enviada para o navegador do cliente.
<code>Cache</code>	Retorna um objeto do tipo <code>HttpCachePolicy</code> com informações sobre as configurações de cache para a resposta atual.
<code>Cookies</code>	Retorna a coleção de <code>Cookies</code> contida na resposta para o navegador do cliente.
<code>Write</code>	O método mais conhecido do objeto <code>HttpResponse</code> é, sem dúvida, o método <code>Write</code> , que é utilizado para enviar texto para o navegador do Cliente. Com o ASP 3.0, utilizávamos, intensamente, o método <code>Write</code> (<code>Response.Write</code>) para enviar as tags HTML que formavam a página de resposta, página esta que era enviada de volta para o cliente. Com os Web Server Controls (vistos nos Capítulos 7, 8 e 9) e suas funcionalidades avançadas, o uso de <code>Response.Write</code> ficou bastante reduzido no ASP.NET, conforme podemos constatar nos diversos exemplos deste livro.

Diretivas de Página

As diretivas de página são utilizadas para definir uma série de “comportamentos” que influenciam a maneira como uma página é processada e exibida no navegador do cliente. As diretivas podem ser incluídas em qualquer local da página, mas é de praxe colocarmos as diretivas no início da página. Uma diretiva pode conter um ou mais atributos, que definem configurações relacionadas com a diretiva.

A sintaxe geral para as diretivas é a seguinte:

```
<% @Diretiva Atributo1="Valor" Atributo2="Valor" ... AtributoN="Valor" %>
```

A Diretiva @Page

Esta diretiva é utilizada para definir atributos específicos para a página ASP.NET. Os atributos são utilizados pelo processador ASP.NET para definir como a página será processada, se será ou não mantida em cache, como a resposta será enviada para o cliente e assim por diante. A diretiva @Page somente pode ser utilizada em arquivos do tipo .aspx.

A seguir descrevemos os principais atributos para esta diretiva.

- ◆ **AspCompact:** Pode conter o valor true ou false. Se for definida em true permite que a página .aspx acesse componentes COM antigos, criados em VB, que utilizam o modelo de Single-thread Apartment – STA, para alocação de memória e execução.

Exemplo: <% @Page AspCompact="true" %>

- ◆ **Buffer:** Define se o buffer está ou não habilitado. Pode conter os valores true ou false.

Exemplo: <% @Page Buffer="false" %>

- ◆ **ClientTarget:** Pode ser utilizado para informar ao processador ASP.NET qual o navegador do cliente, uma vez que a maneira como os controles são processados é otimizada para cada tipo de navegador. Somente são válidos os valores aceitáveis pela propriedade User Agent, do objeto HttpRequest.

- ◆ **Debug:** Pode ser do tipo true ou false. Já utilizamos em alguns exemplos deste livro. Se for definida em true, quando a página é compilada, o processo de debug estará habilitado. Com o processo de debug habilitado, mais informações são geradas caso aconteça algum erro. Esta opção é muito útil quando a página está em desenvolvimento, devendo ser desabilitada quando a página estiver disponível para uso. O padrão é false.

Exemplo: <% @Page Debug="true" %>

- ◆ **EnableViewState:** Pode conter os valores true ou false. Se for true (o padrão), o estado será mantido para a página e para os controles da página; se for false, o estado não será mantido.

- ◆ **ErrorPage:** Define a URL de uma página de erro, para a qual o processamento será redirecionado caso aconteça algum erro para o qual não foi feito o devido tratamento de exceções. Para mais informações sobre o tratamento de exceções, consulte o Capítulo 5.

- ◆ **Inherits:** Utilizada para configuração do mecanismo de Code Behind. Para maiores detalhes consulte o tópico – Um conceito importante: “Code Behind”, neste capítulo.

- ◆ **Language:** Utilizada para definir a linguagem que será utilizada na página. Na prática define qual o compilador que deve ser utilizado para compilar a página. Se não utilizarmos esta diretiva, será utilizada a linguagem VB.NET.
`<%@ Page Language="C#" %>`
- ◆ **Src:** Informa o caminho para o arquivo contendo o código da classe especificada no atributo Inherits. Para maiores detalhes consulte o tópico – Um conceito importante: “Code Behind”, neste capítulo.

A Diretiva @Import

Esta diretiva é utilizada para fazer referência a um namespace. Se formos utilizar classe de um namespace, precisamos utilizar a diretiva import para fazer referência ao respectivo namespace. Por exemplo, quando utilizamos classes do namespace System.Data.OleDb, para fazer conexão com um banco de dados do Microsoft Access, precisamos utilizar a seguinte diretiva:

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>
```

Em cada diretiva @Import somente podemos fazer referência a um namespace. Para fazermos referência a diversos namespaces devemos utilizar várias diretivas @Import.

Existe um conjunto de namespaces aos quais já é feita referência automática, ou seja, não precisamos fazer referência explícita, utilizando a diretiva @Import. Na lista a seguir temos os namespaces aos quais é feita referência, automaticamente:

- ◆ System
- ◆ System.Collections
- ◆ System.Collections.Specialized
- ◆ System.Configuration
- ◆ System.IO
- ◆ System.Text
- ◆ System.Text.RegularExpressions
- ◆ System.Web
- ◆ System.Web.Caching
- ◆ System.Web.Security
- ◆ System.Web.SessionState
- ◆ System.Web.UI
- ◆ System.Web.UI.HtmlControls
- ◆ System.Web.UI.WebControls

Outras Diretivas

Existem outras diretivas, utilizadas com menor freqüência: @Import, @Implements, @Register, @Assembly, @OutputCache e @Reference. Para maiores informações sobre estas diretivas consulte a documentação do Frame-

work .NET no seguinte tópico: .NET Framework Reference -> ASP.NET Syntax -> Web Forms Syntax -> Directive Syntax.

Configurações de Segurança Através de Programação

Em determinadas situações pode ser necessário detectar, no próprio código da página ASP.NET, qual o nome do usuário autenticado e a quais grupos ele pertence. Uma vez sabendo o nome do usuário podemos liberar ou negar o acesso a determinados recursos. Outra aplicação prática seria a criação de conteúdos personalizados para diferentes grupos de usuários. Neste tópico apresentaremos um exemplo de conteúdo personalizado, onde diferentes colunas da tabela Clientes serão exibidas, dependendo do grupo ao qual pertence o usuário que estiver acessando a página.

Os objetos e métodos que veremos neste tópico são utilizados com a autenticação do tipo Windows, a qual normalmente é utilizada em ambientes de Intranet ou Extranet. Neste caso cada usuário é cadastrado e faz o logon com a sua conta de usuário. O usuário pode pertencer a um ou mais grupos.

Para o exemplo que apresentaremos neste capítulo utilizaremos os seguintes usuários:

- ◆ GROZA\suers1
- ◆ GROZA\suers2
- ◆ GROZA\suers2

NOTA: Para maiores informações sobre a criação de usuários, criação de grupos e adição de usuários a grupos, consulte o Anexo I.

Observe que estamos trabalhando em um domínio chamado GROZA. Substitua GROZA pelo nome do domínio que você está utilizando.

Também iremos considerar dois grupos:

Gerentes GROZA\suser1
 GROZA\suser2

Funcionarios GROZA\user2
 GROZA\user3

Podemos notar que o usuário suser2 pertence aos dois grupos.

Acessando Informações Sobre o Usuário Autenticado

Quando utilizamos autenticação do tipo Windows (descrita no Capítulo 14), temos acesso a um objeto chamado User, o qual é acessado através da propriedade User, do objeto HttpContext. A classe HttpContext contém todas as informações sobre a requisição feita pelo navegador do cliente. Uma das informações contidas na requisição é o nome do usuário, senha e domínio, para o caso da autenticação Windows. Informações estas que podem ser acessadas no código de uma página ASP.NET. A classe HttpContext pertence ao namespace System.Web. A propriedade User, da classe HttpContext, retorna diversas informações de segurança sobre a requisição enviada pelo navegador do cliente.

A propriedade User retorna um objeto do tipo IPrincipal, derivado da classe IPrincipal, do namespace System.Security.Principal. A principal propriedade da classe IPrincipal é a propriedade IDentity, a qual é uma instância da classe IDentity, do namespace System.Security.Principal. Através das propriedades da classe IDentity é que temos acesso às informações do usuário que fez a requisição. Esta classe fornece uma série de propriedades, conforme indicado na Tabela 15.3.

Na Tabela 15.3, temos uma descrição das principais propriedades da classe IDentity.

Tabela 15.3 Propriedades da classe IDentity.

Propriedade	Descrição
AuthenticationType	Retorna o tipo de autenticação.
IsAuthenticated	Retorna true se o usuário foi autenticado com sucesso e false, caso contrário.
Name	Retorna o nome do usuário autenticado.

O tipo de objeto IDentity retornado é diferente, para os diferentes tipos de identificação. Para a autenticação Windows, o objeto retornado é do tipo WindowsIdentity, derivado da classe de mesmo nome, pertencente ao namespace System.Security.Principal.

Vamos inicialmente a um exemplo simples, onde retornaremos algumas informações sobre o usuário autenticado.

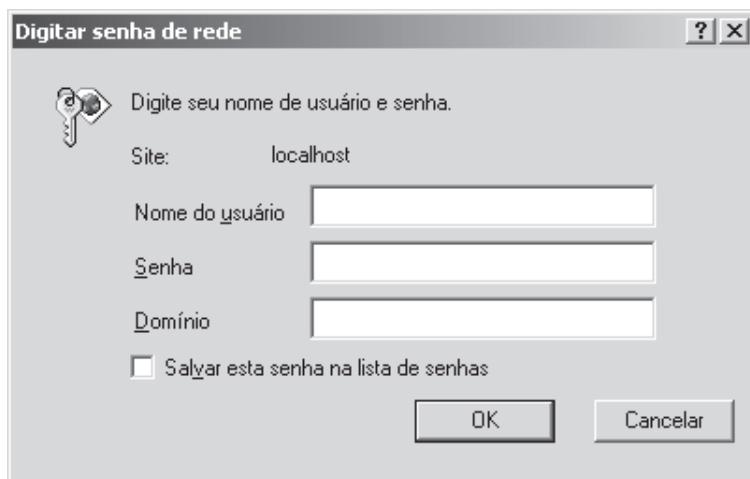
Antes de apresentarmos o exemplo, vamos configurar o arquivo Web.Config, para a pasta Chap15, de tal forma que esteja habilitada a autenticação do tipo Windows. Crie o arquivo Web.Config indicado na Listagem 15.8 e salve-o na pasta Chap15:

Listagem 15.8 – Configurando o tipo de autenticação – Web.Config.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <authentication mode="Windows" />

    <authorization>
      <allow roles="GROZA\Gerentes,GROZA\Funcionários"
            users="GROZA\suser1,GROZA\suser2,GROZA\suser3"/>
      <deny users="*" />
    </authorization>
  </system.web>
</configuration>
```

Ao tentar acessar uma página da aplicação Web Chap15, o usuário receberá uma tela de logon conforme indicado na Figura 15.12.



IMPORTANTE: Não se esqueça de configurar a pasta Chap15 como uma aplicação Web. Para maiores informações sobre como tornar uma pasta virtual em uma aplicação Web, consulte o Capítulo 13.

Figura 15.12: Tela de logon – autenticação do tipo Windows habilitada no arquivo Web.Config.

Se o usuário fornecer um nome de logon que não tem permissão de acesso, a tela de logon será exibida novamente. Isto é feito três vezes, e após a terceira tentativa, será exibida a mensagem de erro indicada na Figura 15.13.

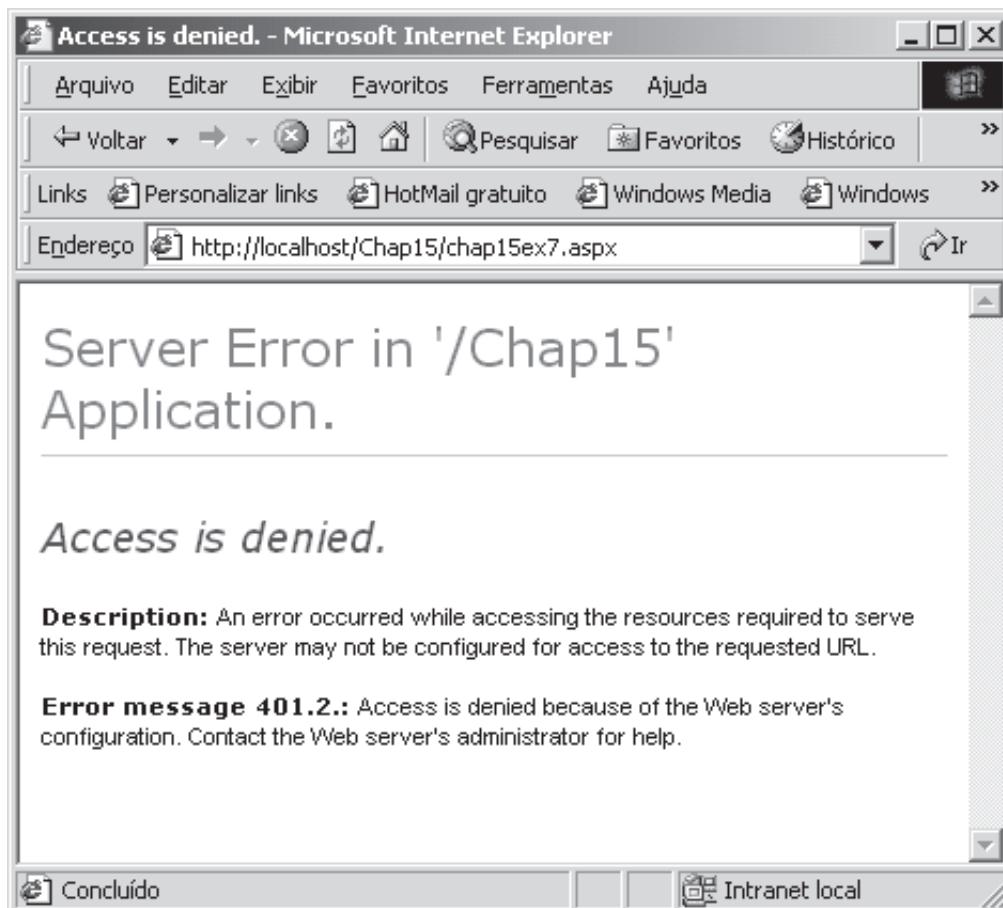


Figura 15.13: Três tentativas de logon sem sucesso.

Se você clicar no botão Cancelar, na tela de logon, também será emitida a mensagem da Figura 15.1.

Conforme configurado no nosso arquivo Web.Config, somente os usuários suser1, suser2 e suser3, do domínio GROZA e os participantes dos grupos Gerentes ou os participantes do grupo Funcionários têm permissão para acessar as páginas da aplicação Web – Chap15. Agora vamos tentar fazer o logon como um dos usuários que têm permissão de acesso. Vou utilizar o usuário suser1, o qual cadastrei com a senha: abc12345. Vou tentar acessar a página Chap15ex7.aspx, criada anteriormente. Ao acessar esta página será exibida a tela de logon. Digite as informações de logon indicadas na Figura 15.14 (senha=abc12345).

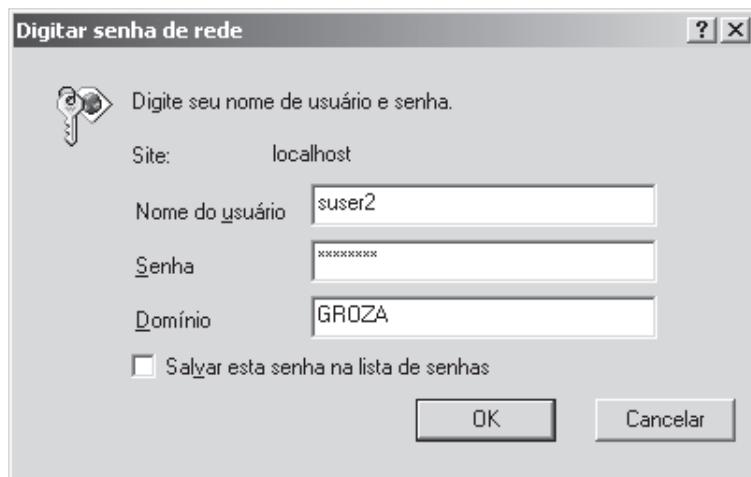


Figura 15.14: Usuário com permissões de acesso.

Neste caso o acesso à página será liberado sem maiores problemas.

Exemplo: Agora vamos a um exemplo mais completo. Criaremos um exemplo chamado Chap15ex8.aspx. Neste exemplo utilizaremos o objeto Identity e o método IsInRole, para determinar se o usuário pertence ao grupo Gerentes ou ao grupo Funcionários. Dependendo do grupo ao qual pertencer o usuário serão exibidas diferentes versões da página.

Listagem 15.9 – Configurações de segurança com o código ASP.NET.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>

<html>

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

protected void Page_Load(Object Src, EventArgs E )
{
    / Variáveis do tipo string que irão conter informações
    / sobre o usuário autenticado.
}
```

```
tring NomeDoUsuario      = User.Identity.Name;
tring TipoDeAutenticacao = User.Identity.AuthenticationType;

/ Verifico se o usuário pertence ao grupo Gerentes.

f (User.IsInRole("GROZA\\Gerentes"))
{
    MostraNome.Text     = "Sr. Gerente: " + NomeDoUsuario + " , seja bem
vindo!";
    MostraTipo.Text     = "Você está autenticado usando: " + TipoDeAutenticacao;
    Mensagem.Text       = "INFORMAÇÕES SOBRE CLIENTES!";

    String DefineConexão= "PROVIDER=MICROSOFT.JET.OLEDB.4.0;" +
                           "DATA SOURCE=c:\\meus documentos\\NorthWind.mdb";

    OleDbConnection MinhaConexão = new OleDbConnection(DefineConexão);

    OleDbDataAdapter MeuComando = new OleDbDataAdapter("SELECT
CódigoDoCliente," + "NomeDaEmpresa,País,Cidade FROM Clientes", MinhaConexão);

    DataSet ds = new DataSet();
    MeuComando.Fill(ds);
    DataView source = new DataView(ds.Tables[0]);

    MinhaGrade.DataSource = source ;
    MinhaGrade.DataBind();
}

else
{
    MostraNome.Text     = "Prezado Funcionário: " + NomeDoUsuario +
" , seja bem vindo!";
    MostraTipo.Text     = "Você está autenticado usando: " + TipoDeAutenticacao;
    Mensagem.Text       = "VOCÊ NÃO TEM PERMISSÃO PARA ACESSAR AS
INFORMAÇÕES DE CLEINTES!";
}
}

</script>

<body>
```

```
<asp:Label  
    id="MostraNome"  
    Text=""  
    Font-Bold="True"  
    BackColor="#c0c0c0"  
    runat="server"  
/>  
  
<BR>  
  
<asp:Label  
    id="MostraTipo"  
    Text=""  
    Font-Bold="True"  
    BackColor="#000000"  
    ForeColor="#ffffff"  
    runat="server"  
/>  
  
<BR>  
  
<asp:Label  
    id="Mensagem"  
    Text=""  
    Font-Bold="True"  
    BackColor="#c0c0c0"  
    runat="server"  
/>  
  
<HR>  
  
<ASP:DataGrid id="MinhaGrade" runat="server"  
    Width="500"  
    BackColor="#ccccff"  
    BorderColor="black"  
    ShowFooter="false"  
    CellPadding=3  
    CellSpacing="0"  
    Font-Name="Verdana"
```

```

Font-Size="8pt"
HeaderStyle-BackColor="#aaaaadd"
MaintainState="false"
/>

```

</body>

</html>

Digite o código da Listagem 15.8 e salve o mesmo em um arquivo chamado chap15ex8.aspx, na pasta chap15, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço: <http://localhost/chap15/chap15ex8.aspx>

Quando for solicitada a tela de logon faça o logon com as seguintes informações:

Nome do usuário: suser2

Senha: abc12345

Domínio: GROZA

Como o usuário suser2 faz parte do grupo Gerentes, você obterá os resultados indicados na Figura 15.15.

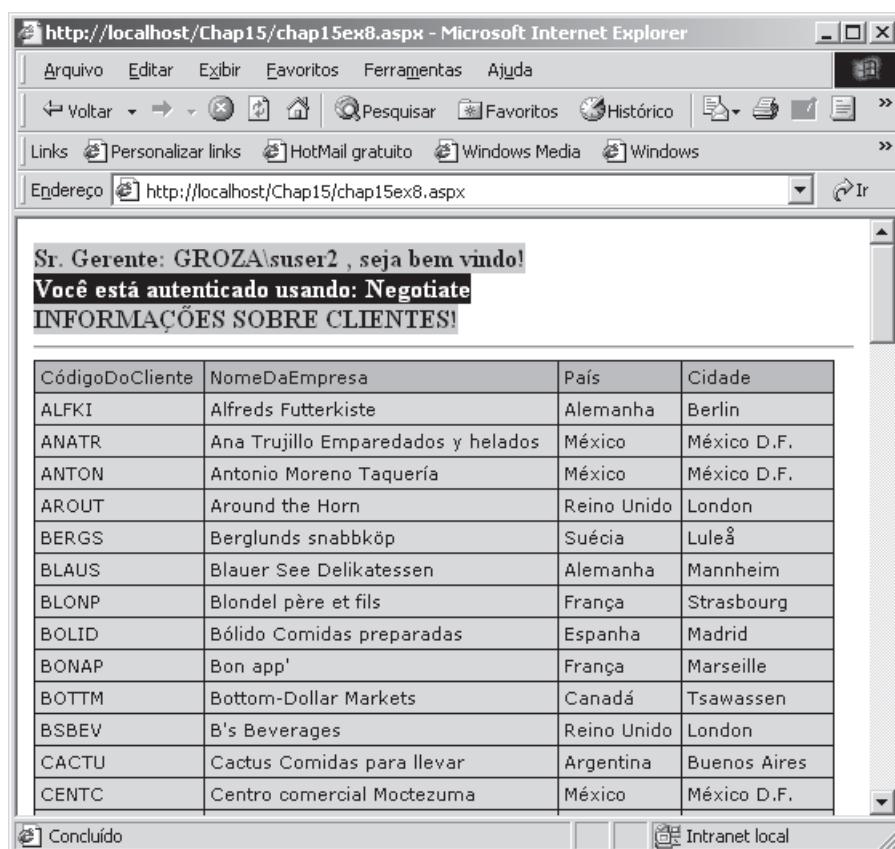


Figura 15.15: Usuário suser2, pertencente ao grupo Gerentes.

Para garantir que a página de logon seja solicitada novamente, feche o navegador, faça o logoff do sistema operacional (Iniciar -> Desligar -> Efetuar o logoff de...). Faça o logon como administrador e acesse a página Chap15ex8.aspx. Na tela de logon digite as seguintes informações:

Nome do usuário: suser3

Senha: abc12345

Domínio: GROZA

Como o usuário suser3 não pertence ao grupo Gerentes, você obterá os resultados indicados na Figura 15.16.

Observe que, para obter as informações sobre o cliente autenticado e o tipo de autenticação, simplesmente utilizamos propriedades do objeto User.Identity:

```
string NomeDoUsuario      = User.Identity.Name;
string TipoDeAutenticacao = User.Identity.AuthenticationType;
```

Para determinar se o usuário pertence ou não ao grupo de gerentes, utilizamos o método IsInRole, para o qual passamos, como parâmetro, o nome do grupo, no formato: DOMÍNIO\NomeDoGrupo:

```
if (User.IsInRole("GROZA\\Gerentes"))
```

O restante do código dispensa maiores comentários.

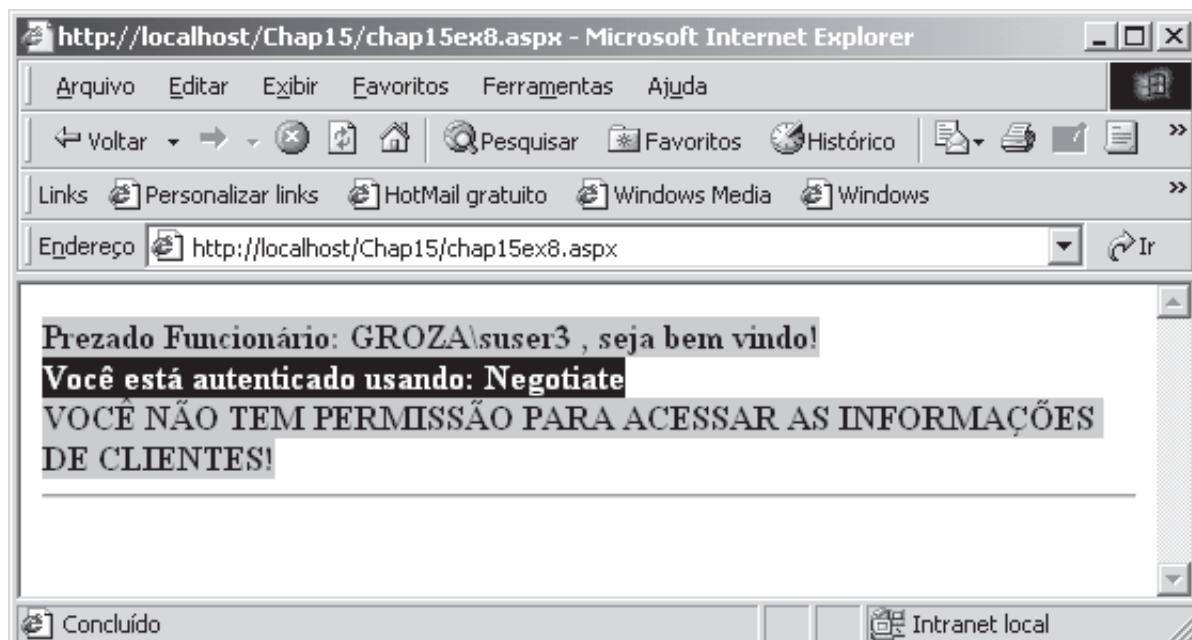


Figura 15.16: Usuário suser3 não pertence ao grupo Gerentes.

Conclusão

Neste capítulo apresentamos uma variedade de assuntos que podem ser úteis para a utilização nas aplicações Web, com ASP.NET, que o leitor venha a desenvolver.

Iniciamos o capítulo com os seguintes exemplos práticos:

- ◆ Como limitar o controle de uma lista, com base no valor selecionado em outra lista.
- ◆ Utilizando o controle DataGrid para editar dados.
- ◆ Um exemplo prático, no qual estaremos exibindo em uma página ASP.NET, dados de uma planilha do Excel.

Em seguida apresentamos o conceito de “Code Behind”, que é uma técnica utilizada para facilitar a separação entre o código responsável pela lógica e o código responsável pela apresentação da página. Com o uso de Code Behind podemos colocar os mecanismos de herança e reaproveitamento de código em prática.

Seguindo na apresentação da nossa “Caixa de Ferramentas”, falamos sobre dois importantes objetos: HttpRequest e HttpResponse. Com estes objetos podemos ter um controle mais eficaz sobre a requisição que é enviada pelo navegador do cliente e sobre a resposta que é enviada de volta pelo servidor.

Depois tratamos sobre as diretivas de página. Uma diretiva é, geralmente, incluída no início da página e é utilizada para instruir o servidor sobre como efetuar o processamento da página.

A assunto final foi sobre as configurações de segurança através da utilização de código.

Com este capítulo encerramos a nossa “longa jornada”, em busca do aprendizado do ASP.NET. A seguir coloco alguns Anexos que apresentam conceitos básicos, utilizados nos capítulos do livro. Recomendo que você leia os anexos sobre assuntos nos quais não se sente confiante. Para referência segue o conteúdo de cada anexo:

- ◆ Anexo I – Tags básicas do HTML, utilizadas nos exemplos do livro. Criação de contas de usuários e grupos de usuários no Windows 2000.
- ◆ Anexo II – O Modelo Relacional de Dados.
- ◆ Anexo III – A linguagem SQL.
- ◆ Anexo IV – Fontes adicionais de informação sobre ASP.NET, na Internet.

Introdução

Este capítulo não faz parte do livro impresso: “ASP.NET: Uma Nova Revolução na Construção de Sites e Aplicações Web”. O presente capítulo somente estará disponível na Internet, no formato .pdf, para download.

Para os leitores que adquiriram o livro, este capítulo é um brinde, uma vez que são tratados novos assuntos, os quais não foram tratados nos capítulos do livro. Para os interessados em adquirir o livro, este capítulo serve para que o leitor tenha uma idéia do estilo do autor.

IMPORTANTE: Para o completo entendimento deste capítulo, são necessários os conhecimentos tratados nos Capítulos 1, 2, 3, 4, 5, 6, 7, 8 e 9 do livro. Os aspectos básicos da linguagem C# e da criação de páginas ASP.NET não serão novamente detalhados.

Vamos iniciar o capítulo falando sobre o “Tratamento de Erros” em páginas ASP.NET. Fazer o tratamento de erros torna nossas aplicações, digamos, “mais elegantes”. Quando ocorre um determinado erro, este é interceptado pelo código de tratamento de erro e uma mensagem “mais esclarecedora”, sobre os possíveis motivos do erro, pode ser gerada. Se não for feito o tratamento de erros, o usuário terá que se contentar com a mensagem padrão emitida pelo servidor Web.

Veremos que com o ASP.NET é possível fazer um tratamento estruturado dos erros que ocorrem em uma página ou aplicação Web. Também podemos fazer com que o Administrador do sistema seja notificado dos erros, seja escrevendo no Log de Eventos do Sistema Operacional ou fazendo com que o código de tratamento de erros envie uma mensagem de e-mail para a Caixa Postal do Administrador.

Concordo que tratamento de erros não é, nem de longe, um assunto tão interessante quanto acesso a Bases de Dados usando ADO.NET, segurança, ou outro qualquer; porém, para que nossas aplicações se comportem de uma maneira adequada, é de fundamental importância que seja dada a devida atenção ao item: “Tratamento de Erros”.

Faremos uma breve revisão das estruturas Try...Catch...Finally, já apresentadas no Capítulo 5, para que o leitor possa relembrar da sintaxe e da forma de utilização destes comandos. Para uma descrição detalhada destes comandos, consulte o Capítulo 5.

Para entender o funcionamento do mecanismo para tratamento de erros, utilizaremos diversos exemplos práticos, onde o leitor poderá ver em

CAPÍTULO 16

Tratamento de Erros e Gerenciamento de Estado

funcionamento o tratamento de erros em páginas ASP.NET. Apresentaremos exemplos onde são tratadas exceções genéricas e também exceções específicas, tais como uma tentativa de conexão com um servidor SQL Server 2000 que não existe.

Para o desenvolvimento de Sites e Aplicações Web profissionais, baseadas no Framework .NET, mais especificamente na tecnologia de páginas ASP.NET, o correto tratamento de erros e exceções é um tópico fundamental. Muitas vezes, devido a prazos esgotados e pressões pela entrega de um aplicativo, o tratamento de erros é negligenciado, o que resulta em programas, no mínimo, “deselegantes” e instáveis, os quais não atendem as necessidades dos usuários.

Tratamento de Erros no Framework .NET

O tão sonhado “Aplicativo Livre de Erros” ainda é um sonho distante. Aplicações são projetadas, concebidas, implementadas, testadas, distribuídas e utilizadas por seres humanos. Seres humanos não são perfeitos, cometem erros, de forma que é mais do que natural que, por maiores que sejam os cuidados, os aplicativos contenham erros.

Determinados erros são possíveis de serem tratados. Tratar um erro significa fazer com que um aplicativo ou página ASP.NET “comporte-se” de uma maneira elegante quando o erro ocorrer. O exemplo clássico é o erro que é gerado quando mandamos o aplicativo ler um arquivo no disquete, porém não colocamos um disquete no drive. Uma maneira “deselegante” do programa comportar-se seria simplesmente emitir uma mensagem de erro e encerrar o aplicativo. Uma maneira mais “elegante” seria detectar que ocorreu um erro e, pelo código do erro, informar que deve ser inserido um drive no disquete. O usuário coloca o disquete no drive e clica em um botão OK e pronto, o erro foi contornado sem maiores traumas.

NOTA: Para maiores informações sobre as estruturas Try...Catch...Finally, consulte o Capítulo 5.

Com ASP.NET tivemos muitas melhorias no tratamento de erros, em relação às versões anteriores do ASP. Os objetos do Framework .NET para tratamento de erros são mais poderosos e fáceis de utilizar. Além disso podemos fazer o tratamento estruturado de erros, utilizando as estruturas try...catch...finally.

Uma das melhorias mais significativas é que o tratamento de erros é implementado pelo próprio CLR – Common Language Runtime, parte integrante do Framework .NET. Com isso as técnicas e comandos de tratamento de erros são independentes da linguagem utilizada. Por exemplo, podemos passar uma exceção gerada em uma página ASP.NET, codificada em C#, para um componente de tratamento de erros criado com VB.NET. As estruturas, mensagens e códigos de erro são os mesmos, independente da linguagem, pois estes elementos são parte integrante do Framework .NET. O CLR usa o mecanismo de exceções para fazer o tratamento de erros. Toda vez que um erro acontece durante a execução de um programa ou de uma página ASP.NET, uma exceção é disparada. O tipo de exceção gerada depende do erro ocorrido. Neste capítulo aprenderemos a detectar a ocorrência de exceções e a tratá-las.

Exceções e a Classe Exception

Quando uma exceção é gerada devido a um erro em um programa ou em uma página ASP.NET, um objeto derivado da classe `Exception` é criado. Através das propriedades da classe `Exception`, podemos acessar uma série de informações sobre a origem do erro que gerou a exceção.

A classe Exception faz parte do namespace System – System.Exception. Esta classe é a classe base, da qual são derivadas todas as outras classes que fazem o tratamento de exceções.

Uma exceção é uma resposta do sistema a ocorrências de condições anormais geradas devido a erros, durante o processamento de um programa ou página ASP.NET. O CLR (Common Language Runtime) nos fornece um modelo para o tratamento de exceções. Este modelo é baseado na criação de objetos de exceção baseados na classe Exception ou nas classes derivadas de Exception. Outro princípio básico do modelo de tratamento de exceções do CLR é a separação entre o código que contém a lógica do programa e o código para o tratamento de exceções, através da utilização das estruturas try...catch...finally, descritas e exemplificadas no Capítulo 5.

Quando uma exceção ocorre em um bloco try, o controle de execução é passado para o bloco catch correspondente, para que seja feito o tratamento da exceção. Se o caminho de execução possui vários métodos, cada um chamando o outro em seqüência, a exceção será repassada até que o CLR encontre um tratamento para a exceção. Se nenhum dos métodos chamados contiver um tratamento para a exceção, o tratador default será chamado pelo CLR para exibir o nome da exceção, uma mensagem de erro e informações sobre o método onde a exceção foi gerada.

Temos duas categorias de exceções baseadas na classe Exception:

- ◆ **SystemException:** São classes de exceções predefinidas, baseadas na classe SystemException.
- ◆ **ApplicationException:** São exceções em nível de aplicação, que são definidas pelos usuários, sendo baseadas na classe ApplicationException.

Diversas informações sobre uma exceção podem ser obtidas a partir das propriedades da classe na qual se baseia o objeto de exceção que foi criado. Normalmente este objeto é baseado na classe Exception.

Na Tabela 16.1 temos a descrição das principais propriedades da classe Exception.

Tabela 16.1 Propriedades da classe System.Exception.

Propriedade	Descrição
HelpLink	Define ou retorna um link para um arquivo de ajuda associado com a exceção.
InnerException	Obtém uma referência ao objeto que representa a exceção que ocorreu em primeiro lugar, ou seja, a primeira exceção gerada na pilha de chamada dos métodos relacionados com o erro que disparou a exceção. Conforme descrevemos anteriormente, uma exceção pode ter sido gerada em um método, porém tratada em um outro método chamado diretamente pelo método que gerou a exceção ou chamado por um método que foi chamado pelo método original e assim por diante.
MessageProperty	Retorna a mensagem associada com a exceção.
TargetSite	Retorna o nome do método original, onde foi inicialmente disparada a exceção.
ToString	Retorna o nome completo da exceção, a mensagem de erro, e outras informações sobre a exceção.

NOTA: Veremos exemplos dos usos destas propriedades nos exemplos práticos, mais adiante neste capítulo.

Revisando as Estruturas Try...Catch...Finally

A seguir coloco, resumidamente, o funcionamento e exemplos dos comandos Try...Catch...Finally, utilizados para o tratamento estruturado de exceções.

Tratar exceções é de fundamental importância para que um programa não seja encerrado inesperadamente. Uma exceção pode acontecer durante o processamento do programa, quando algo inesperado acontece e deve ser tratado pelo programa, para que este não seja encerrado sem que o usuário saiba o que está acontecendo.

Dois exemplos típicos de exceções:

- ◆ O programa tenta fazer uma leitura no disquete e não existe disquete no drive.
- ◆ Uma divisão por zero.

As exceções devem ser detectadas e opções devem ser oferecidas para o usuário do programa. Por exemplo, no caso do disquete que não está no drive, a exceção deve ser detectada e o programa deve exibir uma mensagem solicitando que o usuário insira um disquete no drive. Este procedimento é muito mais amigável do que simplesmente encerrar o programa.

Outra grande vantagem do Framework .NET é que o tratamento de exceções é padronizado, independentemente da linguagem que está sendo utilizada. Uma exceção gerada em um componente escrito em C++ pode ser tratada em um cliente escrito em C# e vice-versa.

Neste tópico veremos como tratar exceções com a linguagem C#.

Utilizando “try” e “catch”

Para definir o tratamento de exceções em nossos programas precisamos organizar os códigos em um bloco try e um bloco catch. Dentro do bloco try colocamos o código que pode gerar uma exceção – por exemplo os comandos que farão a leitura de um arquivo no disquete pois, se o disquete não estiver no drive, será gerada uma exceção. O código para o tratamento da exceção é colocado dentro do bloco catch.

Vamos apresentar um exemplo onde utilizamos try e catch para fazer o tratamento de exceções. O nosso programa solicita que o usuário digite dois números. Depois o programa faz a divisão dos números e exibe o resultado. Para forçar uma exceção vamos fornecer um valor zero para o segundo número, de tal forma que o programa, ao tentar fazer uma divisão por zero, irá gerar uma exceção.

Considere o exemplo da Listagem 16.1.

Listagem 16.1 – Tratamento de exceções com try e catch – ex1cap16.cs

```
using System;
```

```
class ex1cap16
```

```
{  
    // Exemplo 1 - Capítulo 16.  
    // Tratamento de exceções com try e catch.  
    // Por: Júlio Battisti  
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA  
  
    public static void Main()  
    {  
  
        // Início do bloco try.  
        // Contém o código que pode gerar a exceção.  
        try  
  
        {  
            // Declaração das variáveis.  
  
            int divisao;  
  
            // Entrada dos valores de x e y  
  
            Console.Write("Digite o NUMERADOR ->");  
            String Aux1=Console.ReadLine();  
  
            Console.Write("Digite o DENOMINADOR ->");  
            String Aux2=Console.ReadLine();  
  
            // Cálculo da divisão.  
            divisao = Convert.ToInt32(Aux1) / Convert.ToInt32(Aux2);  
  
            // Exibição dos resultados.  
            Console.WriteLine("O valor da DIVISÃO é -> {0}",divisao);  
        }  
        // Final do bloco try.  
  
        // Início do bloco catch.  
        // Código que será executado se uma exceção  
        // for gerada no bloco try.
```

```

    catch (Exception e)

    {

        Console.WriteLine("FOI GERADA A SEGUINTE EXCEÇÃO: " + e.Message);

    }

    // Final do bloco catch.

}

}

```

Digite o exemplo da Listagem 16.1 e salve-o em um arquivo chamado ex1cap16.cs, na pasta C:\Meus documentos. Compile e execute o exemplo da Listagem 16.1. Digite 10 para o numerador e 2 para o denominador. Você obterá os resultados indicados na Figura 16.1.

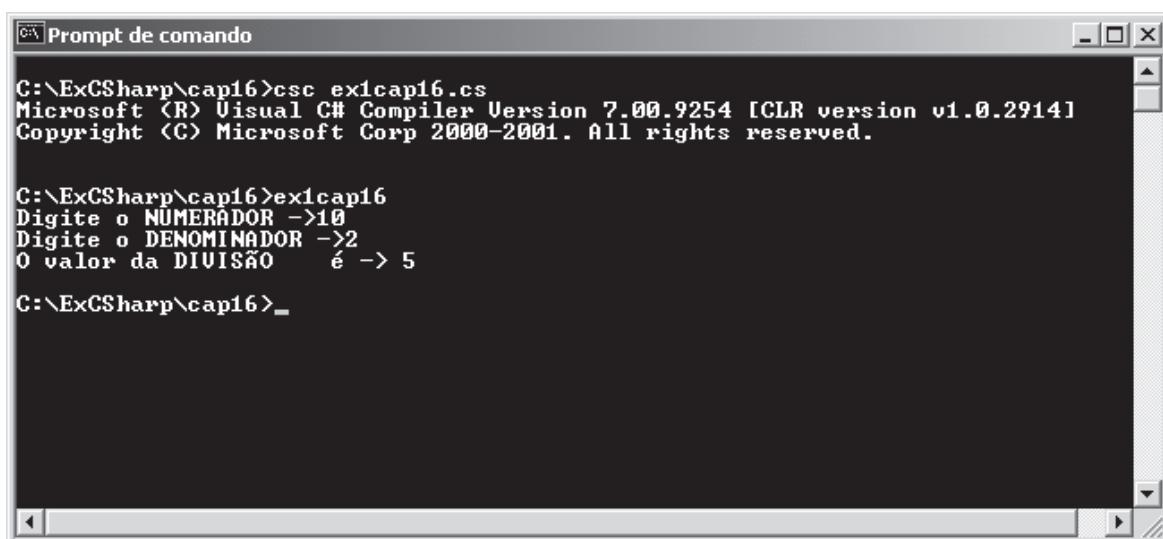


Figura 16.1 Executando, sem exceções, o programa ex1cap16.exe.

Observe que o programa executa normalmente. Agora vamos forçar uma exceção, e para isso digitaremos 0 para o segundo valor, forçando uma divisão por zero. Vamos executar novamente o programa. Digite 10 para o primeiro valor e 0 para o segundo. Você obterá os resultados indicados na Figura 16.2.

Neste segundo caso, ao tentar fazer uma divisão por zero, uma exceção será gerada. Ao ser gerada a execução o código do bloco catch será executado. O bloco catch recebe um parâmetro do tipo Exception. Exception é uma classe do namespace System – System.Exception, conforme descrito anteriormente. Uma das propriedades desta classe é Message, a qual contém a mensagem associada com a exceção, conforme descrito na Tabela 16.1. No nosso exemplo a mensagem é: “Attempted to divide by zero”, o que confirma a nossa tentativa de fazer uma divisão por zero.

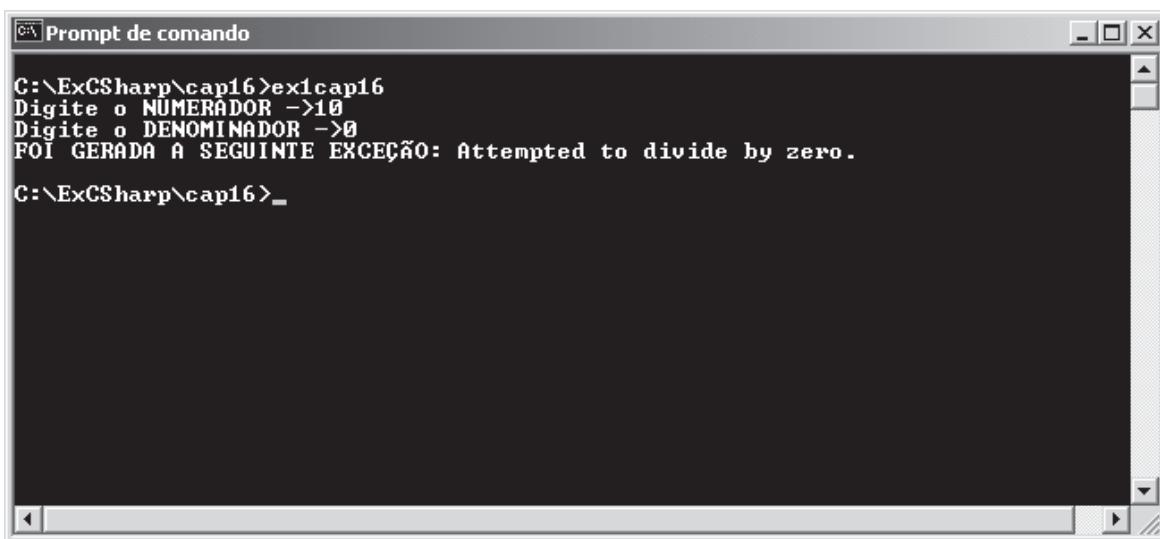


Figura 16.2 Executando, forçando uma exceção, o programa ex1cap16.exe.

Existem classes que tratam exceções mais específicas, como por exemplo:

```
System.OutOfMemoryException  
System.OverflowException  
System.NullReferenceException  
System.NotSupportedException  
System.NotImplementedException  
System.NotFiniteNumberException  
System.MissingMethodException  
System.MissingMemberException  
System.MissingFieldException  
System.MethodAccessException  
System.MemberAccessException  
System.InvalidProgramException  
System.InvalidOperationException  
System.InvalidCastException  
System.IndexOutOfRangeException  
System.FormatException  
System.FieldAccessException  
System.ExecutionEngineException  
System.EntryPointNotFoundException  
System.DuplicateWaitObjectException  
System.DllNotFoundException  
System.DivideByZeroException
```

Utilizando "try" e "finally"

Em algumas situações podemos querer que um determinado bloco de código seja executado, mesmo que não tenha sido gerada nenhuma exceção. Para que isso seja possível podemos utilizar finally ao invés de catch ou em conjunto com catch; desta forma se ocorrer a exceção, o bloco catch será executado e o bloco finally será sempre executado, quer tenha ocorrido ou não uma exceção. O código dentro do bloco finally é sempre executado, mesmo que não tenha sido gerada nenhuma exceção.

Vamos modificar um pouco o exemplo anterior.

Considere o exemplo da Listagem 16.2

Listagem 16.2 – Tratamento de exceções com try e finally – ex2cap16.cs

```
using System;

class ex2cap16
{
    // Exemplo 2 - Capítulo 16.
    // Tratamento de exceções com try e finally.
    // Por: Júlio Battisti
    // MCP, MCP+I, MCSE, MCSE+I, MCSE, MCDBA

    public static void Main()
    {
        // Bloco try.
        // Contém o código que pode gerar a exceção.

        try
        {
            // Declaração das variáveis.

            int divisao;

            // Entrada dos valores de x e y

            Console.WriteLine("Digite o NUMERADOR ->");
            String Aux1=Console.ReadLine();

            Console.WriteLine("Digite o DENOMINADOR ->");
            String Aux2=Console.ReadLine();
        }
    }
}
```

```

// Cálculo da divisão.

divisao = Convert.ToInt32(Aux1) / Convert.ToInt32(Aux2);

// Exibição dos resultados.

Console.WriteLine("O valor da DIVISÃO é -> {0}",divisao);

}

// Final do bloco try.

// Início do bloco finally.

// Código que será executado mesmo que nenhuma exceção
// seja gerada no bloco try.

finally

{

Console.WriteLine("CÓDIGO EXECUTADO TENHA OU NÃO SIDO GERADA UMA EXCEÇÃO");

}

// Final do bloco finally.

}

}

```

Digite o exemplo da Listagem 16.2 e salve-o em um arquivo chamado ex2cap16.cs, na pasta C:\Meus documentos. Compile e execute o exemplo da Listagem 16.2. Digite 10 para o numerador e 2 para o denominador. Você obterá os resultados indicados na Figura 16.3. Observe que o código do bloco finally foi executado, mesmo sem ter sido gerada nenhuma exceção.

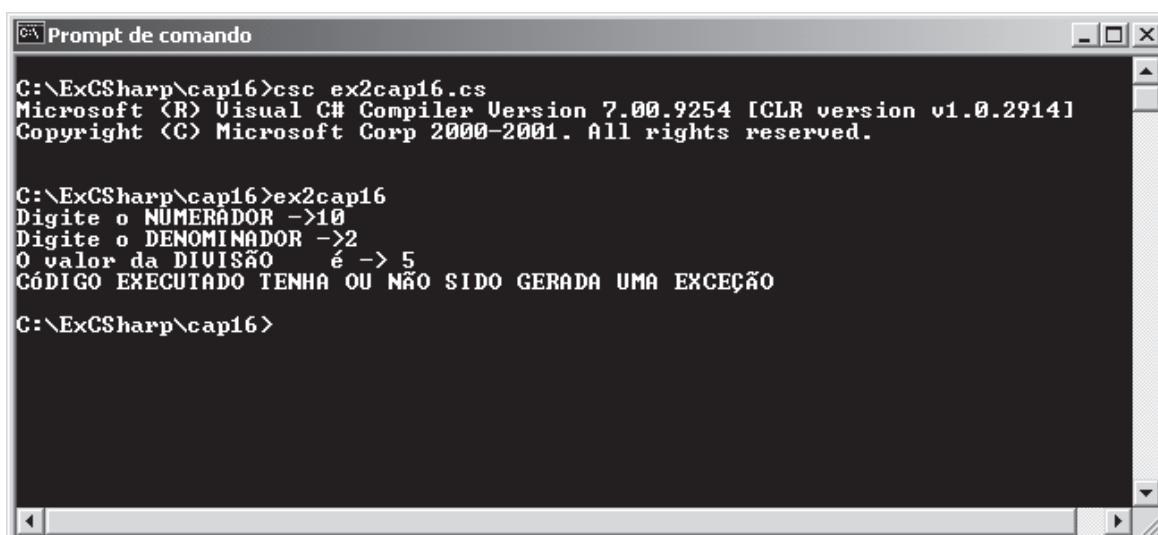


Figura 16.3 Executando, sem exceções, o programa ex2cap16.exe.

Agora vamos forçar uma exceção; para isso digitaremos 0 para o segundo valor, forçando uma divisão por zero. Vamos executar novamente o programa. Digite 10 para o primeiro valor e 0 para o segundo. Na Figura 16.4 é aberta uma janela indicando que ocorreu uma exceção no programa. Esta janela é aberta porque não temos um bloco catch para fazer o tratamento da exceção.

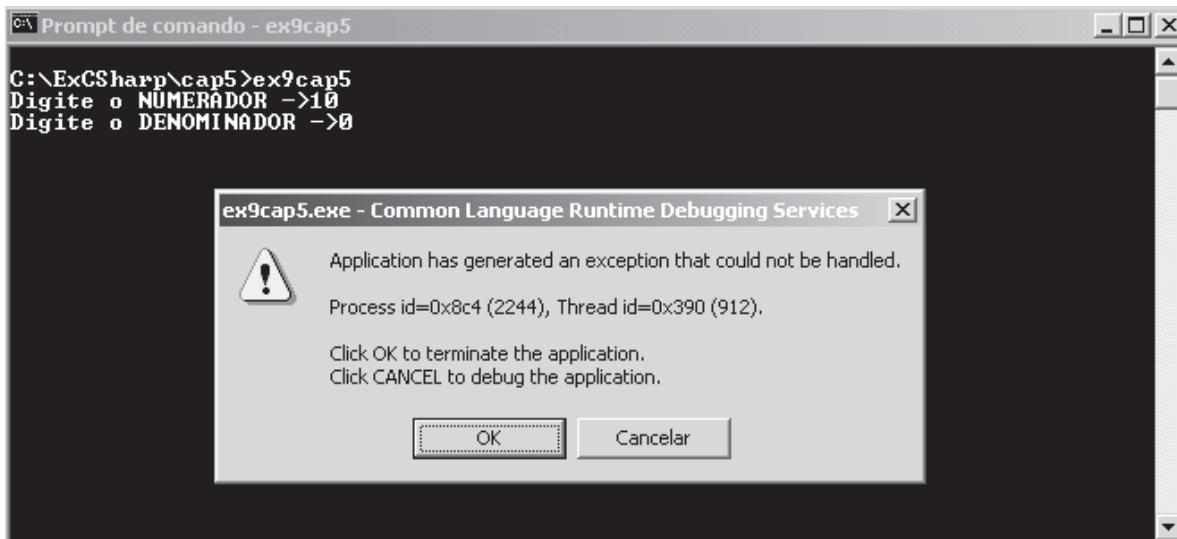


Figura 16.4 Aviso de que uma exceção foi gerada.

Neste caso, como não havia um bloco catch, a exceção não foi tratada. Por isso que surgiu a janela indicada na Figura 16.4. Dê um clique em OK para fechar a janela de aviso e observe que o código do bloco finally foi executado, mesmo tendo sido gerada uma exceção, conforme indicado pela Figura 16.5:

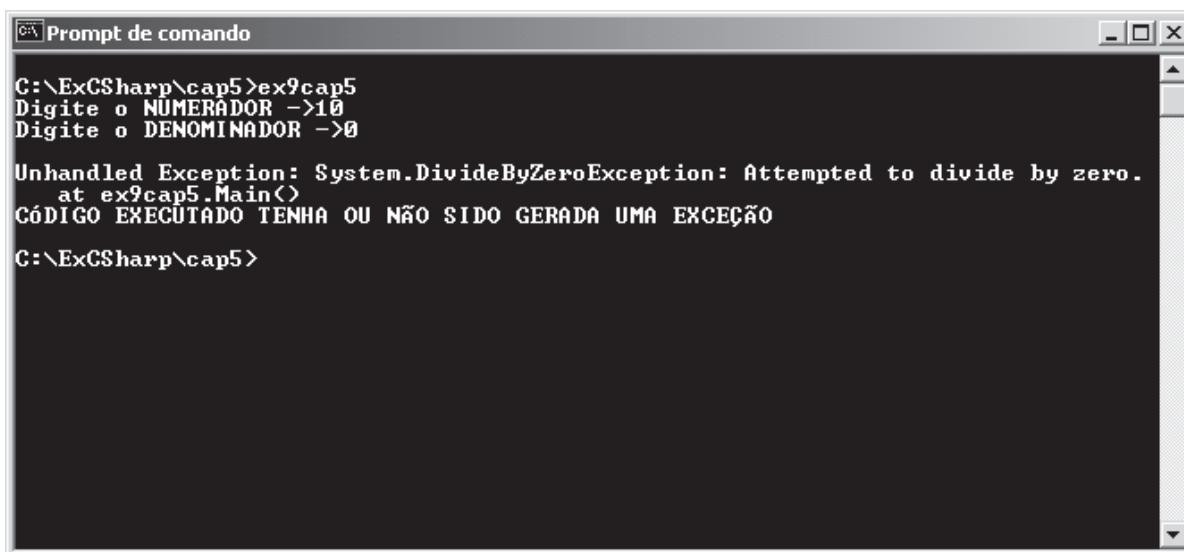


Figura 16.5 O código do bloco finally é sempre executado.

Com essa breve revisão (os comandos Try...Catch...Finally foram detalhadamente explicados no Capítulo 5), podemos apresentar alguns exemplos de tratamento de exceções em páginas ASP.NET.

Múltiplos Blocos Catch e Tratamento de Exceções Específicas

Para um bloco Try, podemos ter múltiplos blocos Catch. Por exemplo, se em uma página ASP.NET prevemos a possibilidade de serem gerados três diferentes tipos de exceções, podemos criar um bloco Try, com três blocos Catch, um para tratar cada um dos tipos de exceções previstos. Por exemplo, podemos criar um bloco Catch para tratar uma exceção do tipo System.OutOfMemoryException, um para tratar uma exceção do tipo System.OverFlowException e uma Terceira para tratar uma exceção do tipo System.NullReferenceException.

Além dos blocos para tratar exceções específicas, podemos e devemos colocar um bloco Catch, bem no final, para tratar exceção genérica – System.Exception. Desta forma não corremos o risco de termos uma exceção não tratada, a qual fará com que a página deixe de ser carregada ou que o programa termine de maneira inesperada. Considere o exemplo genérico da Listagem 16.3:

Listagem 16.3 – Múltiplos blocos Catch.

```
try
```

```
Comando 1
```

```
Comando 2
```

```
...
```

```
Comando n
```

```
// Bloco Catch para tratar uma exceção do tipo OutOfMemoryException.
```

```
catch Exceção1 As OutOfMemoryException
```

```
Comando 1
```

```
Comando 2
```

```
...
```

```
Comando n
```

```
// Bloco Catch para tratar uma exceção do tipo FileNotFoundException.
```

```
catch Exceção2 As FileNotFoundException
```

```
Comando 1
```

```
Comando 2
```

```
...
```

```
Comando n
```

```
// Bloco Catch para tratar qualquer tipo de exceção, ou seja, uma exceção Genérica.
```

NOTA: Para uma referência completa a todos os objetos para tratamentos de exceções disponíveis no Framework .NET, consulte a documentação do produto ou o site www.msdn.com/net.

```
// Este bloco tratará qualquer exceção que ocorra neste programa, com exceção das
// Exceções dos tipos: OutOfMemoryException e FileNotFoundException, já tratadas
// nos blocos Catch anteriormente.

catch ExceçãoGenérica As Exception

Comando 1

Comando 2

...
Comando n
```

Um Exemplo de Tratamento de Exceção do Tipo SqlException

Vamos apresentar uma página ASP.NET onde tentamos fazer uma conexão com um servidor SQL Server 2000 que não existe. Esta tentativa irá gerar uma exceção do tipo `SqlException`, derivada da classe `System.Data.SqlClient.SqlException`. Iremos criar um bloco Try com dois blocos Catch: Um para tratar a exceção específica, do tipo `SqlException`, e outro para tratar uma exceção genérica, do tipo `Exception`.

O exemplo proposto está demonstrado na Listagem 16.4.

Listagem 16.4 – Usando Try...Catch...Finally em um página ASP.NET.

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

<html>

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

protected void Page_Load(Object Src, EventArgs E )
{
    //Início do bloco Try.

    try
    {

        // Crio uma conexão com o banco de dados pubs localizado no servidor local.
        // Vamos acessar a instância SERVIDORXYZ\NETSDK.
        // O Servidor: SERVIDORXYZ não existe.

        // Isso fará com que seja gerada uma exceção do tipo SqlException,
```

```
// a qual será tratada em um bloco Catch específico para o tratamento desta exceção.
```

```
String strCon = "server=SERVIDORXYZ\\NETSDK;uid=sa;pwd=;database=pubs";  
SqlConnection MinhaConexão = new SqlConnection(strCon);  
MinhaConexão.Open();
```

```
// Declaro uma variável do tipo String: auxPropriedades.  
// A variável auxPropriedades irá conter o valor das propriedades  
// da conexão minhaConexão.
```

```
String auxPropriedades;  
auxPropriedades = "Propriedades da conexão:";  
  
auxPropriedades = auxPropriedades + "\n\n" + "ConnectionString: " +  
MinhaConexão.ConnectionString.ToString();  
  
auxPropriedades = auxPropriedades + "\n\n" + "Database: " +  
MinhaConexão.Database.ToString();
```

```
auxPropriedades = auxPropriedades + "\n\n" + "DataSource: " +  
MinhaConexão.DataSource.ToString();  
  
auxPropriedades = auxPropriedades + "\n\n" + "State: " +  
MinhaConexão.State.ToString();
```

```
ExibePropriedades.Font.Bold=true;  
ExibePropriedades.Text=auxPropriedades;
```

```
// Final do Bloco Try.
```

```
}
```

```
// Início do bloco Catch para tratamento da exceção do tipo SqlException.
```

```
catch (SqlException SqlEx)  
{
```

```
    Response.Write("<b> Iniciando o tratamento da exceção do tipo SqlException</b><br>");

Response.Write(SqlEx.ToString() + "<p>");

// Corrijo o nome do Servidor SQL Server e estabeleço a conexão corretamente.

String strCon = "server=SERVIDOR\\NETSDK;uid=sa;pwd=;database=pubs";
SqlConnection MinhaConexão = new SqlConnection(strCon);

MinhaConexão.Open();

// Declaro uma variável do tipo String: auxPropriedades.

// A variável auxPropriedades irá conter o valor das propriedades
// da conexão minhaConexão.

String auxPropriedades;
auxPropriedades = "Propriedades da conexão:";

auxPropriedades = auxPropriedades + "\n\n" + "ConnectionString: " +
    MinhaConexão.ConnectionString.ToString();

auxPropriedades = auxPropriedades + "\n\n" + "Database: " +
    MinhaConexão.Database.ToString();

auxPropriedades = auxPropriedades + "\n\n" + "DataSource: " +
    MinhaConexão.DataSource.ToString();

auxPropriedades = auxPropriedades + "\n\n" + "State: " +
    MinhaConexão.State.ToString();

ExibePropriedades.Font.Bold=true;
ExibePropriedades.Text=auxPropriedades;

// Final do Bloco Catch para tratamento da exceção do tipo SqlException.

}

// Início do bloco Catch para tratamento da exceção do tipo Exception.
```

```
catch (Exception ex)
{
    Response.Write("Tratando uma exceção genérica<p>");

}

// Bloco Finally. É sempre executado, mesmo que nenhuma exceção tenha sido gerada.

finally
{
    Response.Write("<b>Código do bloco Finally sendo executado !!!</b><p>");

}
}

}

</script>

<body>
<h3><font face="Verdana">Classe SqlConnection!!!</font></h3>

<asp:TextBox
    runat=server
    id="ExibePropriedades"
    Text=""
    Rows="10"
    Cols="70"
    Font_Face="Arial"
    Font_Size="3"
    BackColor="lightblue"
    TextMode="MultiLine"
    />

</body>
</html>
```

Digite o código da Listagem 16.4 e salve-o em um arquivo chamado chap16ex1.aspx, na pasta chap16, dentro da pasta wwwroot, conforme descrito no item: “Check List para acompanhar os exemplos deste livro”, no Capítulo 6.

Para acessar esta página utilize o seguinte endereço:

<http://localhost/chap16/chap16ex1.aspx>

Ao carregar a página você irá obter uma página semelhante à página indicada na Figura 16.6.

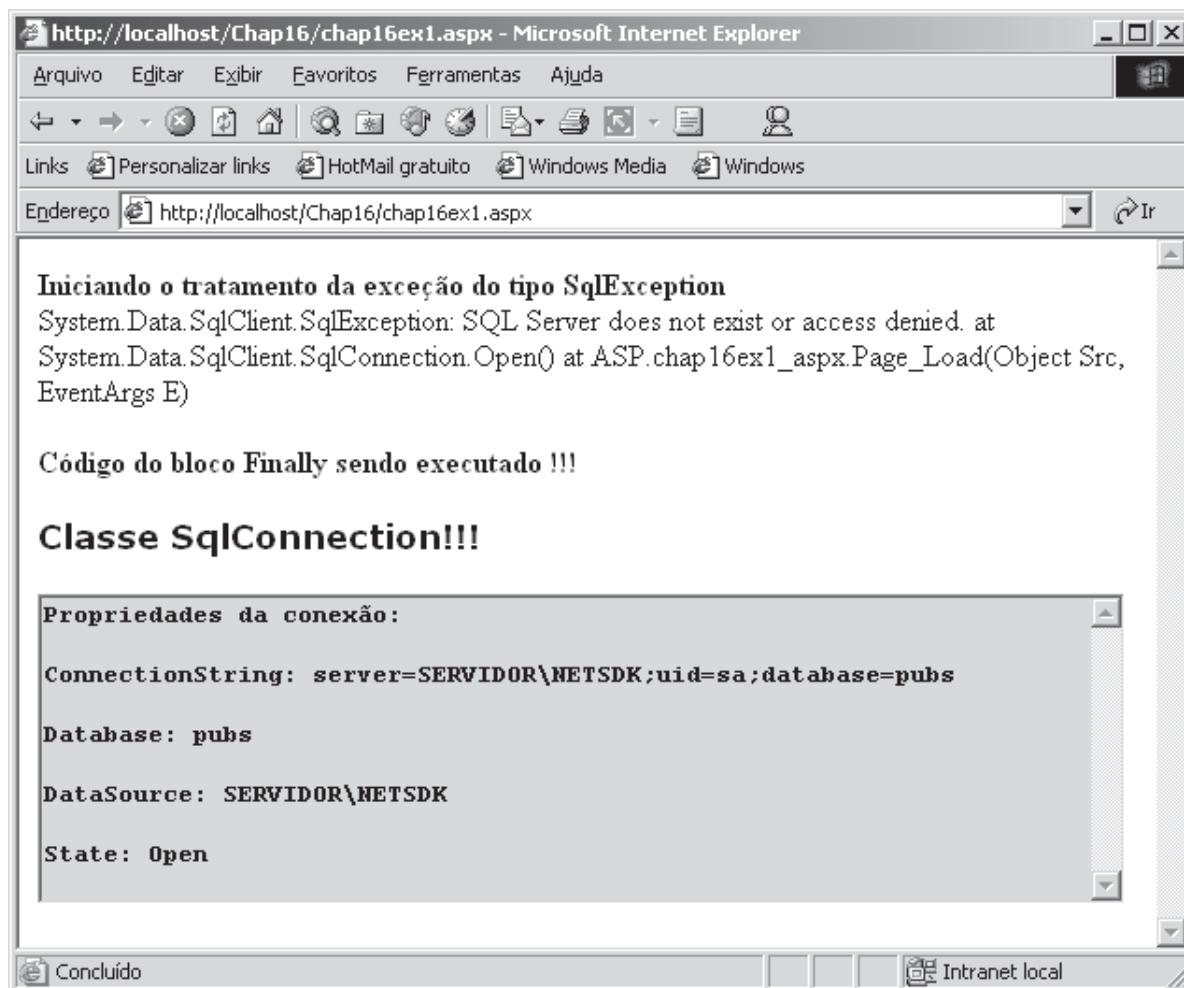


Figura 16.6 Tratamento de Exceções em uma página ASP.NET.

Comentários sobre o código do exemplo – Chap16Ex1.aspx.

No bloco Try tentamos fazer a conexão com um servidor SQL Server 2000 que não existe:

```
String strCon = "server=SERVIDORXYZ\\NETSDK;uid=sa;pwd=;database=pubs";  
SqlConnection MinhaConexão = new SqlConnection(strCon);
```

O servidor SERVIDORXYZ não existe. Neste caso será gerada uma exceção do tipo `SqlException`, a qual será tratada pelo primeiro bloco `Catch`. Este bloco é disparado em resposta a uma exceção do tipo `SqlException`, que é o tipo de exceção que é disparada, quando tentamos conectar com um servidor SQL Server que não existe.

No primeiro bloco Catch retornamos as seguintes mensagens:

1. Iniciando o tratamento da exceção do tipo SqlException.
2. System.Data.SqlClient.SqlException: SQL Server does not exist or access denied. at System.Data.SqlClient.SqlConnection.Open() at ASP.chap16ex1_aspx.Page_Load(Object Src, EventArgs E).

A primeira mensagem é simplesmente um texto retornado por um comando Response.Write.

A segunda mensagem é retornada a partir do método ToString do objeto SqlEx (que é do tipo SqlException). O método ToString simplesmente retorna um texto descritivo da exceção. A seguir temos o comando que retorna esta mensagem:

```
Response.Write(SqlEx.ToString() + "<p>");
```

No restante deste primeiro bloco Catch repetimos os comandos para estabelecer uma conexão com o servidor SQL Server, porém agora utilizamos o nome correto:

```
String strCon = "server=SERVIDOR\\NETSDK;uid=sa;pwd=;database=pubs";
SqlConnection MinhaConexão = new SqlConnection(strCon);
```

Com isso a conexão é estabelecida e as propriedades da conexão são exibidas, no corpo da página em um Web Server control do tipo TextBox com múltiplas linhas.

Colocamos um outro bloco Catch, o qual será disparado em resposta a qualquer exceção que não seja do tipo SqlException. Por exemplo, se tivéssemos uma operação de divisão na página e houvesse uma tentativa de divisão por zero (operação não permitida), uma exceção seria gerada. Como esta exceção não seria do tipo SqlException, ela seria tratada pelo bloco Catch para tratamento de exceções genéricas – exceções do tipo Exception:

```
// Início do bloco Catch para tratamento da exceção do
tipo Exception.

catch (Exception ex)
{
    Response.Write("Tratando uma exceção genérica<p>");
}
```

NOTA: Para maiores informações sobre Web Server Controls, consulte os Capítulos 7, 8 e 9.

O bloco Finally será sempre executado, independente de ter sido, ou não, gerada alguma exceção.

Conclusão

Neste capítulo, que é um brinde para o amigo leitor, falei sobre os aspectos básicos do tratamento de erros/exceções em páginas ASP.NET.

Fizemos uma breve revisão das estruturas Try...Catch...Finally e apresentamos alguns exemplos práticos. Com isso você tem uma boa idéia sobre o tratamento estruturado de exceções.

Um bom estudo a todos e não deixem de entrar em contato através do seguinte endereço:

webmaster@juliobattisti.com.br

Introdução

Este anexo trata de dois assuntos utilizados nos exemplos do livro:

- ◆ A utilização da linguagem HTML.
- ◆ A criação de contas de usuários e grupos no Windows 2000.

Um Passeio Pelo HTML

Conforme vimos neste livro, o ASP.NET fornece um conjunto bastante completo de controles. Os chamados Server Controls possuem uma rica funcionalidade. Porém, mesmo com todo o poder do ASP.NET, é importante conhecermos as principais tags HTML. Neste anexo veremos a sintaxe para as tags que utilizamos nos exemplos deste livro.

Nas Listagem I.1a e I.1b temos a estrutura básica de uma página ASP.NET.

Listagem I.1a – A estrutura básica de uma página ASP.NET com VB.NET.

```
<html>

<script language="VB" runat="server">
    comando1
    comando2
    ...
    comandos
</script>

<body>
    controles e
    demais
    elementos
    da interface
    - Server controls
    - Web Forms
    - etc
</body>
</html>
```

Listagem I.1b – A estrutura básica de uma página ASP.NET com C#.

```
<html>

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

ANEXO

1

Principais Tags do HTML e Criação de Contas e Grupos no Windows 2000

```

comando1
comando2
...
comandon
</script>

<body>
    controles e
    demais
    elementos
    da interface
    - Server controls
    - Web Forms
    - etc
</body>
</html>

```

Toda página ASP.NET tem esta estrutura. Uma parte inicial onde temos o código, responsável pela lógica da página. Na seção de código, como é chamada, podemos colocar comandos para fazer a conexão com um banco de dados, para realizar cálculos, para responder a eventos que acontecem na página, como por exemplo um clique do usuário em um botão, enfim, toda a lógica de programação necessária ao funcionamento da página. A seção de código é representada pelo seguinte trecho.

```

<script language="C#" runat="server">
    comando1
    comando2
    ...
    comandon
</script>

```

Na segunda parte da página, a partir da tag <body>, colocamos os elementos de apresentação da interface. Nesta parte podemos colocar desde código HTML básico até controles mais avançados como os disponibilizados pelo ASP.NET. Aqui a importância de conhecermos as principais tags da linguagem HTML, uma vez que na seção de apresentação da página podemos utilizar, além dos controles do ASP.NET, qualquer tag HTML válida. Em diversos exemplos deste livro utilizamos as tags HTML para criação de tabelas. Com o uso de tabelas fica mais fácil fazer o alinhamento dos diversos elementos da página.

Tudo o que você já conhece de HTML pode ser utilizado na seção de apresentação da página.

Observe que a estrutura de uma página ASP.NET deixa bem clara a separação entre código ASP.NET e código de apresentação, bem diferente do que acontecia com o ASP 3.0 onde seções de código ASP são intercaladas com seções de código HTML.

Neste Anexo iremos apresentar e exemplificar as tags mais utilizadas. Para uma referência completa da linguagem HTML, você pode consultar o site: www.w3.org.

Outros endereços úteis:

- ◆ www.wdvl.com
- ◆ www.internet.com
- ◆ www.htmlgoodies.com

Criação de Contas de Usuários e Grupos no Windows 2000

Quando falamos sobre segurança nos Capítulos 14 e 15, aprendemos a configurar as permissões de acesso em nível de usuário. Cada usuário, no Windows 2000, é identificado com base na conta que ele utiliza para fazer o logon. Na segunda parte deste anexo falaremos um pouco mais sobre o conceito de contas de usuários, como criar contas. Também falaremos sobre o conceito de grupos, como criar grupos e incluir usuários em um ou mais grupos.

A Estrutura Básica de uma Página HTML

Toda página HTML possui uma estrutura básica, bem definida, conforme indicado na Listagem I.2:

Listagem I.2 – A estrutura básica de uma página HTML.

```
<HTML>
<HEAD>
<TITLE>Título que aparece na Barra de Títulos do navegador do Cliente !!</TITLE>
</HEAD>
<BODY>
    tag HTML
    tag HTML
    ...
    tag HTML
</BODY>
</HTML>
```

Onde temos os seguintes elementos:

<HTML> e </HTML>. Estas tags marcam o início e o fim de uma página HTML.

Toda página é dividida em duas partes:

- ◆ <HEAD>...</HEAD>: É o cabeçalho da página, onde podemos inserir uma série de tags que contêm informações sobre a própria página. A tag mais utilizada na seção de Cabeçalho é <TITLE> </TITLE>, a qual é utilizada para definir o título que é exibido na Barra de Títulos do navegador do cliente.

- ◆ <BODY>...</BODY>: Estas tags definem o “corpo” da página, onde são colocados os elementos que o navegador interpreta e exibe para o usuário. Nesta seção fica o conteúdo principal do documento.

Podemos definir uma cor de segundo plano para a página, utilizando o atributo bgcolor, para a tag body, conforme o exemplo a seguir:

```
<body bgcolor="#808000">
```

Ao invés de uma cor de segundo plano, poderíamos definir uma figura de segundo plano, conforme o exemplo indicado a seguir:

```
<body background="http://www.abc.com/imagens/texturas/padrao.jpg">
```

Relação das Principais Tags do HTML Utilizadas nos Exemplos do Livro

Neste tópico apresentaremos uma relação das tags mais utilizadas. São tags para pequenos ajustes na página, tais como inserir uma quebra de linha, colocar um texto em negrito, criar uma tabela, etc.

As Tags Para Criação de Títulos

Existe uma série de tags para a criação de títulos. Estas tags possuem tamanhos e formatações variados. A seguir temos o exemplo de uma destas tags:

```
<H3> Texto do título </H3>
```

Na Listagem I.3 temos um exemplo onde são utilizadas as diversas tags de títulos disponíveis.

Listagem I.3 – Inserindo títulos na página.

```
<html>
<head>
    <title>Exemplo de títulos !!</title>
</head>
<body>

    <h1>Esta é uma tag h1</h1>
    <h2>Esta é uma tag h2</h2>
    <h3>Esta é uma tag h3</h3>
    <h4>Esta é uma tag h4</h4>
    <h5>Esta é uma tag h5</h5>
    <h6>Esta é uma tag h6</h6>

</body>
</html>
```

Na Figura I.1 temos o resultado desta página.

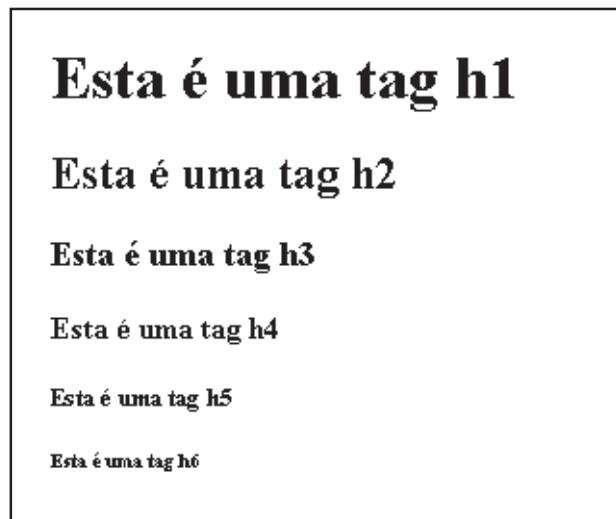


Figura I.1: Diferentes formatos para o título da página.

Tags Para a Criação de Tabelas

Em diversos exemplos deste livro utilizamos tabelas para facilitar o alinhamento dos controles de um formulário. Para entender facilmente as tags para a criação de tabela, basta pensarmos na tabela como uma coleção de linhas e em cada linha como uma coleção de células.

Para criar uma tabela utilizamos a tag:

```
<table>  
</table>
```

Para criarmos uma linha, utilizamos as tags `<hr></hr>`:

```
<table>  
  <tr>  
  </tr>  
  
  <tr>  
  </tr>  
  <tr>  
  </tr>  
</table>
```

Neste caso temos uma tabela com três linhas. O passo final é definir quantas colunas teremos em cada linha. No exemplo a seguir vamos definir duas células por linha. O resultado prático é que teremos uma tabela com três linhas

e duas colunas. Para adicionar células a uma linha, utilizamos as tags <td>/<td>. Entre estas duas tags colocamos o conteúdo da célula. Observe o exemplo da Listagem I.4.

Listagem I.4 – Criando uma tabela simples.

```
<html>

<head>
    <title>Exemplo de criação de tabela.</title>
</head>

<body>
    <table border="1">
        <tr>
            <td>Linha 1, Coluna 1</td>
            <td>Linha 1, Coluna 2</td>
        </tr>

        <tr>
            <td>Linha 2, Coluna 1</td>
            <td>Linha 2, Coluna 2</td>
        </tr>

        <tr>
            <td>Linha 3, Coluna 1</td>
            <td>Linha 3, Coluna 2</td>
        </tr>
    </table>
</body>
</html>
```

Na Figura I.2 temos o resultado desta página.

Observe que utilizamos o atributo border, para definir o tamanho das bordas (linhas de grade) da tabela.

Podemos definir que uma tabela deva ocupar uma porcentagem específica da janela do navegador, tanto na horizontal quanto na vertical:

```
<table border="1" width="90%" height="50%">
```

Linha 1, Coluna 1	Linha 1, Coluna 2
Linha 2, Coluna 1	Linha 2, Coluna 2
Linha 3, Coluna 1	Linha 3, Coluna 2

Figura I.2: Tabela criada com HTML.

Mesmo que o usuário maximize, restaure ou redimensione a janela, o navegador irá manter as proporções especificadas nesta tag.

Para definir uma cor de fundo para a tabela, utilizamos o atributo bgcolor, conforme exemplo a seguir:

```
<table border="1" bgcolor="#c0c0c0">
```

Podemos definir a cor de segundo plano, individualmente, para uma única célula, utilizando o atributo bgcolor para a tag <td> que define a célula:

```
<td bgcolor="#ff0000">Conteúdo da célula</td>
```

Podemos mesclar células utilizando os atributos:

```
colspan="número de colunas a ser mescladas"
```

```
rowspan="número de linhas a ser mescladas"
```

Observe o exemplo da Listagem I.5.

Listagem I.5 – Mesclando linhas e colunas.

```
<html>

  <head>
    <title>Mesclando linhas e colunas!</title>
  </head>

  <body>

    <table border="1">
      <tr>
        <td colspan="2">Mesclando duas colunas</td>
      </tr>
      <tr>
        <td>Coluna 1</td>
        <td>Coluna 2</td>
      </tr>
    </table>
  </body>
</html>
```

```

<td>Linha 2, Coluna 1</td>
<td rowspan="2">Meslando duas linhas</td>
</tr>

<tr>
    <td>Linha 3, Coluna 1</td>
</tr>

</table>
</body>
</html>

```

Na Figura I.3 temos o resultado desta página.

Mesclando duas colunas	
Linha 2, Coluna 1	Meslando duas linhas
Linha 3, Coluna 1	

Figura I.3: Mesclando linhas e colunas.

Inserindo uma Linha Horizontal

Também utilizamos, em diversos exemplos, a tag <HR>. Esta tag insere uma linha horizontal na página. Existem alguns atributos que podem ser utilizados para definir o formato da linha:

```
<hr align="left" size="5" color="#FF0000">
```

Neste exemplo criamos uma linha com alinhamento à esquerda, com espessura de 5 pixels e de cor vermelha. A tag <HR>, automaticamente, insere uma quebra de linha.

Tags Para Formatação Básica do Texto

Existe um conjunto de tags que permite a definição de efeitos como negrito, itálico e sublinhado.

No exemplo da Listagem I.6 utilizamos diversas tags para formatação de texto.

Listagem I.6 – Formatação de texto com o HTML.

```

<html>
<head>

```

```
<title>Tags para formatação de texto</title>
</head>

<body>

<h2>Formatação de texto:</h2>

<p><B>Este texto em negrito</strong></B>

<p><I>Este texto em itálico</I></p>

<p><U>Este texo sublinhado</U></p>

<p><B><I>Este texto em negrito e itálico</B></I></p>

</body>
</html>
```

Na Figura I.4 temos o resultado desta página.

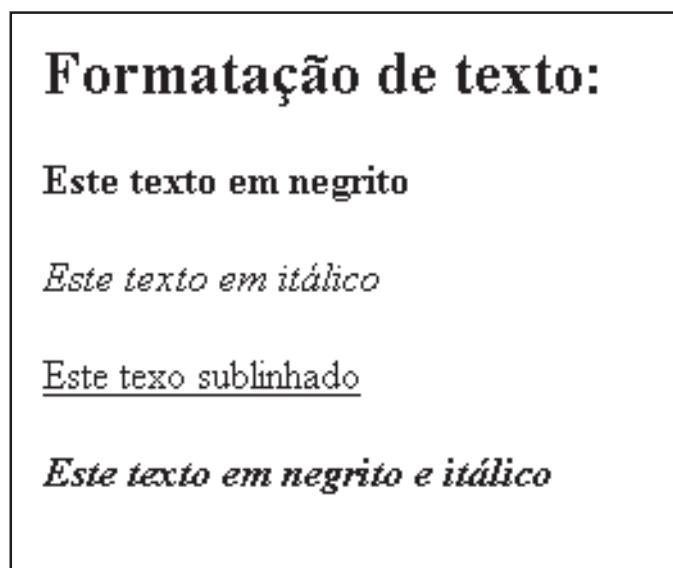


Figura I.4: Formatação básica de texto.

Utilizamos as seguintes tags:

- ◆ : Negrito
- ◆ <I></I>: Itálico
- ◆ <U></U>: Sublinhado

Observe que é possível combinar duas ou mais tags para aplicar múltiplas formatações. A tag <P> é utilizada para a definição de parágrafo. Ao fechamos a tag (</p>) será feita uma quebra automática de linha.

Formatação de Fonte

Utilizamos a tag para definir algumas características da fonte, como por exemplo:

- ◆ Cor da fonte
- ◆ Tipo de fonte
- ◆ Tamanho

No exemplo da Listagem I.7 utilizamos a tag para formatar texto.

Listagem I.7 – Formatação de texto com a tag .

```
<html>

<head>
<title>Tags para formatação de fonte!</title>
</head>

<body>

<h2>Formatação de texto:</h2>

<p>
<font face="Arial" size="6" color="#FF0000">
Fonte arial, cor vermelha e tamanho 6
</font>
</p>

<p>
<font face="Arial" size="4" color="#0000FF">
Fonte arial, cor azul e tamanho 4
</font>
</p>

<p>
<font face="Arial" size="3" color="#00FF00">
```

```
Fonte arial, cor vermelha e tamanho 3
</font>
</p>

<p>
<font face="Arial" size="2" color="#FF8000">
Fonte arial, cor laranja e tamanho 2
</font>
</p>

</body>
</html>
```

Na Figura I.5 temos o resultado desta página.

A Tag <A>

Esta tag é utilizada para criar links em uma página HTML.

No exemplo da Listagem I.8 utilizamos a tag <A> para criar alguns links.

Formatação de texto:

Fonte arial, cor vermelha e tamanho 6

Fonte arial, cor azul e tamanho 4

Fonte arial, cor vermelha e tamanho 3

Fonte arial, cor laranja e tamanho 2

Figura I.5: Formatação básica de fonte.

Listagem I.8 – Criando links com a tag <A>.

```
<html>

<head>
<title>Criação de links!</title>
</head>
```

```
<body>

<h2>Livros do autor Júlio Battisti:</h2>

<ul>
    <li>
        <a href="http://www.juliobattisti.com/livros/windows2000.htm">
            Windows 2000 Server
        </a>
    </li>

    <li>
        <a href="http://www.juliobattisti.com/livros/asp3.htm">
            ASP 3.0
        </a>
    </li>

    <li>
        <a href="http://www.juliobattisti.com/livros/sql2000.htm">
            SQL Server 2000
        </a>
    </li>
</ul>

</body>
</html>
```

Na Figura I.6 temos o resultado desta página.



Figura I.6: Criação de links.

O Conceito de Contas de Usuários no Windows 2000

Quando trabalhamos com uma rede de computadores, segurança é um dos itens de maior importância. O Administrador deve ser capaz de permitir que cada usuário somente tenha acesso aos recursos – sejam eles arquivos, impressoras, páginas ou aplicações Web e serviços –, os quais sejam necessários para a realização do seu trabalho. Por exemplo, um usuário que trabalha no departamento de embalagem não deve ser capaz de acessar informações sobre salários, contidas nos arquivos de um computador do departamento de Recursos Humanos.

NOTA: Também utilizamos as tags `` e `` para a criação de uma lista com bullets.

No Windows 2000 Server, podemos limitar os recursos aos quais cada usuário tem acesso, através do uso de permissões. As permissões de acesso podem ser atribuídas para um usuário individualmente, ou para um Grupo de Usuários. Para que possamos atribuir permissões, cada usuário deve ser cadastrado no sistema. Cadastrar o usuário significa criar uma “Conta de Usuário” para o mesmo. Com uma conta o usuário pode efetuar o logon e receber permissões para acessar os mais variados recursos disponibilizados pelo Windows 2000 Server.

Uma conta pode ser criada em um Controlador de Domínio – situação em que a conta é válida e reconhecida em todo o domínio; ou a conta pode ser criada em um Servidor Membro – situação em que a conta somente é válida e reconhecida no Servidor Membro onde ela foi criada.

Contas criadas em um Controlador de Domínio são chamadas de “Domain User Accounts” (Contas de Usuários do Domínio). Essas contas permitem que o usuário faça o logon em qualquer computador do domínio e receba permissões para acessar recursos em qualquer computador do domínio. Vamos trabalhar e criar contas em um domínio chamado CARUNCHO, com um domínio DNS chamado caruncho.com. Para criar contas em Servidores Membro, o procedimento é bastante semelhante, apenas a quantidade de campos de informação de cada conta é um pouco menor.

NOTA: Para maiores informações sobre Domínios, Controladores de Domínio e Member Servers, consulte o Capítulo 4 do livro: “Série Curso Básico & Rápido Microsoft Windows 2000 Server”, de minha autoria, publicado pela Axcel Books (www.axcel.com.br).

Contas criadas em um Servidor Membro são chamadas de “Local User Accounts” (Contas de Usuários Locais). Essas contas somente permitem que o usuário faça o logon e receba permissões para acessar recursos do computador onde a conta foi criada. Sempre que possível evite criar Contas Locais em servidores que fazem parte de um domínio. Utilizar as contas do Domínio, as quais ficam armazenadas no Active Directory, torna a administração bem mais fácil.

Outro detalhe que você deve observar é a utilização de um padrão para o nome das contas de usuários. Você deve estabelecer um padrão para a criação de nomes, pois não podemos ter dois usuários com o mesmo nome de logon dentro da mesma Unidade Organizacional. Por exemplo se tivermos na mesma Unidade Organizacional, dois José da Silva e os dois resolverem utilizar como logon “jsilva”, estaremos com um problema. Para isso é importante que seja definido um padrão e no caso de nomes iguais deve ser definida uma maneira de diferenciá-los. Por exemplo poderíamos usar como padrão a primeira letra do nome e o último sobrenome. No caso de nomes iguais, acrescentam-se números. No nosso exemplo, o primeiro José da Silva cadastrado ficaria como jsilva, já o segundo a ser cadastrado ficaria como jsilva1. Caso no futuro tivéssemos mais um José da Silva dentro da mesma Unidade Organizacional, este seria o jsilva2 e assim por diante.

Quando formos criar nomes de logon para os usuários, devemos levar em consideração os seguintes fatos:

- ◆ Nomes de Usuários do Domínio devem ser únicos dentro da Unidade Organizacional onde o usuário for cadastrado.
- ◆ Podem ter no máximo 20 caracteres.
- ◆ Os seguintes caracteres não podem ser utilizados: “ / \ : ; [] | = , + * ? < >

Sempre que você for cadastrar um usuário também deve ser cadastrada uma senha para o mesmo; além disso podemos especificar um número mínimo de caracteres aceito para a senha, conforme veremos mais adiante nesta lição. O número máximo de caracteres da senha é 128.

Vamos praticar um pouco. Vamos criar algumas contas de usuários (lembrando que as telas de exemplo mostram a criação de contas em um Controlador de Domínio), e depois vamos alterar algumas propriedades destas contas.

IMPORTANTE: Para as senhas, o Windows 2000 Server distingue letras maiúsculas de minúsculas. Por exemplo a senha “Abc123” é diferente da senha “abc123”.

Exemplo:

Criar as seguintes contas de usuários com as respectivas senhas:

Tabela I.1 Contas de Usuários do Domínio CARUNCHO

Nome da conta	Senha	Nome completo
jsilva	senha123	José da Silva
maria	maria123	Maria do Socorro
paulo	paulo123	Paulo Pereira

Para criar a Conta para o usuário José da Silva:

1. Efetue o logon como Administrador.
2. Dê um clique no botão Iniciar, aponte para Programas e, dentro de Programas, aponte para Ferramentas administrativas.
3. No menu que surge, dê um clique na opção: Usuários e computadores do Active Directory.

Será inicializado o MMC e carregado o Snap-In para Gerenciamento do Diretório, conforme indicado pela Figura I.7

4. Dê um clique no sinal de + ao lado de caruncho.com (provavelmente o nome do seu domínio seja diferente, dê um clique no sinal de + ao lado do nome do seu domínio).

Abaixo de caruncho.com surgem diversas opções.

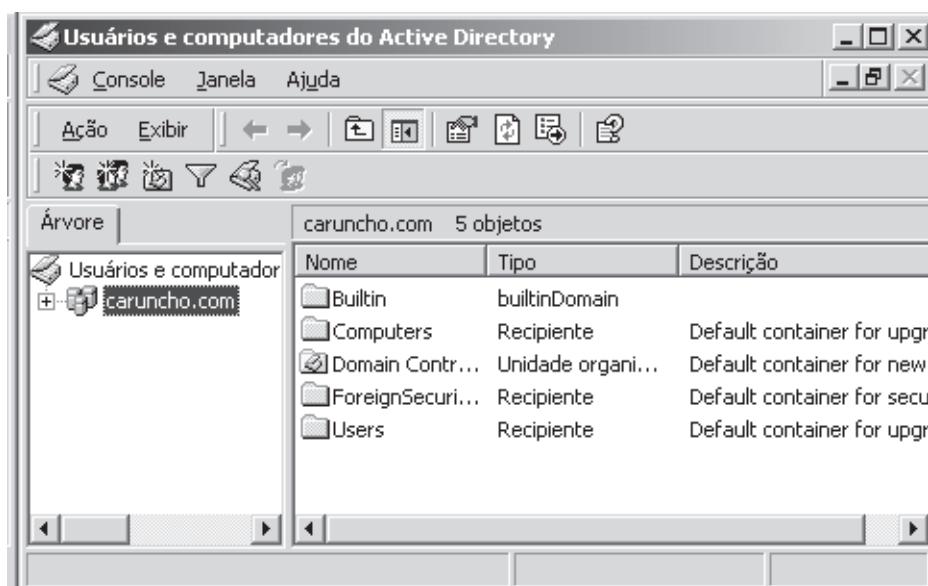


Figura I.7: O Snap-in para Gerenciamento do Diretório.

5. Dê um clique na opção Users (ou Usuários se você estiver utilizando o Windows 2000 Server em Português). No painel da direita é exibida uma listagem com o nome de todos os usuários, conforme indicado na Figura I.8.
6. Dê um clique, com o botão direito do mouse, sobre a opção Users (ou Usuários se for o caso).

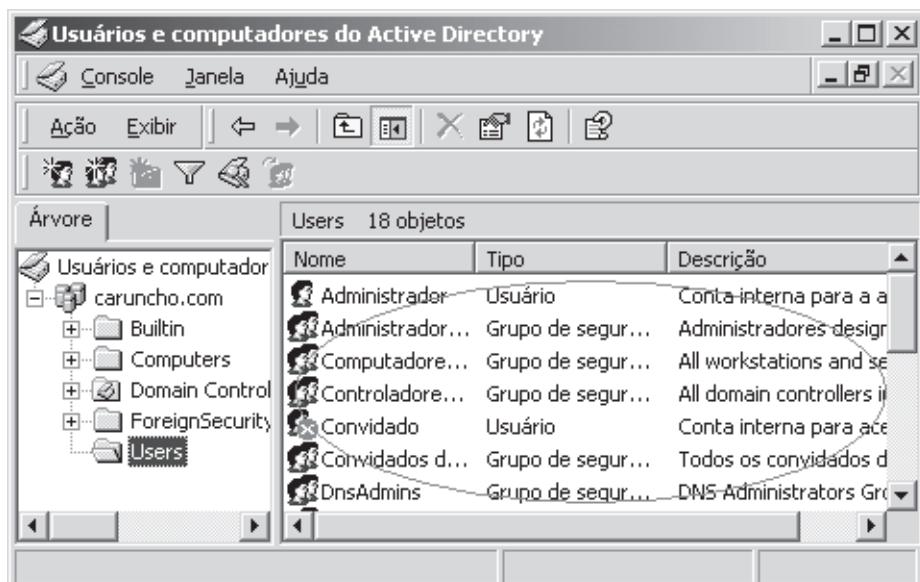


Figura I.8: Listagem com todos os usuários já cadastrados.

7. No menu que surge, aponte para a opção New (Novo) e no menu que surge dê um clique em Usuário. Surge um assistente para ajudá-lo a criar um novo usuário. Digite as informações para criar o usuário jsilva, conforme indicado na Figura I.9.

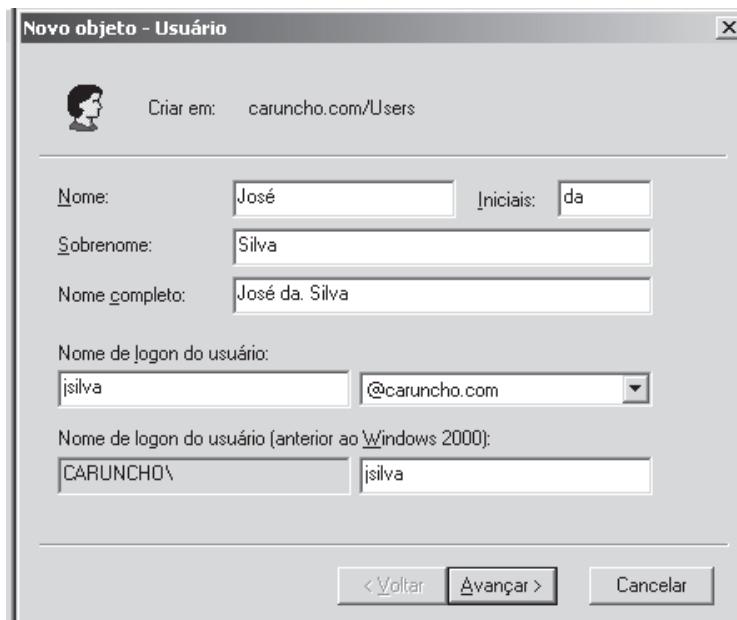
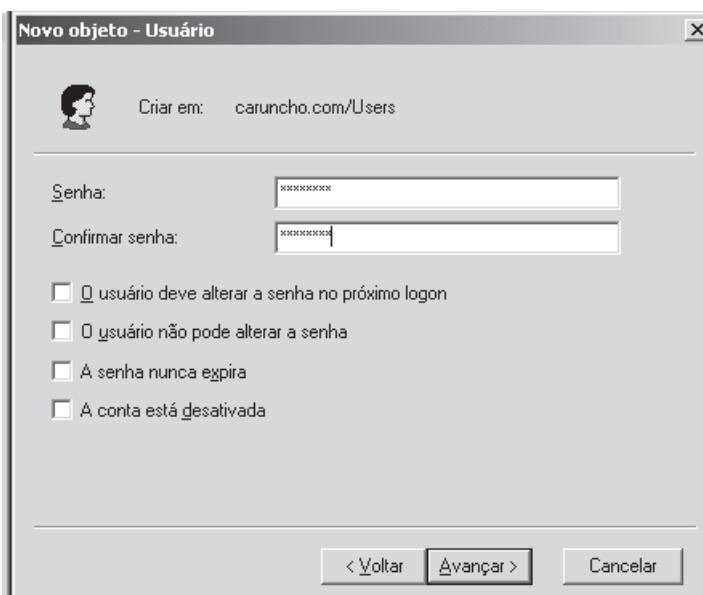


Figura I.9: Criando o usuário jsilva.

8. Dê um clique no botão Avançar para ir para a próxima etapa.
9. Na próxima tela você deve digitar a senha do usuário duas vezes, conforme indicado na Figura I.10. Digite senha123 nos campos Senha e Confirmar senha. Observe que à medida que você digita a senha, o Windows 2000 Server exibe apenas asteriscos (*) nos campos Senha e Confirmar senha.

IMPORTANTE: Nome de logon do usuário é o nome que o usuário utiliza para efetuar o logon em computadores com o Windows 2000 Server. Já Nome de logon do usuário (anterior ao Windows 2000) é o nome que o usuário utiliza para efetuar o logon em computadores com versões mais antigas do Windows, tais como o Windows NT Server 4.0. Por simplicidade estes dois nomes devem ser iguais; observe que à medida que você digitar o primeiro, o segundo será automaticamente preenchido.



NOTA: À medida que você for digitando o Nome, Iniciais e Sobrenome, o Windows 2000 Server vai preenchendo o campo Nome completo. Caso você queira, é possível alterar o Nome completo, sem que isso provoque mudança nos demais campos.

Figura I.10: Cadastrando a senha para o usuário.

Outras opções que podem ser configuradas nesta tela:

- ◆ O usuário deve alterar a senha no próximo logon: Se esta opção estiver marcada, na primeira vez que o usuário fizer o logon, será solicitado que o mesmo altere a sua senha. Esta opção é utilizada para que o usuário possa colocar uma senha que somente ele conhece, pois na primeira vez que o usuário é cadastrado, a senha é digitada pelo Administrador, o qual fica sabendo a senha do usuário. No próximo logon o usuário altera a senha de tal maneira que somente ele saiba qual a senha para a sua conta.
- ◆ O usuário não pode alterar a senha: A senha somente pode ser alterada pelo Administrador. Normalmente utilizada para empregados temporários e para estagiários.
- ◆ A senha nunca expira: Independente das políticas de segurança do domínio, se esta opção estiver marcada, o usuário nunca precisa trocar a sua senha. Caso contrário, de tempos em tempos (conforme configurado nas políticas de segurança do domínio), o usuário deve trocar a senha.
- ◆ A conta está desativada: O Administrador marca esta opção para bloquear a conta de um usuário. Usuários com a conta bloqueada não podem mais efetuar logon e, consequentemente, não podem mais acessar recursos da rede. Esta opção normalmente é utilizada para desativar, temporariamente, a conta de empregados que estão em férias. Quando o empregado retorna ao serviço, o Administrador libera a sua conta, simplesmente desmarcando esta opção.

10. Certifique-se de que as 4 opções acima descritas estão desmarcadas e dê um clique no botão Avançar.
11. Surge uma tela informando que um novo objeto será criado. Lembre-se de que todos os elementos do Active Directory são chamados de objetos. Dê um clique no botão Concluir. Feito isso, o usuário José da. Silva já aparece na listagem de usuários, conforme indicado na Figura I.11.

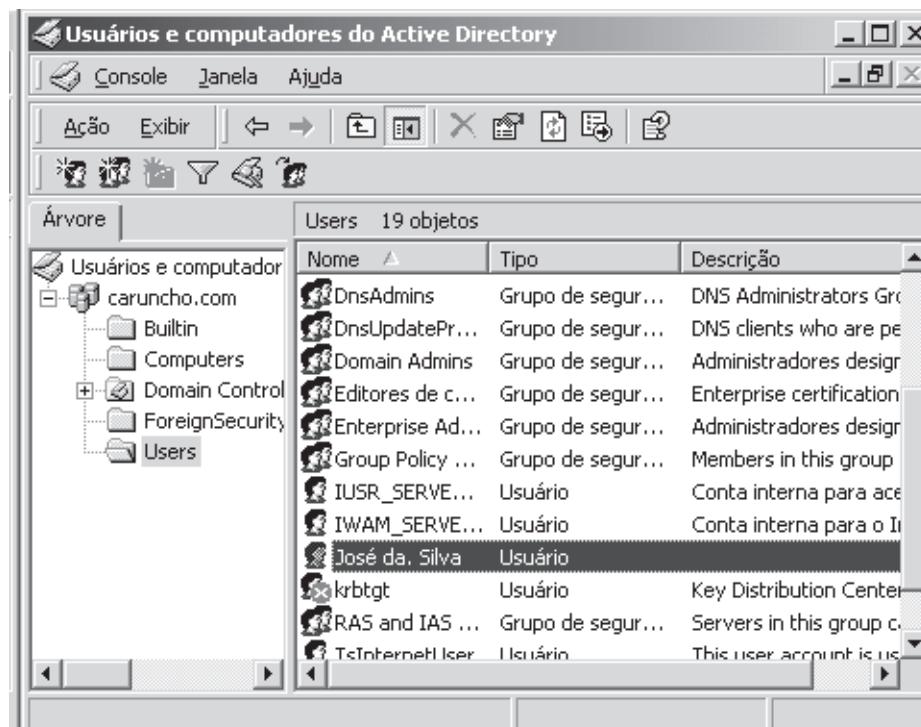


Figura I.11: Usuário José da. Silva já cadastrado.

Feche a janela indicada na Figura I.11.

Agora vamos efetuar o logoff do usuário Administrador e fazer o logon com a conta jsilva, para verificar se a mesma está funcionando corretamente.

Para efetuar o logoff do usuário Administrador e se logar como jsilva:

1. Feche todos os aplicativos que você tiver aberto.
2. Dê um clique no botão Iniciar e depois na opção Desligar.
3. Surge a janela Desligar o Windows.
4. Na lista escolha “Efetuar logoff de Administrador” e dê um clique em OK.
5. Em poucos instantes o logoff é efetuado e o Windows 2000 Server volta a mostrar a tela inicial de logon.
6. Pressione Ctrl+Alt+Del.
7. Na tela que surge, no campo Nome do usuário, digite jsilva.
8. No campo Senha digite senha123.
9. Caso o campo Efetuar logon em: não esteja sendo exibido, dê um clique no botão Opções. O campo Efetuar logon em: mostra o nome do domínio no qual você está se logando.

Dê um clique em OK e, pronto, você terá se logado com a conta de usuário jsilva, recém-criada.

Faça o logoff do usuário jsilva.

Alterando Propriedades Importantes das Contas de Usuários

Vamos aprender a alterar uma conta de usuário, e também veremos algumas propriedades importantes que podem ser alteradas depois que a conta é criada. Veremos como renomear uma conta de usuário e como definir as políticas de segurança relativas às senhas dos usuários, tais como tamanho mínimo da senha, período para expiração das senhas, etc.

Exemplo: Neste primeiro exemplo, vamos ver como alterar a conta do usuário jsilva.

NOTA: Crie as contas para os usuários maria e paulo, com as senhas indicadas na Tabela I.1, e depois faça o logon com cada uma das contas, para testar se as mesmas foram criadas corretamente.

Para alterar a conta do usuário jsilva.

1. Efetue o logon como Administrador.
2. Dê um clique no botão Iniciar, aponte para Programas e dentro de Programas aponte para Ferramentas administrativas.
3. No menu que surge, dê um clique na opção: Usuários e computadores do Active Directory.
4. Será inicializado o MMC e carregado o Snap-In para Gerenciamento do Diretório.
5. Dê um clique no sinal de + ao lado de caruncho.com (provavelmente o nome do seu domínio seja diferente, dê um clique no sinal de + ao lado do nome do seu domínio).

Abaixo de caruncho.com surgem diversas opções.

6. Dê um clique na opção Users (ou Usuários se você estiver utilizando o Windows 2000 em Português). No painel da direita é exibida uma listagem com o nome de todos os usuários cadastrados.
7. Localize o usuário José da Silva e dê um clique duplo sobre o mesmo para abrir as propriedades da conta deste usuário. Surge uma janela, com diversas guias, através das quais podemos configurar uma série de propriedades, conforme indicado na Figura I.12. Ao abrir as propriedades de uma conta de usuário, a guia Geral já vem selecionada por padrão.
8. Digite as informações para os campos Descrição, Escritório, Telefone e Correio eletrônico, conforme indicado na Figura I.12.
9. Dê um clique no botão OK para fechar a janela de propriedades e salvar as alterações.

Você terá voltado à janela Usuários e computadores do Active Directory. Observe que a coluna Descrição para o usuário José da Silva já exibe o valor que você digitou no campo Descrição da guia Geral.

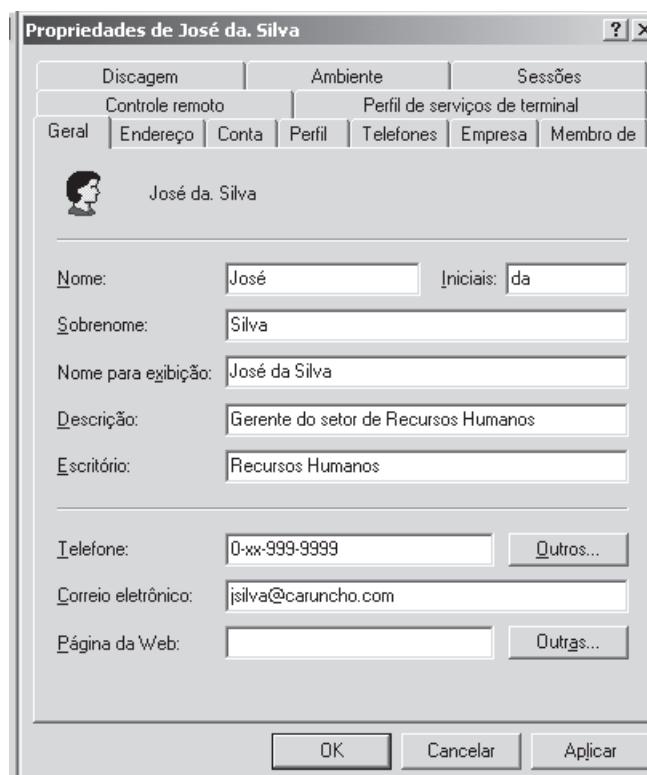


Figura I.12: Alterando as propriedades da conta do usuário José da Silva.

Exemplo: Agora vamos aprender o significado e como alterar algumas propriedades mais avançadas das contas de usuários. Vamos aprender a limitar as horas em que o usuário pode efetuar o logon, bem como limitar os computadores nos quais o usuário pode efetuar o logon.

NOTA: Altere as propriedades das contas maria e paulo, criadas anteriormente. Preencha os campos Descrição, Escritório, Telefone e Correio eletrônico. Invente valores para esses campos, o objetivo é fixar os passos para alterar as propriedades de uma conta de usuário.

Para alterar propriedades importantes do usuário jsilva:

1. Efetue o logon como Administrador.
2. Dê um clique no botão Iniciar, aponte para Programas e dentro de Programas aponte para Ferramentas administrativas.
3. No menu que surge, dê um clique na opção: Usuários e computadores do Active Directory.

Será inicializado o MMC e carregado o Snap-In para “Gerenciamento do Diretório”.

4. Dê um clique no sinal de + ao lado de caruncho.com (provavelmente o nome do seu domínio seja diferente, dê um clique no sinal de + ao lado do nome do seu domínio).
5. Abaixo de caruncho.com surgem diversas opções.
6. Dê um clique na opção Users. No painel da direita é exibida uma listagem com o nome de todos os usuários cadastrados.
7. Localize o usuário José da Silva e dê um clique duplo sobre o mesmo para abrir as propriedades da conta deste usuário. Surge uma janela, com diversas guias, através das quais podemos configurar uma série de propriedades, conforme indicado na Figura I.12. Ao abrir as propriedades de uma conta de usuário, a guia Geral já vem selecionada por padrão.
8. Dê um clique na guia Endereço. Preencha os campos conforme indicado na Figura I.13.

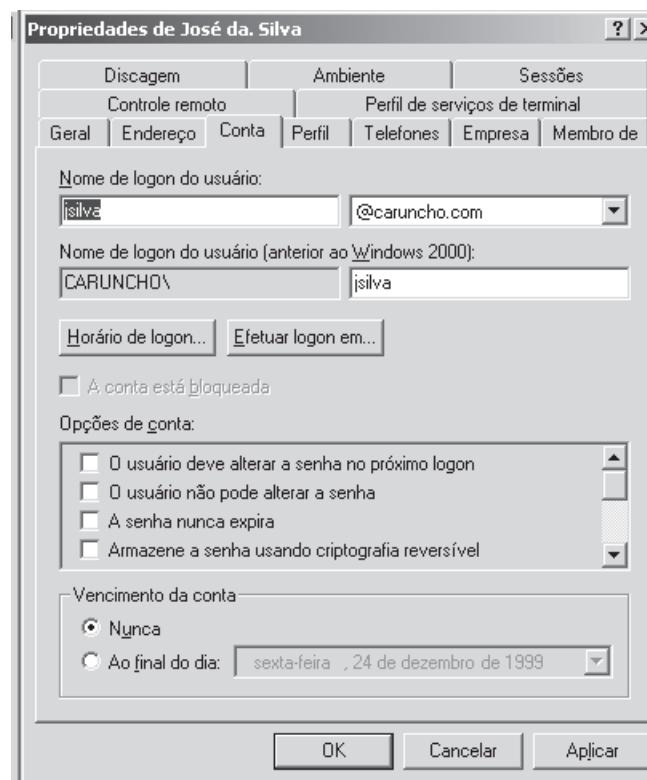


Figura I.13: Preenchendo os campos da guia Endereço.

9. Dê um clique na guia Conta. Nesta guia você pode alterar as informações básicas sobre a conta do usuário, conforme indicado pela Figura I.14. Um administrador pode utilizar essa guia, por exemplo, para desativar a conta de um empregado em férias. Para isso basta marcar a opção “A conta está desativada”.

10. Na guia Conta você também pode estipular em que horário do dia o usuário pode fazer o logon. Por padrão o logon é permitido durante as 24 horas do dia, quando a conta é criada. Podem existir situações em que determinados usuários somente devem ter permissões para se logar durante um certo período.
11. Para definir o período em que o usuário pode se logar, na guia Conta, dê um clique no botão “Horário de logon...”, que será exibida a janela indicada na Figura I.15.
12. Observe que por padrão o logon é permitido durante as 24 horas de todos os dias. Vamos alterar essa configuração de tal forma que o usuário jsilva somente possa se logar das 8 da manhã às 18 horas da tarde de segunda à sexta-feira.
13. Quadrinho azul indica horário permitido e quadradinho branco, horário não permitido.

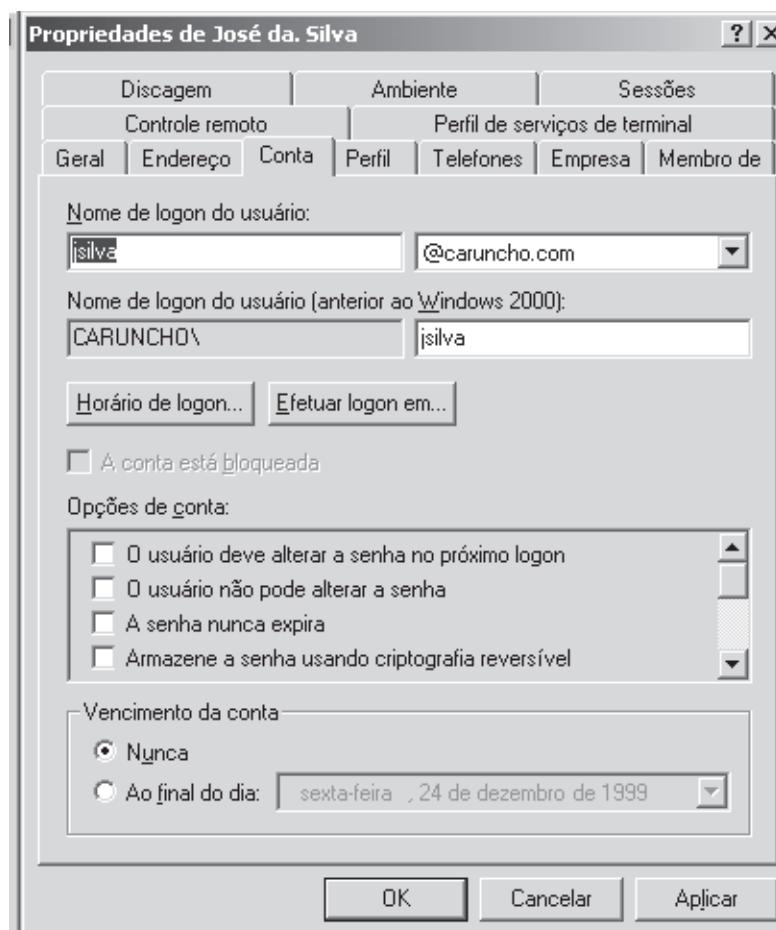


Figura I.14: Alterando as propriedades básicas da conta do usuário.

Para alterar a cor de um quadradinho, basta pressionar o mouse sobre o mesmo, segurar o mouse pressionado e ir arrastando para marcar um ou mais quadradinhos. À medida que você vai arrastando, os quadradinhos vão sendo selecionados. Depois de selecionados basta dar um clique na opção desejada: Logon permitido ou Logon negado, e o Windows 2000 Server altera a cor do quadradinho, conforme a opção escolhida.

14. Utilize a técnica de arrastar, para configurar os horários permitidos conforme indicado na Figura I.16, e depois dê um clique no botão OK. Você estará de volta à guia Conta.

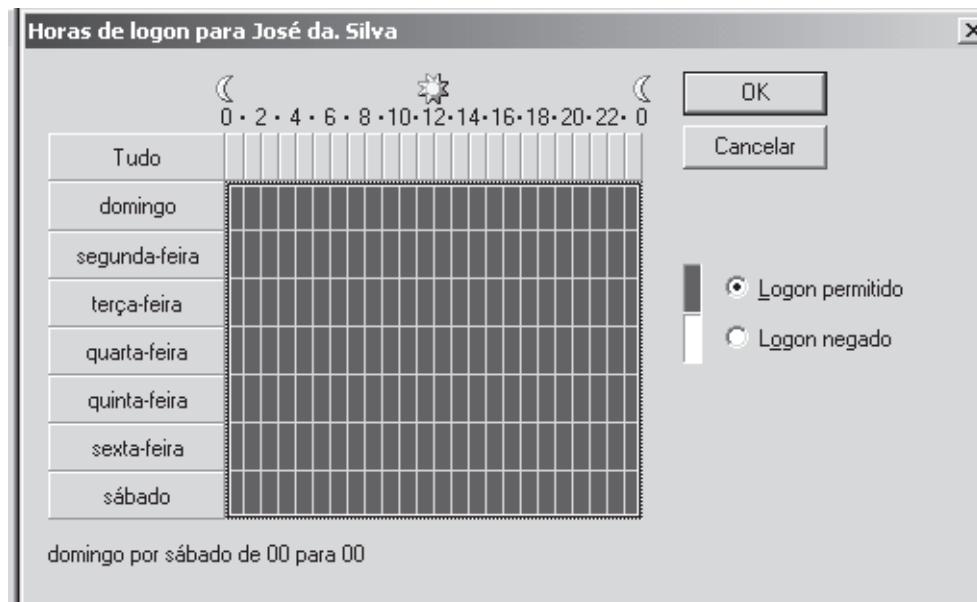


Figura I.15: Configurando os horários de logon permitidos.

NOTA: Para selecionar um dia todo, por exemplo Domingo, basta clicar no botão Domingo. Isso é muito mais fácil do que arrastar o mouse sobre todos os quadradinhos do Domingo. O mesmo é válido para o botão das horas. Se você clicar no botão 8, estará selecionando o quadradinho correspondente às 8 horas de todos os dias.

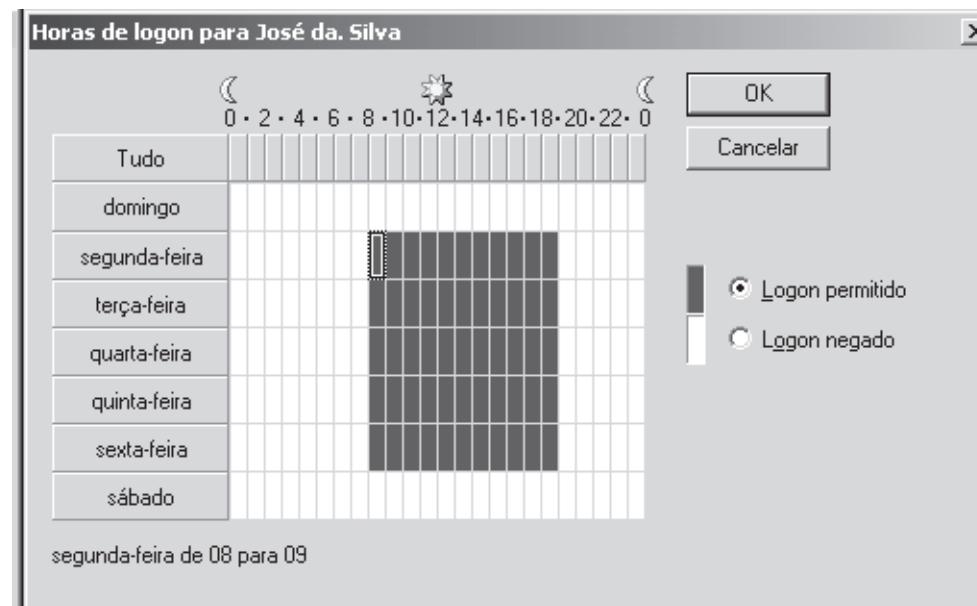


Figura I.16: Logon permitido somente de segunda à sexta-feira, das 8:00 às 18:00 hs.

15. Agora vamos limitar os computadores nos quais o usuário jsilva pode efetuar o logon. Esse procedimento normalmente é adotado com empregados temporários ou estagiários, de tal forma que o Administrador possa controlar em quais computadores esses usuários podem efetuar o logon.

16. Ainda na guia Conta, dê um clique no botão “Efetuar logon em...” .
17. Surge a janela indicada na Figura I.17. Observe que, por padrão, o usuário pode efetuar o logon em qualquer computador do domínio.
18. Para limitar o logon a um número restrito de computadores, dê um clique na opção Os seguintes computadores.
19. No campo Nome do computador digite o nome de um computador onde o usuário pode efetuar o logon e clique no botão Adicionar.
20. Repita a operação do passo anterior para cada computador que você quiser adicionar. Observe na Figura I.18, que estamos dando permissões para o usuário jsilva se logar somente em dois computadores: server1 e server2. Para remover um computador da lista, dê um clique no nome do computador para marcá-lo, e depois dê um clique no botão Remover.
21. Após ter adicionado os computadores desejados, dê um clique no botão OK.
22. Você estará de volta à guia Conta.

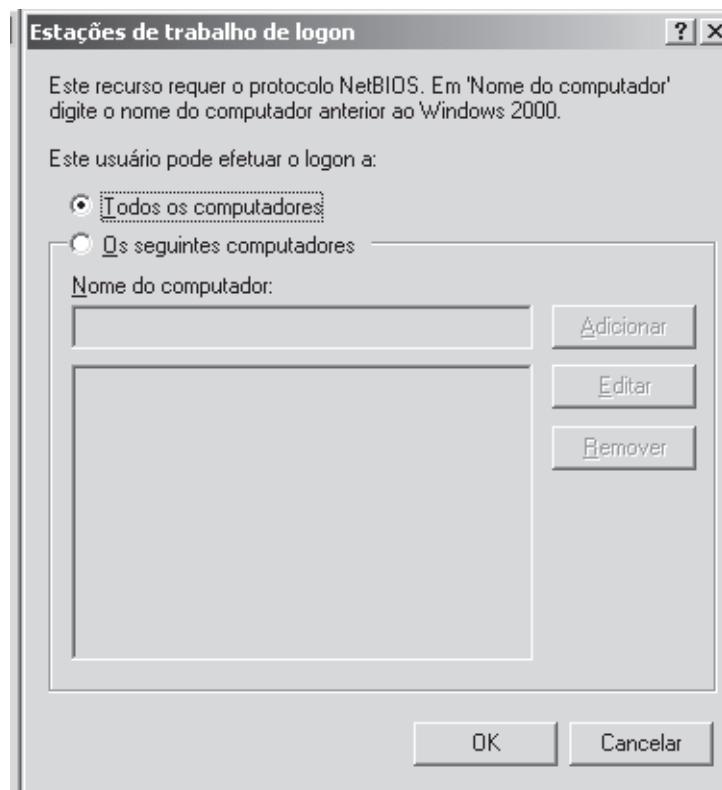


Figura I.17: Logon permitido em todos os computadores do domínio por padrão.

23. Dê um clique no botão OK para fechar as propriedades da conta jsilva.
24. Feche o Console Usuários e computadores do Active Directory.
25. Com as modificações que acabamos de fazer, o usuário jsilva somente poderá se logar no período das 8:00 às 18:00 hs, e somente nos computadores server1 e server2.



Figura I.18: Adicionando computadores à lista.

Exemplo: Vamos aprender a renomear uma conta de usuário.

Para renomear uma conta de usuário:

1. Efetue o logon como Administrador.
2. Dê um clique no botão Iniciar, aponte para Programas e dentro de Programas aponte para Ferramentas administrativas.
3. No menu que surge, dê um clique na opção: Usuários e computadores do Active Directory.
4. Será inicializado o MMC e carregado o Snap-In para Gerenciamento do Diretório.
5. Dê um clique no sinal de + ao lado de caruncho.com (provavelmente o nome do seu domínio seja diferente; dê um clique no sinal de + ao lado do nome do seu domínio).
6. Abaixo de caruncho.com surgem diversas opções.
7. Dê um clique na opção Users (ou Usuários se for o caso). No painel da direita é exibida uma listagem com o nome de todos os usuários cadastrados.
8. Localize o usuário José da Silva e dê um clique duplo sobre o mesmo para abrir as propriedades da conta deste usuário.
9. Dê um clique na guia Conta. Altere os campos “Nome de logon do usuário” e “Nome de logon do usuário” (anterior ao Windows 2000), de jsilva para jsilva2, conforme indicado pela Figura I.19.

NOTA: Altere as propriedades das contas maria e paulo. Invente informações para os endereços das mesmas. Configure as contas para que as mesmas somente possam efetuar o logon das 7:00 às 12:00, de terça à sexta-feira. Limite o logon destas contas apenas ao computador com o nome de server1.

IMPORTANTE: Quando falamos em renomear um usuário, significa renomear o “Nome de logon do usuário”. No nosso exemplo vamos alterar o nome de logon de jsilva para jsilva2.

10. Dê um clique em OK para fechar a janela com as propriedades do usuário e depois feche o console Usuários e computadores do Active Directory.
11. Efetue o logoff do usuário Administrador.

Faça o logon como jsilva2 e para a senha digite senha123, para testar que o usuário jsilva foi renomeado com sucesso para jsilva2.

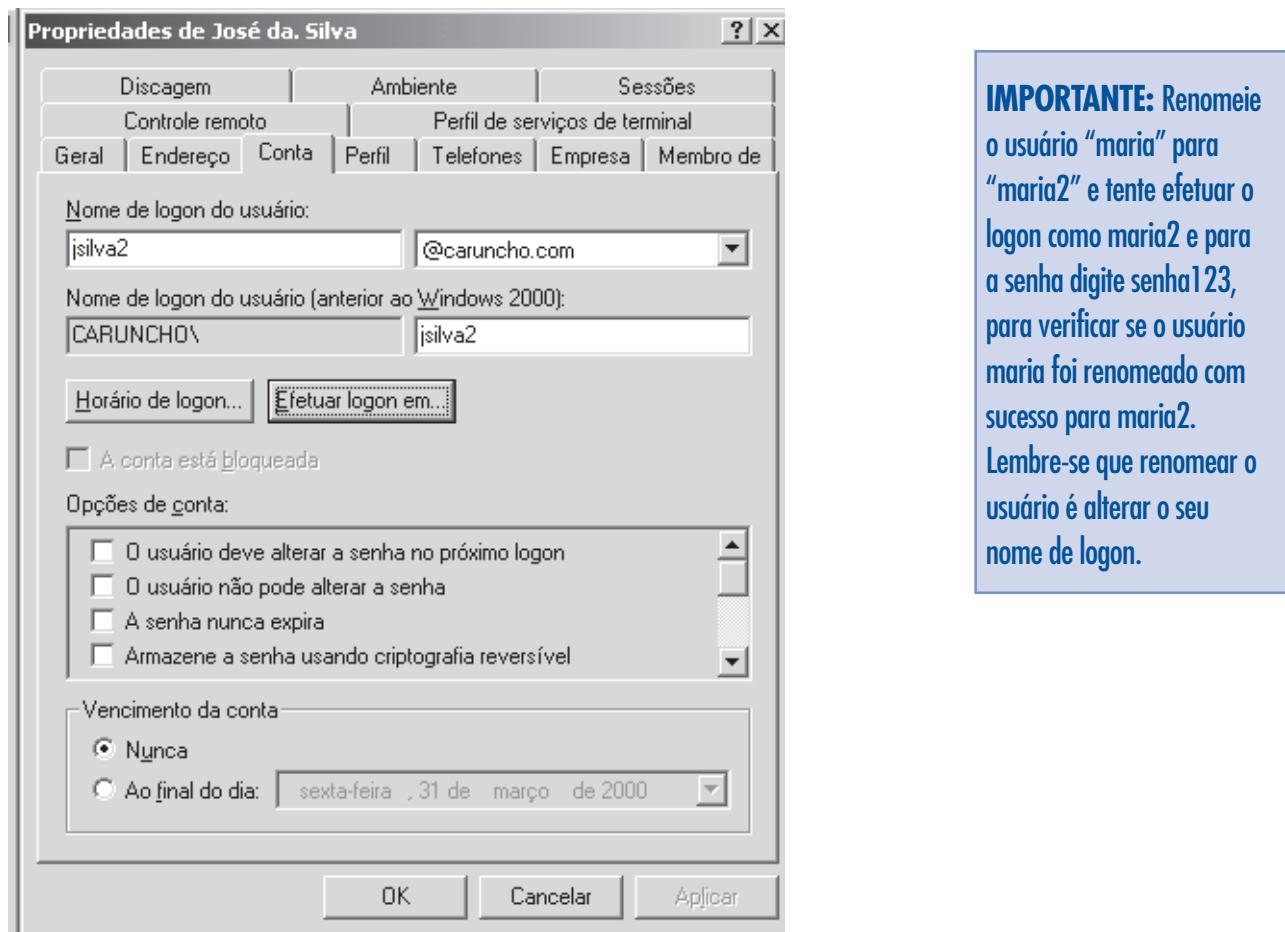


Figura I.19: Alterando o nome de logon de jsilva para jsilva2.

Definindo as Políticas de Senhas

O Windows 2000 Server permite que sejam definidos alguns parâmetros para as senhas a serem utilizadas pelos usuários. Por exemplo, você pode definir que as senhas devem ter um tamanho mínimo de 8 caracteres, ou que as mesmas devem ser trocadas de 40 em 40 dias e que não pode ser utilizada uma senha igual às três anteriores.

Estes são apenas alguns exemplos do que chamamos de Account Policies, que nada mais são do que algumas regras que as senhas devem obedecer.

Podemos definir as “Account Policies” através do console “Diretivas de segurança de domínio”, o qual é acessível através do menu Ferramentas administrativas do menu Programas.

Exemplo: Vamos definir alguns parâmetros, utilizando o console Diretivas de segurança do domínio.

Para definir algumas regras para as senhas:

1. Efetue o logon como Administrador.
2. Dê um clique no botão Iniciar, aponte para Programas e dentro de Programas aponte para Ferramentas administrativas.
3. No menu que surge, dê um clique na opção: Diretivas de segurança do domínio.
4. Será inicializado o MMC e carregado o Snap-In que permite que sejam configurados vários aspectos de segurança para o domínio.
5. Dê um clique no sinal de + ao lado da opção Windows Settings (Configurações do Windows), para abri-la.
6. Nas opções que surgem abaixo de Windows Settings, dê um clique no sinal de + ao lado da opção Security Settings (Configurações de Segurança); serão exibidas diversas opções, conforme mostrado na Figura I.20.

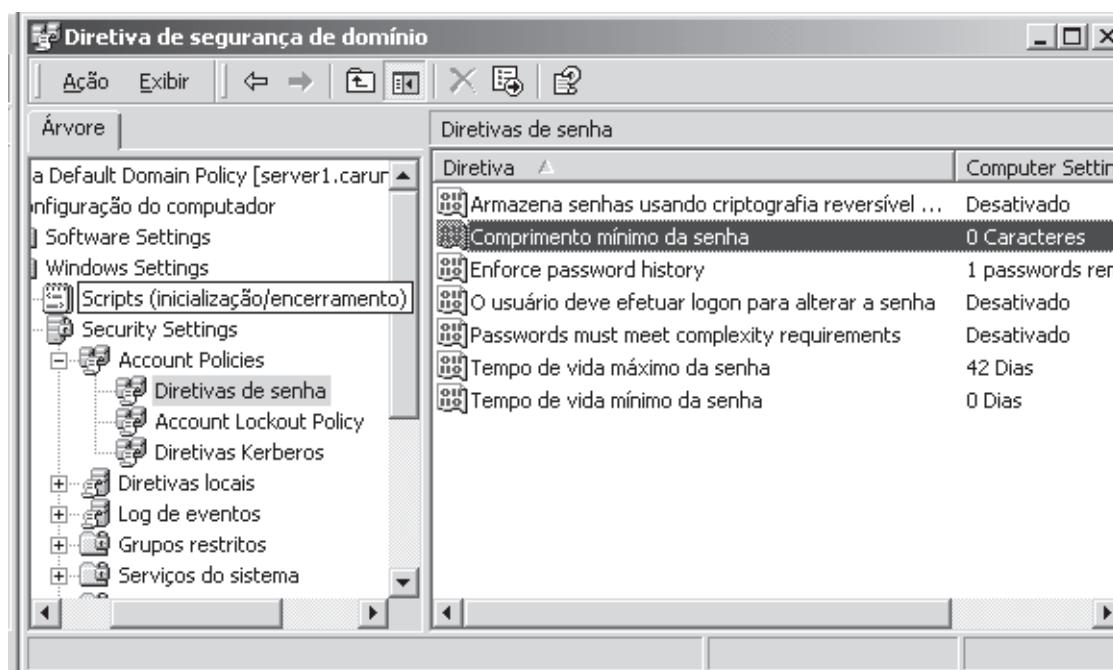


Figura I.20: Configurações de segurança para o domínio.

7. Dê um clique no sinal de + ao lado da opção Accounts Policies para abri-la.
8. Nas opções que aparecem, abaixo de Accounts Policies, dê um clique na opção Diretivas de senha. No painel da direita do MMC são exibidas as diversas diretivas que podem ser alteradas.
9. Localize a opção Comprimento mínimo de senha e dê um clique duplo sobre a mesma. Será aberta a janela Configuração da diretiva de segurança. Observe que por padrão o comprimento mínimo é definido em zero, o que permite que o usuário deixe a senha em branco.

10. Permitir que o usuário deixe a senha em branco não é uma boa política de segurança. Clicando na setinha para cima, altere o valor de zero para 8 caracteres, conforme indicado na Figura I.21.

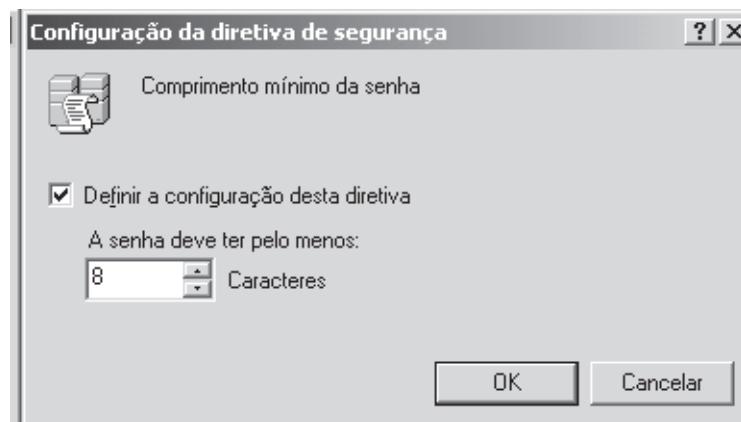


Figura I.21: Exigindo que as senhas tenham pelo menos 8 caracteres.

11. Dê um clique no botão OK para fechar a janela Configuração da diretiva de segurança. Você estará de volta ao console Diretiva de segurança do domínio.
 12. Localize a opção Enforce password history (Forçar histórico de senhas), e dê um clique duplo sobre a mesma. Será aberta a janela indicada na Figura I.22.

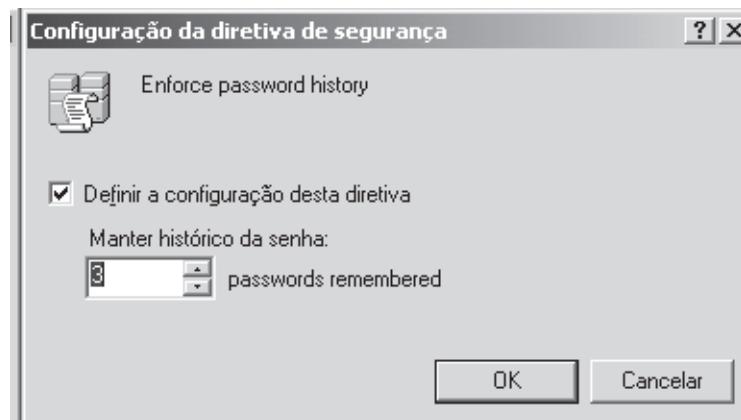


Figura I.22: Alterando o histórico de senhas.

13. Nesta janela você define o histórico de senhas. Altere o campo Manter histórico de senha: de 1 para 3 e dê um clique no botão OK para fechar a janela. Isso significa que, quando o usuário for alterar a senha, o mesmo não pode utilizar uma senha igual às últimas três utilizadas.
 14. Você estará de volta ao console Diretiva de segurança do domínio.
 15. Localize a opção Tempo de vida máximo da senha, e dê um clique duplo sobre a mesma. Será aberta a janela indicada na Figura I.23.

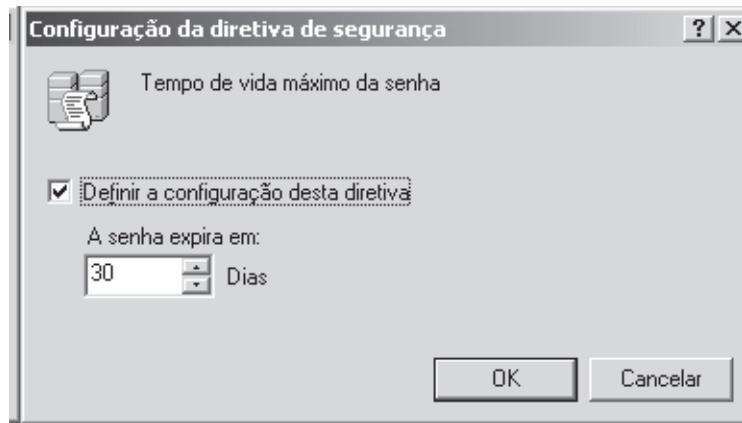


Figura I.23: Alterando o tempo máximo de vida da senha.

16. Nesta janela você define durante quantos dias uma senha é válida. Depois de passado este prazo, ao efetuar o logon o usuário será avisado de que a senha expirou e que a mesma deve ser alterada. Observe que por padrão o tempo de vida da senha é 42 dias.
17. Altere o tempo máximo de vida da senha para 30 dias e dê um clique no botão OK.
18. Você estará de volta ao console Diretiva de segurança do domínio.
19. Observe que os novos valores para as diretrizes Comprimento mínimo de senha, Enforce password history e Tempo de vida máximo de senha já apresentam os novos valores configurados.
20. Feche o console Diretiva de segurança do domínio.

IMPORTANTE: Com as modificações que fizemos nesta lição, as senhas devem ser alteradas de 30 em 30 dias (diretriz Tempo de vida máximo de senha), devem ter no mínimo 8 caracteres (letras, números, e qualquer um dos caracteres especiais permitidos) – diretiva Comprimento mínimo de senha –, e ao alterar a senha, o usuário não poderá repetir uma das três últimas senhas utilizadas (diretriz Enforce password history).

Grupos de Usuários e Tipos de Grupos Existentes no Windows 2000 Server

Neste tópico vamos ver um pouco de teoria sobre grupos de usuários. Em seguida vamos praticar criando alguns grupos e adicionando alguns membros aos grupos criados.

Um grupo de usuários é uma coleção de contas de usuários. Por exemplo, podemos criar um grupo chamado Contabilidade, do qual farão parte todos os usuários do departamento de Contabilidade.

A principal função dos grupos de usuários é facilitar a administração e a atribuição de permissões para acesso a recursos, tais como: pastas compartilhadas, impressoras remotas, serviços diversos, páginas e aplicações Web etc.

Ao invés de darmos permissões individualmente, para cada um dos usuários que necessitam acessar um determinado recurso, podemos criar um grupo e atribuir permissões para o grupo. Para que um usuário tenha permissão ao recurso, basta incluir o usuário no grupo, pois todos os usuários de um determinado grupo herdam as permissões do grupo.

NOTA: Altere as diretrizes de segurança, de tal maneira que o tamanho mínimo permitido para as senhas seja 10, que não seja permitida utilizar uma senha igual às 4 últimas e que o tempo máximo de vida seja de 45 dias.

Quando um usuário troca de seção, por exemplo, basta trocar o usuário de grupo. Vamos supor que o usuário jsilva trabalhe na seção de contabilidade e pertença ao grupo Contabilidade. Ao ser transferido para a seção de marketing, basta retirarmos o usuário do grupo Contabilidade e adicioná-lo ao grupo Marketing. Com isso o jsilva deixa de ter as permissões atribuídas ao grupo Contabilidade e passa a ter as mesmas permissões que tem o grupo Marketing. Veja o quanto a utilização de grupos pode facilitar a atribuição de permissões.

Podemos inclusive ter situações mais específicas. Vamos supor que exista um sistema chamado SEAT.NET, para o qual somente um número restrito de usuários deva ter acesso, sendo que são usuários de diferentes seções. A maneira mais simples de solucionar isso é criar um grupo chamado Seat.Net e dar permissões de acesso para esse grupo. Assim, cada usuário que precisar acessar o sistema SEAT.NET deve ser incluído no grupo Seat.Net. Quando o usuário não deve mais ter acesso ao sistema SEAT.NET, basta removê-lo do grupo Seat.Net.

Na Figura I.24 vemos uma ilustração para o conceito de Grupo de usuários. O Grupo Contabilidade possui direito para um recurso compartilhado, o qual pode ser acessado através da rede. Todos os usuários que pertencem ao grupo contabilidade também possuem permissão para o recurso compartilhado, uma vez que os usuários de um grupo herdam as permissões do grupo.

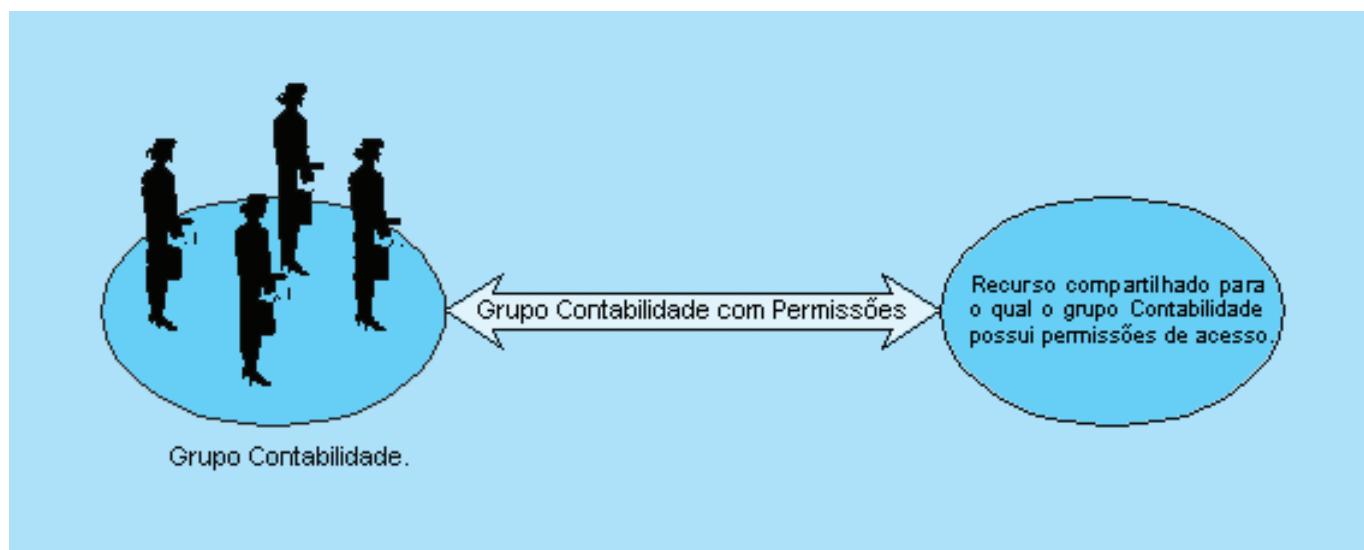


Figura I.24: O usuário herda as permissões do grupo.

Quando estiver trabalhando com grupos de usuários, considere o seguinte:

- ◆ Grupos são uma coleção de contas de usuários.
- ◆ Os membros de um grupo herdam as permissões atribuídas ao grupo.
- ◆ Os usuários podem ser membros de vários grupos.
- ◆ Grupos podem ser membros de outros grupos.

Agora vamos dar uma olhada nos tipos de grupos existentes no Windows 2000 Server.

Podemos ter dois tipos de grupos no Windows 2000 Server:

- ◆ **Grupos de segurança (Security Groups):** Normalmente utilizado para atribuir permissões a recursos da rede. O Windows 2000 Server somente utiliza Grupos de segurança. Um grupo de segurança também pode ser utilizado como um grupo de distribuição, embora essa não seja uma situação muito comum. Esses grupos, assim como as contas de usuários, são armazenados no Banco de dados do diretório.
- ◆ **Grupos de distribuição (Distribution Groups):** São utilizados para funções não relacionadas com segurança. Uma das utilizações típicas para um Grupo de distribuição é o envio de mensagens de e-mail para um grupo de usuários de uma só vez. Somente programas que foram programados para trabalhar com o Active Directory poderão utilizar Grupos de distribuição. Provavelmente as novas versões dos principais sistemas de correio eletrônico estarão habilitadas para trabalhar com o Active Directory. O Exchange Server da Microsoft é integrado com o Active Directory. Não podemos utilizar grupos de distribuição para funções relacionadas com segurança.

Escopo de grupos de usuários:

Quando criamos um grupo de usuários, devemos selecionar um tipo e um escopo. O Escopo permite que o grupo seja utilizado de diferentes maneiras para a atribuição de permissões. O escopo de um grupo determina em que partes da rede poderemos usar o grupo para atribuir permissões para o grupo. Existem três escopos para grupos de usuários, conforme descrito a seguir:

Grupos globais (Global group):

- ◆ Somente podem conter membros do domínio no qual o grupo é criado.
- ◆ Podem receber permissões para recursos localizados em qualquer domínio.

Grupos locais do domínio (Domain local group):

- ◆ Podem conter membros de qualquer domínio.
- ◆ Somente podem receber permissões para o domínio no qual o grupo é criado.

Grupos universais (Universal group):

- ◆ Podem conter membros de qualquer domínio.
- ◆ Podem receber permissões para recursos localizados em qualquer domínio.

O Escopo de um grupo também determina quem pode ser membro do grupo. Tanto usuários como outros grupos podem ser membros de um determinado grupo. Considere as seguintes regras para membros de grupos:

Grupo global:

- ◆ Pode conter: Contas de usuários e grupos globais do mesmo domínio.
- ◆ Pode ser membro de: Grupos universais e grupos locais do domínio em qualquer domínio. Grupos globais no mesmo domínio.

Grupo local do domínio:

- ◆ Pode conter: Contas de usuários, grupos universais e grupos globais de qualquer domínio. Grupos locais do domínio do mesmo domínio.
- ◆ Pode ser membro de: Grupos locais do domínio do mesmo domínio.

Grupo universal:

- ◆ Pode conter: Contas de usuários, grupos universais, e grupos globais de qualquer domínio.
- ◆ Pode ser membro de: Grupos locais do domínio ou grupos universais de qualquer domínio.

Criando Grupos de Usuários e Adicionando Usuários aos Grupos

Neste tópico vamos praticar um pouco. Primeiro vamos criar um grupo, e depois vamos adicionar alguns usuários ao grupo recém-criado.

Exemplo: Para criar um grupo Global chamado Grupo1, faça o seguinte:

1. Faça o logon com Administrador.
2. Dê um clique no botão Iniciar, aponte para Programas e aponte para Ferramentas administrativas.
3. No menu de opções que surge, dê um clique na opção Usuários e computadores do Active Directory.
4. Surge o console para criação de contas de usuários e grupos.
5. No painel da esquerda localize caruncho.com (ou o nome do domínio que você estiver utilizando, caso esteja utilizando um nome diferente), e dê um clique com o botão direito do mouse sobre o nome do domínio.
6. No menu de opções que surge, aponte para Novo e, dentro do menu Novo, dê um clique em Grupo. Surge a janela Novo objeto – Grupo. Preencha as informações conforme indicado na Figura I.25.

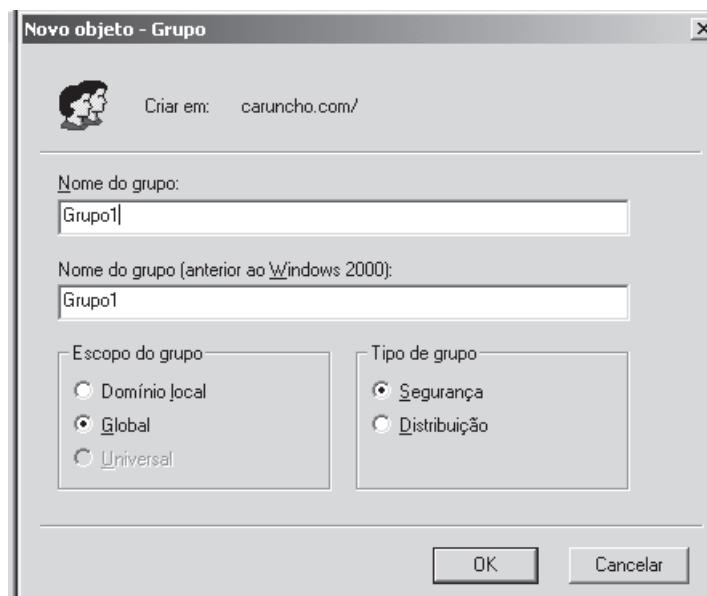


Figura I.25: Criando um grupo de segurança com escopo global.

7. Dê um clique no botão OK para criar o grupo.
8. Você estará de volta ao console Usuários e computadores do Active Directory.
9. Observe no painel da esquerda que já aparece o grupo chamado Grupo1. Feche essa janela.

Para alterar o Grupo1 e adicionar os usuários jsilva2, maria e paulo, a este grupo, faça o seguinte:

1. Faça o logon com Administrador.
2. Dê um clique no botão Iniciar, aponte para Programas e aponte para Ferramentas administrativas.
3. No menu de opções que surge, dê um clique na opção Usuários e computadores do Active Directory.
4. Surge o console para criação de contas de usuários e grupos.
5. No painel da direita, localize Grupo1 e dê um clique duplo para abrir as propriedades do mesmo. Surge a janela indicada na Figura I.26, onde a guia Geral vem selecionada por padrão. Preencha os campos Descrição e Comentários, conforme indicado na Figura I.26.
6. Dê um clique na guia Membros. Vamos utilizar esta guia para adicionar os usuários jsilva, maria e paulo como membros de Grupo1. A Figura I.27 mostra a guia Membros, sem nenhum membro adicionado.
7. Para adicionar membros ao grupo, dê um clique no botão Adicionar.



Figura I.26: Alterando as propriedades do Grupo1.



Figura I.27: Ainda não adicionamos nenhum membro ao grupo Grupo1.

8. Surge a janela Selecione Usuários, Contatos ou Computadores, conforme indicado na Figura I.28. Nesta janela é exibida uma listagem com todos os usuários cadastrados no domínio caruncho.com (ou o nome de domínio que você estiver utilizando).

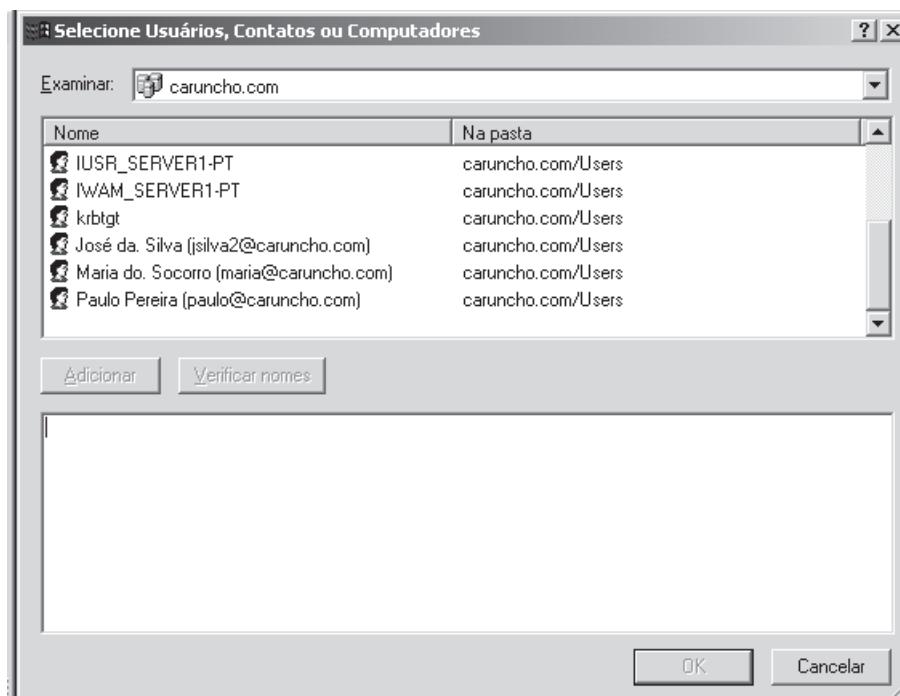


Figura I.28: Lista de Usuários, Contatos e Computadores do domínio caruncho.com.

9. Para adicionar o usuário jsilva (José da Silva) localize-o na listagem, dê um clique sobre ele para marcá-lo e depois dê um clique no botão Adicionar. Você também pode dar um clique duplo sobre o nome do usuário, que o mesmo será adicionado.
10. Repita a operação do passo anterior, para os usuários Maria do Socorro e Paulo Pereira.
11. Dê um clique no botão OK para fechar essa janela e voltar à guia Membros.
12. Sua janela deve estar conforme indicado na Figura I.29, a qual indica que os usuários José da Silva (jsilva2), Maria do Socorro (maria) e Paulo Pereira (paulo) foram adicionados como membros do grupo Grupo1.
13. Dê um clique em OK para fechar a janela de propriedades do grupo Grupo1.
14. Feche o console Usuários e Computadores do Active Directory.

A partir deste momento, qualquer permissão que for atribuída ao grupo Grupo1, será herdada por todos os seus membros, no nosso exemplo, pelos usuários jsilva2, maria e paulo.

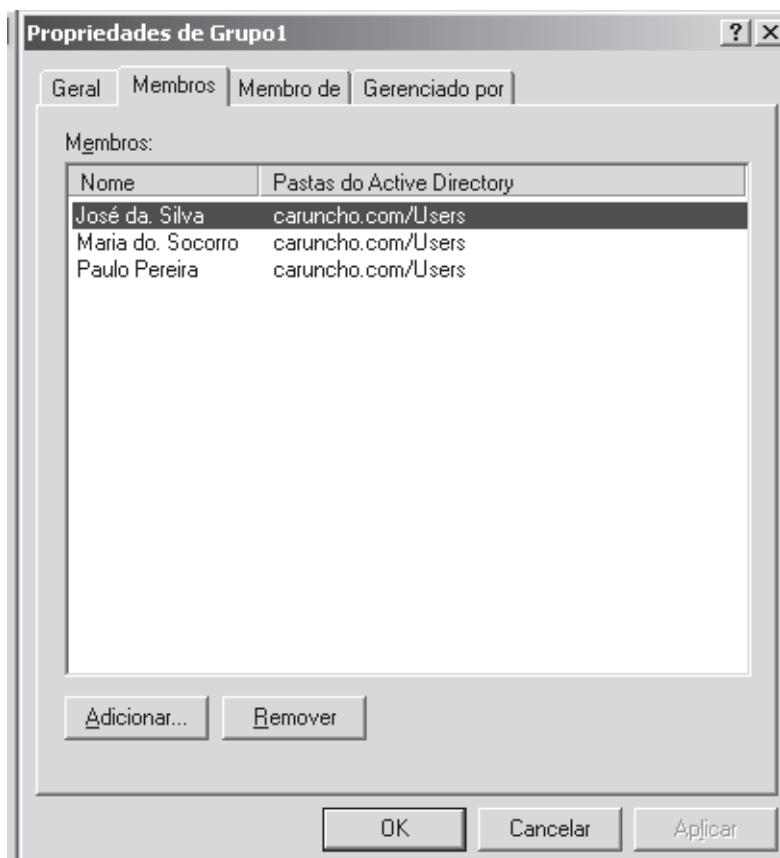


Figura I.29: Três usuários adicionados como membros do grupo Grupo1.

Outra conta criada quando da instalação do Windows 2000 Server é a conta de usuário Convidado (Guest). Esta conta normalmente é utilizada para acesso de usuários que não possuem uma conta cadastrada no domínio. Por padrão esta conta está desabilitada. O Administrador pode habilitar a conta Convidado. Porém isso deve ser feito com cuidado.

NOTA: Crie um grupo de escopo global, do tipo segurança, chamado de Contabilidade. Adicione os usuários maria e paulo ao grupo Contabilidade. Como descrição coloque: Grupo para receber permissões – seção de contabilidade.

IMPORTANTE: Existem algumas contas de usuário que são criadas no momento em que o Windows 2000 Server é instalado. Essas contas são conhecidas como "Built-in Accounts". A mais importante delas é a conta Administrador. Essa conta tem poderes totais sobre o domínio, não possuindo nenhuma restrição de segurança. Muito cuidado com quem vai usar essa conta em um ambiente de produção. Precisa ser uma pessoa qualificada e que saiba o que está fazendo. A conta Administrador pode ser renomeada, porém não pode ser excluída nem bloqueada.

Sempre que um usuário precise acessar algum recurso, o ideal é cadastrar o usuário e incluir o mesmo no grupo (ou grupos), que têm permissão para acessar o recurso desejado.

Existem também alguns grupos criados quando da instalação do Windows 2000 Server, são os chamados “Built-in Groups”. O mais importante de todos é o grupo Administradores. Todo membro deste grupo tem plenos poderes no domínio. No console Usuários e Computadores do Active Directory (Iniciar – Programas – Ferramentas administrativas – Usuários e computadores do Active Directory), existe uma opção chamada Built-in. Ao clicar nessa opção será exibida uma listagem com diversos grupos criados durante a instalação do Windows 2000 Server. Observe, na coluna tipo, que o tipo destes grupos é Grupo de segurança – local interno e na coluna descrição podemos ver um resumo das permissões de cada um dos grupos.

Para maiores informações sobre o Windows 2000 você pode consultar um dos seguintes endereços:

- ◆ www.microsoft.com/technet
- ◆ www.microsoft.com/windows2000
- ◆ www.labmice.net
- ◆ www.2000tutor.com

Introdução

Neste Anexo aprenderemos sobre os conceitos básicos de bancos de dados relacionais.

Durante as décadas de 70 e 80 as aplicações eram baseadas em computadores de grande porte, conhecidos como Mainframes. Nesta época as aplicações normalmente eram desenvolvidas em linguagens como Cobol, PL1, Algol ou Natural Adabas, que ficavam residentes nos Mainframes. Os dados também ficavam residentes nos Mainframes. A lógica para acesso aos dados estava embutida dentro da própria aplicação. Com isso, se mudasse a estrutura de armazenamento das informações, a aplicação teria que ser reescrita. Para acessar as aplicações, o usuário utilizava os chamados “Terminal burros” (também conhecidos como “Terminal verdes”, devido à cor das letras normalmente ser verde em um fundo preto). Estes terminais eram, simplesmente, uma extensão do Mainframe, equipados com teclado e vídeo. Nenhum processamento era realizado no próprio terminal. Todo e qualquer comando devia ser enviado para processamento no Mainframe, e os resultados enviados de volta para o terminal.

Observe que utilizei os verbos sempre no passado. Você deve estar pensando: “Este cara está maluco, pois ainda hoje muitas aplicações rodam, e bem, no bom e velho Mainframe.” E eu serei obrigado a concordar com o amigo leitor. Muitas e importantes são as aplicações que ainda rodam em Mainframes e duvido que em um futuro próximo, digamos cinco anos, todas estas aplicações sejam migradas para outras plataformas e modelos de desenvolvimento. Mas também precisamos admitir que é muito pequeno, para não dizer mínimo, o desenvolvimento de novas aplicações para o modelo Terminal – Mainframe.

Com o deslocamento do modelo de desenvolvimento do Mainframe; primeiro para o modelo Cliente Servidor clássico de duas camadas, depois a evolução para o modelo de desenvolvimento Web com 3 ou mais camadas, também sofreram modificações os Bancos de dados que dão suporte ao desenvolvimento destas aplicações. O Mainframe utiliza formatos de Bancos de dados proprietários, criados pelos fabricantes dos equipamentos, na maioria dos casos a IBM.

Com a expansão cada vez maior das Redes locais de computadores e a utilização do modelo Cliente Servidor, a utilização dos chamados Bancos de dados Relacionais cresceu bastante. Hoje a grande maioria das novas aplicações é baseada em Bancos de dados Relacionais.

Neste anexo falaremos sobre os princípios básicos dos Bancos de dados Relacionais, que é o modelo de banco de dados utilizado pelo Microsoft SQL

ANEXO

2

O Modelo de Dados Relacional

Server 2000, pelo Microsoft Access, pelo ORACLE, pelo DB2 da IBM e por aí vai, conforme indicado pela listagem a seguir:

- ◆ Microsoft Access – para aplicações de menor porte
- ◆ Oracle
- ◆ DB2 da IBM
- ◆ Sybase
- ◆ MySQL para Linux
- ◆ Paradox
- ◆ Dbase
- ◆ Ingress

O modelo relacional também possui suas limitações, porém é bastante adequado para a grande maioria das aplicações comerciais atualmente em uso ou sendo desenvolvidas.

Algumas aplicações especiais que necessitam utilizar dados mais complexos, como por exemplo arquivos multimídia, dados espaciais e séries temporais, podem se beneficiar mais das características dos chamados Bancos de dados orientados a Objetos. Este tipo de Banco de dados ainda não é amplamente utilizado, a não ser em situações específicas.

Um exemplo de Banco de dados orientado a Objetos é o Jasmine da CA.

Devido à grande aceitação e utilização dos Bancos de dados Relacionais é que estaremos estudando os princípios básicos deste tipo de banco de dados, ao longo deste Anexo. Caso você queira maiores detalhes sobre este tipo de banco de dados, existe uma farta Bibliografia, bem como inúmeros sites na Internet com informações sobre o assunto.

Conceitos Básicos de Bancos de Dados Relacionais

Neste item iremos revisar alguns conceitos básicos sobre Bancos de dados relacionais. Estes conceitos são importantes para a correta utilização dos bancos de dados, bem como para o projeto e criação dos mesmos.

Em muitas situações teremos que conectar nossas aplicações Web e páginas ASP.NET com bancos de dados já existentes; neste caso precisamos conhecer os conceitos aqui apresentados, para podermos utilizar o banco de dados de uma maneira otimizada.

Em outras situações teremos que criar o banco de dados a ser utilizado pela aplicação que está sendo desenvolvida. Neste caso, os conceitos apresentados neste capítulo auxiliam na criação de um banco de dados melhor estruturado e otimizado, tanto em termos de espaço de armazenamento, quanto da qualidade, confiabilidade e disponibilidade das informações nele contidas.

Veremos os seguintes conceitos:

- ◆ Entidades e Atributos, a base de um banco de dados
- ◆ Chave Primária
- ◆ Relacionamentos entre Entidades (Tabelas)
- ◆ Integridade Referencial
- ◆ Normalização de Tabelas
- ◆ Análise de um banco de dados relacional

Entidades e Atributos

Toda a informação de um Banco de dados relacional é armazenada em Tabelas, as quais também são chamadas de Entidades. Por exemplo, poderíamos ter uma Tabela “Clientes”, onde seriam armazenadas informações sobre os diversos clientes, uma tabela Produtos, onde são armazenadas informações sobre os produtos e assim por diante.

Para cada um dos Clientes poderíamos armazenar informações tais como: Nome, Rua, Bairro, Telefone, CEP, Data de Nascimento, etc.

Essas diversas características de cada Cliente são os “Atributos” do Cliente, muitas vezes chamados de “campos da entidade Cliente”, ou, de maneira mais simples: “Os campos da tabela Clientes”.

“O Conjunto de todos os Atributos de um cliente e os valores dos atributos forma o Registro do Cliente”. Com isso teremos a tabela constituída por um conjunto de Registros (uma linha completa com informações sobre o cliente) e cada Registro formado por um conjunto de atributos (Nome, Endereço, etc.).

Resumindo:

- ◆ Entidade ou Tabela – Um conjunto de registros sobre um determinado assunto.
- ◆ Campos ou Atributos – Características individuais de cada Entidade.

Considere o Exemplo da Figura 1.1, onde temos uma tabela Clientes com os seus diversos Campos (atributos):

No exemplo da Figura II.1, temos uma entidade: Clientes e seus diversos atributos:

- ◆ Código do Cliente
- ◆ Nome da Empresa
- ◆ Nome do Contato
- ◆ Cargo do Contato
- ◆ Endereço

Em cada linha temos um conjunto de atributos e seus valores. Cada linha forma um Registro que identifica um Cliente. Cada Coluna é um atributo da tabela Clientes.

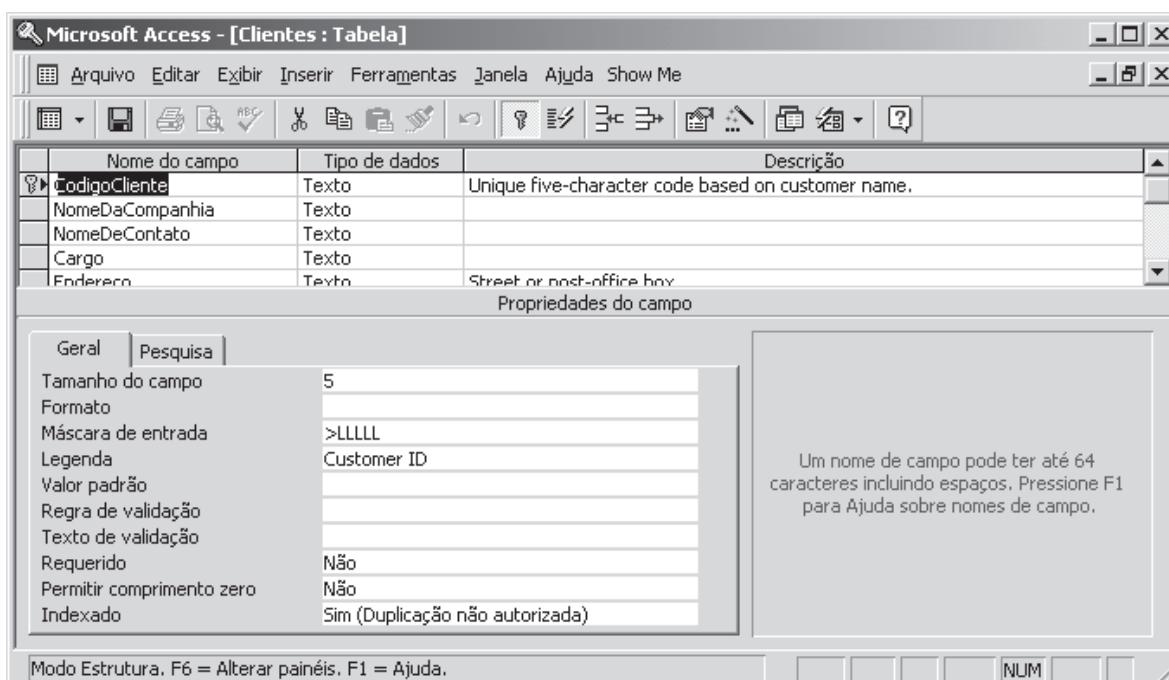


Figura II.1: A entidade (tabela) Clientes e seus diversos atributos (campos).

Um dos grandes desafios em se projetar um banco de dados com sucesso é a correta determinação das entidades que existirão no banco de dados, bem como dos atributos de cada entidade. Mais adiante veremos algumas dicas e técnicas para determinar as tabelas necessárias, bem como os campos necessários em cada tabela.

É importante lembrar que o que determina quais as tabelas e campos necessários é o escopo do problema que está sendo abordado. Por exemplo, se estamos desenvolvendo um sistema para acompanhamento do desempenho individual de cada funcionário, com certeza não teremos necessidade de uma tabela de Clientes. Por outro lado, se o desempenho de cada funcionário a ser acompanhado estiver ligado ao número de clientes atendidos pelo funcionário ou ao volume de vendas, a tabela Clientes passa a ter importância para o problema em questão.

Com isso podemos dizer que o que determina as tabelas e campos necessários é o problema real que o sistema a ser desenvolvido deverá solucionar.

Outro fato importante, e que iremos repetir ao longo deste ANEXO, é que cada tabela deve conter dados de SOMENTE um determinado assunto. Não devemos “misturar” dados de diversos assuntos em uma mesma tabela. Observe o exemplo da tabela indicada na Figura II.2:

TABELA DE CLIENTES E SEUS PEDIDOS					
Nome	Endereço	Fone	Nºm. Pedido	Data	Valor
José da Silva	Rua ABC – 40	222-2222	10234	10-01-2001	R\$ 450,00
José da Silva	Rua ABC – 40	222-2222	10345	12-01-2001	R\$ 370,00
José da Silva	Rua ACB – 40	222-2222	12321	23-02-2001	R\$ 234,30
Para Pedro	Rua YYU – 234	111-1111	12345	23-02-2001	R\$ 654,33
Para Pedro	Rua YYU – 234	111-1111	12444	28-02-2001	R\$ 456,70
Maria José	Rua BBB – 221	123-2222	12445	01-03-2001	R\$ 443,25
Maria José	Rua BBB – 221	123-2222	12446	07-03-2001	R\$ 500,32

Figura II.2: Uma tabela problemática, na qual estamos “misturando” assuntos.

Observe que, nesta tabela, cometemos o erro de “misturar” dois assuntos:

- ◆ Dados sobre os “Clientes”
- ◆ Dados sobre os “Pedidos dos Clientes”

Quando este tipo de situação acontece, temos uma série de problemas que irão se refletir em todo o sistema que está sendo desenvolvido. Dentre os principais problemas podemos citar:

- ◆ Informação repetida: Observe que para cada pedido de um determinado cliente, precisamos informar novamente os campos: Nome, Endereço e Fone.
- ◆ Informação inconsistente: Observe que por um erro de digitação, o nome do cliente José da Silva está digitado incorretamente (sem o acento) no segundo registro, e o seu endereço está digitado incorretamente, no terceiro registro. Isso causa inconsistências no banco de dados, gerando resultados errados quando forem feitas pesquisas pelo nome do cliente ou pelo endereço.

Para evitar este tipo de problema, deveríamos separar as informações dos Clientes e dos seus Pedidos em duas tabelas distintas. Veremos como fazer isso mais adiante neste Anexo.

O Conceito de Chave Primária

O Conceito de Chave Primária é fundamental para entender o funcionamento de um Banco de dados relacional. Vamos procurar entender o que significa um campo ser a Chave Primária de uma tabela.

Ao definirmos um campo como sendo uma Chave Primária, estamos informando ao banco de dados que não podem existir dois registros com o mesmo valor no campo Chave Primária, ou seja, os valores no campo Chave Primária precisam ser únicos.

Por exemplo, se defino um campo “Número da Identidade” da tabela Clientes como sendo uma Chave Primária, estou dizendo ao banco de dados que não podem existir dois clientes com o mesmo valor no campo “Número da Identidade”. Na prática estou garantindo que não podem ser cadastrados dois clientes com o mesmo Número de Identidade.

Em outras palavras poderíamos dizer que o Campo Chave Primária identifica de maneira única cada registro da tabela, isto é, de posse do valor da Chave Primária somente localizaremos um registro com aquele valor no campo Chave Primária.

Este é um conceito muito importante, pois conforme veremos mais adiante os conceitos de Integridade Referencial e Normalização estão diretamente ligados ao conceito de Chave Primária.

Alguns exemplos de campos que podem ser definidos como “Chave Primária” em suas respectivas tabelas:

- ◆ O campo Número do Pedido, em uma tabela de Pedidos.
- ◆ O campo Número da Identidade ou CPF em uma tabela de Clientes Pessoa Física.
- ◆ O campo Número do CNPJ em uma tabela de Clientes Pessoa Jurídica.

- ◆ O campo Código do Produto em uma tabela de Produtos.
- ◆ O campo ISBN em uma tabela de Livros.
- ◆ O campo Código do Autor em uma tabela de autores.

Na Figura I.3 vemos um exemplo da tabela Cliente onde o campo Código do Cliente é definido como uma Chave Primária. Observe que não existem dois clientes com o mesmo código.

Um detalhe importante é que a Chave Primária pode ser formada pela combinação de Mais do que um campo. Podem existir casos em que um único campo não é capaz de atuar como Chave Primária, pelo fato de este campo apresentar valores repetidos. Nestes casos podemos definir uma combinação de dois ou mais campos para ser a nossa Chave Primária. Além disso, uma tabela somente pode ter uma Chave Primária, seja ela simples ou composta.

Um cuidado especial que devemos ter é quanto ao desempenho das consultas em tabelas que possuem Chave Primária composta por mais do que um campo. Em muitas situações, o desempenho das consultas é inversamente proporcional ao tamanho da Chave Primária. Com isso quanto maior o tamanho da Chave Primária, menor o desempenho das consultas, isto é, mais demoradas se tornam as consultas. Na prática dificilmente teremos uma Chave Primária composta por mais do que 3 campos. Se você se deparar com uma situação em que precise de uma Chave Primária composta de quatro ou mais campos, revise o projeto do banco de dados, porque devem existir alguns problemas.

	Customer ID	Company Name	Contact Name	Contact Title
▶	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative
	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner
	ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner
	AROUT	Around the Horn	Thomas Hardy	Sales Representative
	BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator
	BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative
	BLONP	Blondel père et fils	Frédérique Citeaux	Marketing Manager
	BOLID	Bólido Comidas preparadas	Martín Sommer	Owner
	BONAP	Bon app'	Laurence Lebihan	Owner
	BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager
	BSBEV	B's Beverages	Victoria Ashworth	Sales Representative
	CACTU	Cactus Comidas para llevar	Patricia Simpson	Sales Agent
	CENTC	Centro comercial Moctezuma	Francisco Chang	Marketing Manager
	CHOPS	Chop-suey Chinese	Yang Wang	Owner

Figura II.3: O campo Código do Cliente é uma Chave Primária.

Relacionamentos Entre Tabelas

Na prática, em um Banco de dados relacional, podem existir diversas tabelas, como por exemplo:

- ◆ Clientes
- ◆ Produtos

- ◆ Pedidos
- ◆ Detalhes do Pedido
- ◆ Fornecedores
- ◆ Categorias
- ◆ Funcionários, etc.

Embora as informações estejam separadas em cada uma das Tabelas, devemos ter algum mecanismo que nos permita reunir dados de duas ou mais tabelas em um relatório ou consulta. Por exemplo, para fazer um relatório do total de vendas por funcionário, precisarei de informações das seguintes tabelas:

- ◆ Funcionários
- ◆ Pedidos
- ◆ Detalhes do pedido

O mecanismo que nos permite acessar, de maneira consolidada, dados de diversas tabelas é chamado de “Relacionamento entre tabelas”.

Por exemplo: Um Pedido é feito por um Cliente e neste Pedido podem existir diversos Itens, os quais são armazenados na tabela Detalhes do Pedido. Além disso cada Pedido possui um número único, mas um mesmo Cliente pode fazer diversos pedidos.

Veja que o parágrafo acima descreve relações do mundo real. Estas relações do mundo real são o nosso guia, para definir as relações entre as diversas tabelas do banco de dados.

Em um banco de dados precisamos de um mecanismo para representar estes relacionamentos da vida Real, em termos das tabelas e seus atributos. Isto é possível com a utilização dos Relacionamentos, os quais podem ser de três tipos:

- ◆ Um para Um
- ◆ Um para Vários
- ◆ Vários para Vários

Vamos analisar cada um desses tipos, individualmente.

Relacionamento do Tipo Um Para Um

Esta relação existe quando os campos que se relacionam são ambos Chaves Primárias em suas respectivas tabelas. Cada um dos campos não apresenta valores repetidos. Na prática existem poucas situações onde utilizaremos um relacionamento deste tipo.

Vamos imaginar o seguinte exemplo: Imagine uma escola com um cadastro de Alunos na tabela Alunos; destes apenas uma pequena parte participa da Banda da Escola. Por questões de projeto do banco de dados, podemos optar por criar uma segunda tabela “Alunos da Banda”, a qual pode se relacionar com a tabela Alunos através de um relacionamento

Um para Um. Cada aluno somente é cadastrado uma vez na tabela Alunos e uma única vez na tabela Alunos da Banda. Poderíamos utilizar o campo “Matrícula do Aluno” como o Campo que relaciona as duas tabelas.

Na tabela Alunos da Banda poderíamos colocar apenas o Número da Matrícula do aluno, além das informações a respeito do Instrumento que ele toca, tempo em que está na banda, etc. Quando fosse necessário buscar as informações tais como nome, endereço, etc., as mesmas podem ser recuperadas através do relacionamento existente entre as duas tabelas, evitando, com isso, que a mesma informação (Nome, Endereço, etc.) tenha que ser duplicada nas duas tabelas, o que diminui a probabilidade de erros de digitação.

Na Figura II.4 vemos o exemplo de um relacionamento do tipo Um para Um entre as tabelas “Alunos” e “Alunos da Banda.”

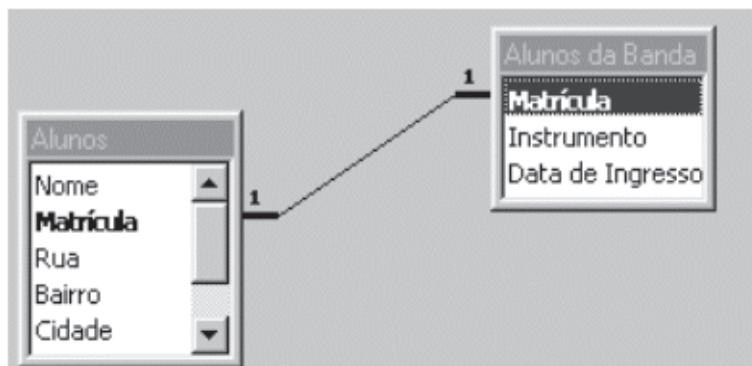


Figura II.4: Um relacionamento do tipo Um para Um.

Relacionamento do Tipo Um Para Vários

Este, com certeza, é o tipo de relacionamento mais comum entre duas tabelas. Uma das tabelas (o lado um do relacionamento) possui um campo que é a Chave Primária e a outra tabela (o lado vários) se relaciona através de um campo cujos valores podem se repetir – este campo é conhecido como Chave Estrangeira.

Considere o exemplo entre a tabela Clientes e a tabela Pedidos. Cada Cliente somente é cadastrado uma única vez (por isso o campo “Código do Cliente” é uma Chave Primária, indicando que não podem existir dois clientes com o mesmo código), portanto a tabela Clientes será o lado um do relacionamento. Porém cada cliente pode fazer diversos pedidos; por isso que o campo “Código do Cliente” poderá aparecer várias vezes na tabela Pedidos, tantas vezes quantos forem os pedidos que o Cliente tenha feito. Por isso que temos um relacionamento do tipo Um para Vários entre a tabela Clientes e Pedidos, através do campo “Código do Cliente”, indicando que um mesmo Cliente pode fazer diversos pedidos.

Na Figura II.5 vemos um exemplo de um relacionamento Um para Vários entre as Tabelas Clientes e Pedidos, através do campo código do cliente.

No lado “Um” do relacionamento o campo é definido como uma Chave Primária (campo CódigoDoCliente na tabela Clientes) e no lado “Vários” não (campo CódigoDoCliente na tabela Pedidos), indicando que no lado vários o Código do Cliente pode se repetir, o que faz sentido, uma vez que um mesmo cliente pode fazer diversos pedidos.

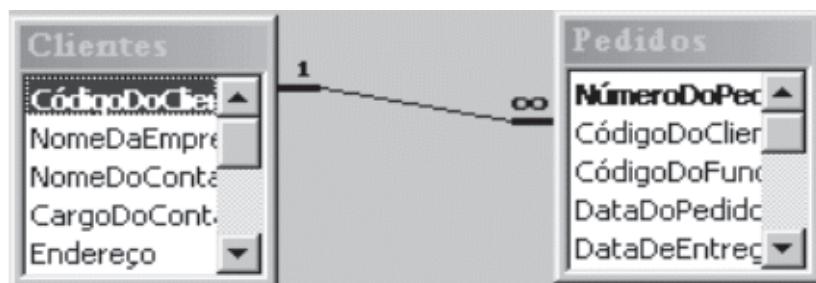


Figura II.5: Um relacionamento do tipo Um para Vários.

Podemos citar outro exemplo clássico de relacionamentos do tipo Um para Vários:

Entre as tabelas Pedidos e Detalhes do Pedido, através do campo NúmeroDoPedido. Na tabela Pedidos o campo NúmeroDoPedido é Chave Primária. Na tabela Detalhes do Pedido, o campo NúmeroDoPedido não é Chave Primária. Na tabela Detalhes do Pedido temos as informações sobre os itens de cada pedido. Este relacionamento indica que “um” mesmo “pedido” pode ter “diversos” “itens”, o que faz sentido.

Na Figura II.6 vemos um exemplo de um relacionamento Um para Vários entre as Tabelas Pedidos e Detalhes do Pedido, através do campo NúmeroDoPedido.

NOTA: O campo do lado vários do relacionamento também é conhecido como sendo uma “Chave Estrangeira”. No exemplo da Figura II.6, o campo NúmeroDoPedido, na tabela Detalhes do Pedido, seria a nossa Chave Estrangeira do relacionamento em questão.

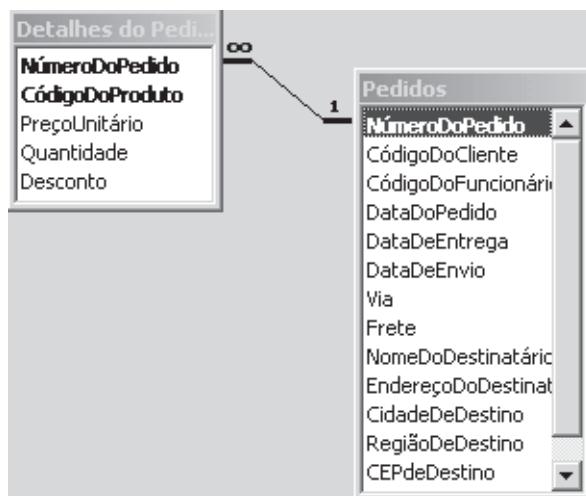


Figura II.6: Mais um exemplo de relacionamento do tipo Um para Vários.

Relacionamento do Tipo Vários Para Vários

Este tipo de relacionamento ocorre em uma situação onde, nos dois lados do relacionamento, os valores podem se repetir. Vamos considerar o caso entre as tabelas Produtos e a tabela Pedidos. Posso ter vários Pedidos nos quais aparece um determinado produto; além disso vários Produtos podem aparecer no mesmo Pedido. Esta é uma situação em que temos um relacionamento do tipo Vários para Vários: Vários produtos podem aparecer em Vários pedidos e Vários pedidos podem conter Vários produtos.

Na prática não temos como implementar um relacionamento deste tipo, devido a uma série de problemas que este tipo de relacionamento implicaria.

Para evitar este problema é bastante comum “quebrarmos” um relacionamento do tipo Vários para Vários em dois relacionamentos do tipo Um para Vários. Isso é feito através da criação de uma nova tabela, a qual fica com o lado Vários dos relacionamentos. No nosso exemplo poderíamos criar a tabela Detalhes do Pedido, onde ficam armazenadas as informações sobre os diversos itens de cada pedido. Desta forma, ao invés de termos um relacionamento do tipo Vários para Vários, teremos dois relacionamentos do tipo um para vários, conforme indicado na Figura II.7.

Esta situação em que um relacionamento Vários para Vários é “quebrado” em dois relacionamentos do tipo Um para Vários é bastante comum. Diversas vezes utilizamos esta técnica para eliminar problemas no banco de dados, tais como informação repetida e inconsistência de dados.

Agora que já conhecemos os tipos de relacionamentos existentes, no próximo item veremos o conceito de “Integridade Referencial.” O mecanismo da Integridade Referencial é utilizado para garantir a consistência dos dados.

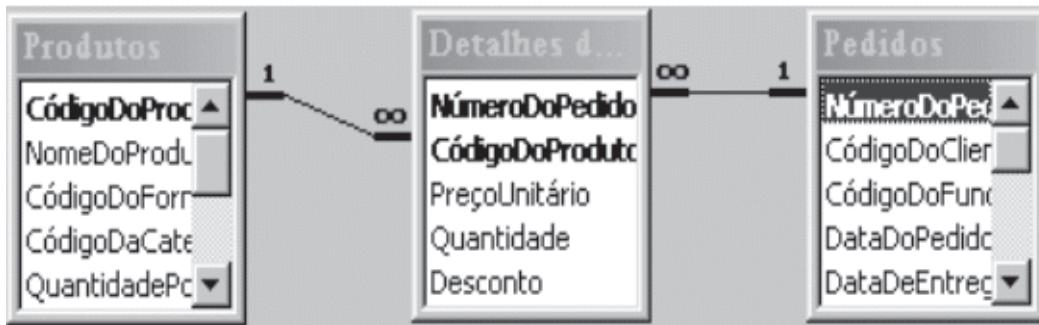


Figura II.7: “Quebrando” um relacionamento Vários para Vários.

Integridade Referencial

A Integridade Referencial é utilizada para garantir a integridade dos dados entre as diversas tabelas relacionadas, evitando inconsistências nos dados, bem como repetições desnecessárias.

Por exemplo, existe um relacionamento do tipo Um para Vários entre a tabela Clientes e a tabela Pedidos (um cliente pode fazer vários pedidos). Com a Integridade Referencial, o banco de dados não permite que seja cadastrado um Pedido para um Cliente que ainda não foi cadastrado na tabela Clientes. Através da Integridade Referencial, também podemos garantir o seguinte:

- ◆ Quando o Código de um cliente for alterado na tabela Clientes, o banco de dados atualiza, automaticamente, todos os Códigos do Cliente na tabela Pedidos, de tal maneira que não fiquem registros Órfãos, isto é, registros de Pedidos com um Código de Cliente que não existe mais na tabela Clientes. Essa ação é conhecida como “Propagar atualização dos campos relacionados” ou “Propagar atualizações em cascata”.
- ◆ Quando um cliente for excluído da tabela Clientes, podemos fazer com que o banco de dados exclua, na tabela Pedidos, todos os Pedidos para o cliente que está sendo excluído. Essa opção é conhecida como “Propagar

exclusão dos registros relacionados” ou “Propagar exclusões em cascata”, e pode ser habilitada ou não, dependendo do projeto do banco de dados. Caso seja necessário manter todo o histórico de compras do cliente, por exemplo, esta opção não deve ser habilitada. Com isso, quando o cliente for eliminado da tabela Clientes, os seus pedidos continuarão gravados na tabela Pedidos.

Essas opções são definidas no momento da criação do banco de dados, quando são criadas as tabelas e definidos os relacionamentos entre as tabelas.

A opção de Propagar atualização dos campos relacionados é utilizada na maioria das situações; já a opção de Propagar exclusão dos registros relacionados deve ser estudada caso a caso.

Normalização de Tabelas

O objetivo da normalização é evitar os problemas provocados por falhas no Projeto do banco de dados, bem como eliminar a “mistura” de assuntos e as correspondentes redundâncias de dados.

Uma “Regra de Ouro” que devemos observar quando do projeto de banco de dados é a de “Não Misturar assuntos em uma mesma Tabela”.

Por exemplo na tabela Clientes devemos colocar somente campos relacionados com o assunto Clientes. Não devemos misturar campos relacionados com outros assuntos, tais como Pedidos, Produtos, etc. Essa “Mistura de Assuntos” em uma mesma tabela acaba por gerar repetição desnecessária dos dados bem como inconsistências.

O Processo de Normalização aplica uma série de regras sobre as entidades de um banco de dados, para verificar se estas estão corretamente projetadas. Embora existam 5 formas normais (ou regras de Normalização), na prática usamos um conjunto de três Formas Normais.

Normalmente, após a aplicação das regras de Normalização, algumas tabelas acabam sendo divididas em duas ou mais tabelas, o que no final acaba gerando um número maior de tabelas do que o originalmente existente. Este processo causa a simplificação dos atributos de uma tabela, colaborando significativamente para a estabilidade do modelo, reduzindo-se consideravelmente as necessidades de manutenção.

Vamos estudar e entender o Processo de Normalização na Prática, através de exemplos.

Primeira Forma Normal

Regra: “Uma tabela está na Primeira Forma Normal quando seus atributos não contêm Grupos de Repetição.”

Por isso dissemos que uma tabela que possui Grupos de Repetição não está na Primeira Forma Normal. Considere a tabela Indicada na Figura II.8:

Podemos notar que uma tabela com esta estrutura apresenta diversos problemas. Por exemplo, se um casal tiver mais do que um filho, teríamos que digitar o Nome do Pai e da Mãe diversas vezes, tantas quantos forem os filhos. Isso forma um “Grupo de Repetição”. Além do mais pode ser que, por erro de digitação, o nome dos pais não seja digitado

exatamente igual todas as vezes, o que pode acarretar problemas na hora de fazer pesquisas ou emitir relatórios. Este problema ocorre porque “Misturamos Assuntos” em uma mesma tabela. Colocamos as informações dos Pais e dos Filhos em uma mesma tabela.

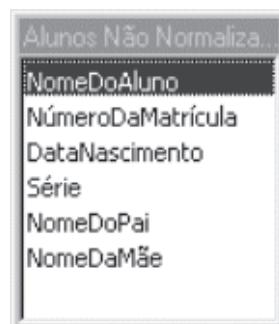


Figura II.8: Uma tabela que não está na Primeira Forma Normal.

A Resolução para este problema é simples:

“Criamos uma tabela separada para a Informação dos Pais e relacionamos a tabela Pais com a tabela Filhos através de um relacionamento do tipo Um para Vários, ou seja, Um casal pode ter Vários filhos.

Esta solução é indicada na Figura II.9.

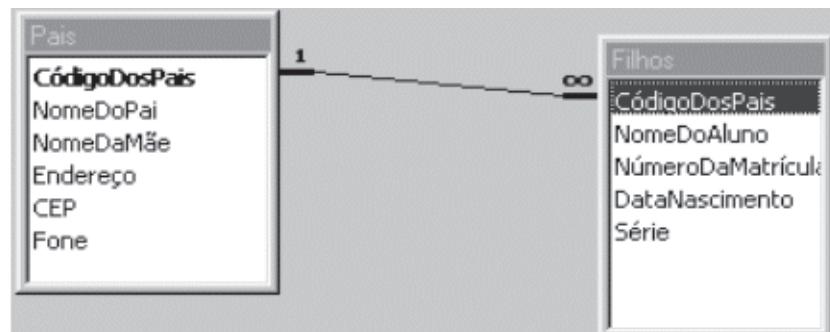


Figura II.9: As tabelas Pais e Filhos estão na primeira forma normal.

As duas tabelas resultantes da aplicação da primeira forma normal: Pais e Filhos estão na primeira forma normal; a tabela original, a qual misturava informações de Pais e Filhos, não estava na primeira forma normal.

Segunda Forma Normal

Podemos aplicar a segunda forma normal quando tivermos uma chave primária composta por mais de um campo, isto é, uma chave primária composta. Neste caso, devemos observar se todos os campos que não fazem parte da chave dependem de todos os campos que compõem a Chave Primária. Se algum campo depender somente de parte da chave composta, então este campo deve pertencer a outra tabela.

Observe o exemplo indicado na tabela da Figura II.10.

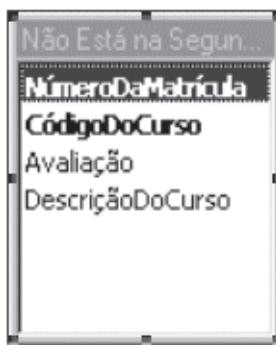


Figura II.10: Uma tabela que não está na Segunda Forma Normal.

A Chave Primária composta é formada pela combinação dos campos NúmeroDaMatrícula e CódigoDoCurso. O campo Avaliação depende tanto do CódigoDoCurso quanto do NúmeroDaMatrícula, porém o campo DescriçãoDoCurso depende apenas do CódigoDoCurso. Com isso temos um campo que não faz parte da Chave Primária e depende apenas de um dos campos que compõem a Chave Primária. Com isso dizemos que esta tabela não está na segunda forma normal.

A resolução para este problema também é simples: Dividimos a tabela, que não está na segunda forma normal, em duas outras tabelas, conforme indicado pela Figura II.11, sendo que as duas tabelas resultantes estão na segunda forma normal.



Figura II.11: Duas tabelas que estão na Segunda Forma Normal.

Terceira Forma Normal

Na definição dos campos de uma entidade podem ocorrer casos em que um campo não seja dependente diretamente da Chave Primária, ou de parte dela, mas sim dependente de um outro atributo constante na tabela, atributo este que não é a Chave Primária.

Quando isto ocorre, dizemos que a tabela não está na terceira forma normal, conforme indicado pela tabela da Figura II.12.

Observe que o campo DescriçãoDoCurso depende apenas do campo CódigoDoCurso, o qual não faz parte da Chave Primária. Por isso dizemos que esta tabela não está na terceira forma normal.

IMPORTANTE: A Distinção entre a Segunda e a Terceira forma normal, que veremos no próximo item, muitas vezes é confusa. A Segunda Forma normal está ligada à ocorrência de chaves primárias compostas.

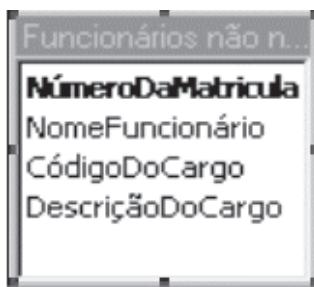


Figura II.12: Uma tabela que não está na terceira forma normal.

A solução para este caso também é simples. Novamente basta dividir a tabela em duas outras, conforme indicado pela Figura II.13. As duas tabelas resultantes estão na terceira forma normal.



Figura II.13: Duas tabelas que estão na terceira forma normal.

Passos Para Projetar um Banco de Dados

Neste item iremos apresentar os passos básicos para projetar um banco de dados. Aplicaremos os conhecimentos sobre Entidades, Atributos, Relacionamentos, Chave Primária e Normalização.

Um banco de dados bem projetado fornece um acesso conveniente às informações desejadas. Com uma boa estrutura, gasta-se menos tempo na construção de um banco de dados e, ao mesmo tempo, asseguram-se resultados mais rápidos e precisos.

Etapas na estruturação de um banco de dados:

- ◆ Determinar qual o objetivo do banco de dados. Isto ajuda na determinação de quais dados devem ser armazenados.
- ◆ Determinar as tabelas necessárias. Após definirmos o objetivo do banco de dados, as informações devem ser definidas e separadas em assuntos diferentes, tais como Clientes, Empregados, Pedidos, pois cada um irá compor uma tabela no banco de dados.
- ◆ Determinar os campos necessários em cada tabela. Definir quais informações devem ser mantidas em cada tabela. Por exemplo, a tabela Clientes poderia ter um campo para o Código Do Cliente; outro para o Nome Do Cliente e assim por diante.

IMPORTANTE: Com isso podemos concluir que, como resultado do processo de Normalização, iremos obter um número maior de tabelas, porém sem problemas de redundância e inconsistência dos dados, ou com estes problemas minimizados.

- ◆ Determinar quais campos serão as chaves primárias. Determinar, em cada tabela, qual ou quais campos serão utilizados como Chave Primária. Esta é uma etapa importante para a definição dos Relacionamentos que vêm a seguir.
- ◆ Determinar os relacionamentos. Decidir como os dados de uma tabela se relacionam com os dados de outras tabelas. Por exemplo, Clientes podem fazer Vários Pedidos. Fornecedores podem fornecer Vários Produtos, etc.
- ◆ Refinar a estrutura do banco de dados. Antes de inserir muitos dados, ou até mesmo antes de inserir qualquer dado, verificar se a estrutura contém erros, isto é, verificar se os resultados obtidos são os desejados. Isto, normalmente, pode ser obtido através do processo de Normalização.

Com uma boa estrutura, gasta-se menos tempo na construção e manutenção do banco de dados e, ao mesmo tempo, asseguram-se resultados mais rápidos e precisos.

Dicas para determinação dos campos em uma tabela:

- ◆ Relacionar diretamente cada campo ao assunto da tabela. Se um campo descreve o assunto de uma tabela diferente, este campo deve pertencer a outra tabela. O mesmo acontece quando uma informação se repete em diversas tabelas. Este é um indício de que existem campos desnecessários em algumas tabelas.
- ◆ Não incluir dados derivados ou calculados. Não é recomendado armazenar o resultado de cálculos nas tabelas. O correto é que o cálculo seja executado quando necessitarmos do resultado.
- ◆ Incluir todas as informações necessárias. Como é fácil esquecer informações importantes, deve-se ter em mente todas as informações coletadas desde o início do processo e perguntar se com elas é possível obter todos os resultados esperados.
- ◆ Armazenar todas as informações separadamente. Existe uma tendência em armazenar informações em um único campo. Por exemplo, o nome do curso e o tempo de duração em um mesmo campo. Como as duas informações foram combinadas em um único campo, ficará difícil conseguir um relatório classificado pelo tempo de duração dos cursos, por exemplo.

Como escolher o campo que será a Chave Primária?

Um bom Sistema Gerenciador de Banco de Dados Relacionais (SGBDR) é aquele que encontra e nos fornece, rapidamente, todas as informações necessárias que nele estejam armazenadas, mesmo em diferentes tabelas. Para que isto seja possível é necessário incluir um campo ou conjunto de campos que identifiquem de um modo único cada registro de uma tabela. Esta informação é chamada Chave Primária, conforme descrito anteriormente. Deve-se ter certeza de que este campo (ou conjunto de campos) seja sempre diferente para cada registro, por não serem permitidos valores duplicados em um campo de Chave Primária.

Ao escolher campos de Chave Primária, considere os seguintes detalhes:

NOTA: Neste item estamos falando do projeto físico do banco de dados, isto é, a definição de quais tabelas e do relacionamento entre as mesmas. O resultado final desta fase será o que chamamos de Diagrama Entidade x Relacionamentos. O projeto completo de uma aplicação de n camadas que irá rodar na rede de uma empresa envolve várias outras etapas. A descrição completa de todas as etapas foge ao escopo deste anexo, sendo disciplina de Análise de Sistemas. Um bom livro sobre o assunto é o seguinte: "Analyzing Requirements and Defining Solution Architectures MCSD Training Kit", Microsoft Press, ISBN: 0735608547

- ◆ Não é permitido duplicidade de valores ou nulos (informações desconhecidas). Caso não exista um identificador único para uma determinada tabela, pode-se usar um campo que numere os registros seqüencialmente.
- ◆ Pode-se utilizar o valor deste campo para encontrar registros.
- ◆ O tamanho da Chave Primária afeta a velocidade das operações; portanto, para um melhor desempenho, devemos utilizar o menor tamanho que acomode os valores necessários para armazenar no campo.

Na Figura II.14 temos um exemplo de um Diagrama Entidade x Relacionamentos. Este diagrama mostra a estrutura do Banco de dados NorthWind.mdb, o qual é fornecido juntamente com o Microsoft Access e também como exemplo na instalação do Microsoft SQL Server 2000. Os campos que são Chave Primária estão indicados em negrito.

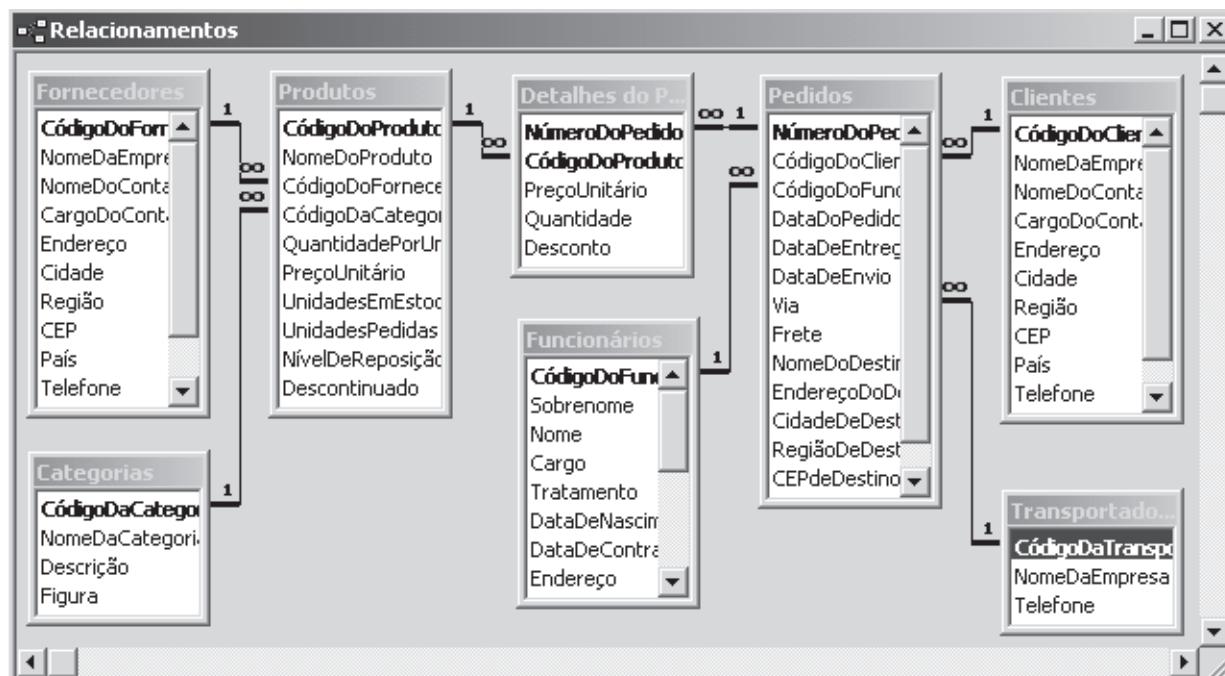


Figura II.14: Um diagrama Entidade x Relacionamentos.

Conclusão

Os conceitos apresentados neste anexo fornecem as bases necessárias para que você entenda o modelo de dados relacional. Sem um correto entendimento deste modelo, fica muito difícil entender e utilizar as classes do Framework .NET para trabalho com banco de dados.

Introdução

Nos exemplos deste livro, utilizamos uma série de comandos para realizar uma série de operações sobre os dados armazenados, como por exemplo:

- ◆ Selecionar um conjunto de registros com base em critérios especificados
- ◆ Ordenar um conjunto de registros com base em um ou mais campos de dados
- ◆ Alterar informações no banco de dados
- ◆ Excluir informações no banco de dados
- ◆ Inserir informações no banco de dados

Para realizar estas operações, é utilizada uma linguagem conhecida com SQL – Structured Query Language. A linguagem SQL foi desenvolvida pela IBM, porém, devido ao seu poder e facilidade de utilização, rapidamente tornou-se um padrão de mercado, sendo hoje utilizada pela maioria dos bancos de dados que seguem o modelo Relacional.

ANEXO

3

A Linguagem SQL

NOTA: Pequenas diferenças podem existir entre os comandos SQL de diferentes bancos de dados como por exemplo o Microsoft SQL Server e o Oracle. Neste anexo estaremos utilizando a sintaxe dos comandos SQL utilizados pelo Microsoft SQL Server.

Noções Básicas da Linguagem SQL – Structured Query Language

Como o próprio nome sugere – Microsoft SQL Server – a linguagem utilizada pelo Microsoft SQL Server 2000 é o SQL – Structured Query Language. O SQL é utilizado para uma série de operações, conforme descrito na Introdução deste anexo.

Neste tópico teremos uma noção básica da linguagem SQL. Aprenderemos os comandos para manipulação de dados – consultar, inserir, alterar e excluir.

Conhecendo o SQL

O SQL foi desenvolvido para ser uma linguagem padrão para operações em banco de dados. A linguagem SQL foi elaborada para ser independente de hardware ou

software. Ao usar SQL, você não precisa saber a respeito do software de banco de dados ou do hardware envolvido em uma operação. Tudo o que você precisa conhecer é o método (instrução) SQL padrão para solicitar informações, que obrigatoriamente é o mesmo (não sejamos tão otimistas, digamos que é “quase o mesmo”), em todos os sistemas que utilizam o SQL.

Obs.: Existem pequenas diferenças nas implementações do SQL de diferentes fabricantes. Desta forma algumas rotinas escritas utilizando o SQL Plus do Oracle podem não rodar, sem alterações no Microsoft SQL ou do Microsoft Access e vice-versa.

Na listagem a seguir temos um exemplo de uma instrução SQL. Embora a mesma pareça complexa, neste primeiro momento, veremos que a Linguagem SQL é bastante simples e de fácil aprendizado.

Exemplo de um comando SQL:

```
SELECT Orders.OrderID, Orders.OrderDate, Orders.ShipCountry, Orders.ShipCity  
FROM Orders  
WHERE Orders.ShipCountry Like '[A-M]%'  
AND  
Orders.OrderDate > 01/01/1997  
ORDER BY Orders.OrderDate
```

Uma Instrução SQL descreve o conjunto de dados que você deseja recuperar (quais campos, de quais tabelas, Critérios de filtragem, classificação, Expressões Calculadas, etc.).

Todas as instruções SQL são conduzidas com um único comando que contém uma descrição completa da informação exigida. Ao escrever uma instrução SQL, você não deve se preocupar em como os dados são recuperados, mas somente com o conteúdo do conjunto de dados. Esse é o principal benefício do método SQL.

Lembre-se de que o SQL é um padrão de mercado para expressões de consulta em banco de dados. Embora a maior parte das versões do SQL compartilhe elementos básicos, elas não são idênticas. O Access SQL usa algumas palavras-chave que você não encontra em outras versões do SQL. Também existem pequenas diferenças entre o Microsoft Access e o Microsoft SQL Server.

Veremos através de exemplos a utilização das principais palavras-chaves do SQL para a construção de expressões SQL para pesquisa em banco de dados.

A Instrução SELECT

Esta com certeza é a Instrução SQL mais importante; não existe pesquisa que não utilize esta instrução. Vamos conhecê-la em detalhes.

A Instrução Select é utilizada para especificar quais os campos de quais tabelas farão parte da consulta, quais os critérios de pesquisa que serão utilizados, qual a ordem de classificação, etc.

A sintaxe simplificada da instrução é conforme indicado abaixo:

```
SELECT  
campo1, campo2, campo3  
FROM nome_da_tabela  
[WHERE condição ]  
[GROUP BY nome_do_campo ]  
[HAVING ... ]  
[ORDER BY... ]
```

Na Tabela III.1 temos uma descrição destes elementos:

Tabela III.1 Os principais elementos da instrução Select.

Elemento	Descrição
campo1, campo2, ... campon	Nome dos campos a serem retornados a partir de uma ou mais tabelas.
nome_da_tabela	Nome da tabela a partir da qual os dados devem ser recuperados.
WHERE	Permite que sejam especificados critérios de pesquisa.
GROUP BY	Podemos agrupar os resultados em torno de um ou mais campos de dados. Por exemplo, em um relatório de vendas anuais, posso ter os resultados agrupados por mês, com a soma das vendas do mês logo após a listagem de vendas do mês.
HAVING	Especifica critérios para serem utilizados juntamente com Group By.
ORDER BY	Podemos ordenar os resultados obtidos com base em um ou mais campos de dados. Por exemplo, podemos ordenar uma listagem de vendas em ordem alfabética do nome do cliente ou do nome do vendedor.

Vamos analisar alguns exemplos práticos de utilização da Linguagem SQL.

Exemplo 1: Criar uma instrução SQL que retorne o campo NúmeroDoPedido, o campo DataDoPedido, o campo Frete e o campo PaísDeDestino da tabela Pedidos.

```
SELECT Pedidos.NúmeroDoPedido, Pedidos.DataDoPedido, Pedidos.PaísDeDestino,  
Pedidos.Frete FROM Pedidos
```

Observe que os nomes dos campos estão separados por vírgula, e além disso estamos utilizando o nome completo, isto é: Nome_da_tabela.Nome_do_campo. Também poderíamos utilizar o comando, sem o nome da tabela antes do nome do campo, conforme indicado no seguinte comando:

```
SELECT NúmeroDoPedido, DataDoPedido, PaísDeDestino, Frete FROM Pedidos
```

Por exemplo, imagine que estejamos escrevendo uma instrução SQL para criar uma listagem com o Código e o Nome do Cliente, bem como todos os pedidos efetuados pelo Cliente. Acontece que o código do Cliente, no nosso exemplo, existe nas duas tabelas: Clientes e Pedidos. Neste caso devemos especificar o nome da tabela, antes do nome do campo.

Exemplo 2: Alterar a instrução SQL anterior para que os registros sejam classificados em ordem crescente pelo valor do Frete.

IMPORTANTE: Somente é obrigatória a utilização do nome da tabela antes do nome do campo quando o mesmo campo existir em duas ou mais tabelas que fazem parte da instrução Select.

```
SELECT NúmeroDoPedido, DataDoPedido, PaísDeDestino, Frete FROM Pedidos
ORDER BY Frete
```

Observe a utilização da cláusula “ORDER BY Frete” para classificar os registros em ordem Crescente. A classificação em ordem crescente é o padrão . Quando formos classificar em ordem decrescente, precisamos especificar a palavra DESC, conforme indicado no seguinte comando:

```
SELECT NúmeroDoPedido, DataDoPedido, PaísDeDestino, Frete FROM Pedidos
ORDER BY Frete DESC
```

Exemplo 3: Agora vamos inserir condições. Muitas vezes as condições são chamadas de filtro, uma vez estabelecida uma condição, somente os registros que “atendem” a condição especificada serão retornados. Desta forma a condição atua como se fosse um filtro. Neste exemplo vamos alterar a instrução SQL anterior para que sejam exibidos somente os pedidos cujo PaísDeDestino seja Brasil.

```
SELECT NúmeroDoPedido, DataDoPedido, PaísDeDestino, Frete FROM Pedidos
WHERE PaísDeDestino='Brasil'
ORDER BY Frete
```

Observe a utilização da Cláusula WHERE para filtrar somente os pedidos cujo PaísDeDestino seja Brasil. Como o campo PaísDeDestino é um campo do tipo texto, o valor do critério (Brasil) tem que vir entre apóstrofes. Vamos trabalhar um pouco mais com a cláusula WHERE.

Exemplo 4: Altere a instrução SQL anterior para que sejam exibidos somente os pedidos para o Brasil ou Alemanha como PaísDeDestino.

```
SELECT NúmeroDoPedido, DataDoPedido, PaísDeDestino, Frete FROM Pedidos
WHERE Pedidos.PaísDeDestino='Brasil' Or Pedidos.PaísDeDestino='Alemanha'
ORDER BY Frete
```

Observe a utilização da cláusula OR ligando os dois Critérios. Lembre que a cláusula OR retorna um registro se o PaísDeDestino atender um dos dois critérios, isto é, se for Brasil ou se for Alemanha, o registro será selecionado, que é exatamente o que desejamos, ou seja, todos os pedidos para o Brasil ou para a Alemanha.

Exemplo 5: Altere a instrução SQL anterior, retirando o critério para PaísDeDestino. Adicione um critério para NúmeroDoPedido maior do que 10500, retire a classificação do campo Frete a classifique pelo campo NúmeroDoPedido.

```
SELECT NúmeroDoPedido, DataDoPedido, PaísDeDestino, Frete FROM Pedidos
WHERE NúmeroDoPedido>10500
ORDER BY NúmeroDoPedido
```

Observe a cláusula WHERE utilizando o operador de comparação maior do que (>) e a classificação através da cláusula ORDER BY no campo NúmeroDoPedido.

Podemos ver, através dos exemplos, que a linguagem SQL não é tão difícil como pode parecer à primeira vista. Observe que a sintaxe da linguagem é bastante intuitiva e orientada à extração de dados através das consultas.

Vamos continuar analisando alguns exemplos de aplicação da Linguagem SQL com a utilização de recursos mais avançados.

Exemplo 6: Alterar a instrução SQL anterior, e adicionar um critério de tal maneira que somente sejam exibidos os pedidos para o Ano de 1995. Tirar o critério do campo Número do Pedido.

```
SELECT NúmeroDoPedido, DataDoPedido, PaísDeDestino, Frete FROM Pedidos
WHERE Year(DataDoPedido)=1995
ORDER BY NúmeroDoPedido
```

Observe a utilização da função Year para extraímos apenas o Ano do campo DataDoPedido a fim de especificarmos como critério o Ano=1995. Também a cláusula Order By foi mantida, classificando a listagem em ordem crescente pelo número do pedido. A utilização de funções junto com os comando SQL nos fornece inúmeras possibilidades de refinamento em nossas consultas.

Exemplo 7: Alterar a instrução SQL anterior, para que sejam exibidos somente os pedidos no Período de 01/01/1995 a 31/07/1995 e que tenham como PaísDeDestino Brasil, Argentina, Alemanha ou Canadá.

```
SELECT NúmeroDoPedido, DataDoPedido, PaísDeDestino, Frete FROM Pedidos
WHERE DataDoPedido Between '1/1/95' And '8/31/95'
AND
PaísDeDestino In ('Brasil','Argentina','Alemanha','Canadá')
ORDER BY NúmeroDoPedido
```

Observe a utilização de vários critérios em diferentes campos. Colocamos critérios nos campos DataDoPedido e PaísDeDestino. Os critérios de dois ou mais campos são ligados através do operador AND, indicando que um registro deve atender ambos os critérios para ser selecionado. Também temos a utilização dos operadores Between (Entre) para selecionar as datas dentro de um determinado intervalo e do operador In (Em) para selecionar o campo PaísDeDestino que seja igual a um dos valores apresentados na lista. Observe, também, que os valores de data vêm delimitados por apóstrofes (''). Se você estiver utilizando o Microsoft Access, o valor para os campos do tipo Data deve vir entre sinais de #, ao invés do apóstrofe.

Exemplo 8: Criar uma instrução SQL que retorne o campo NúmeroDoPedido, o campo DataDoPedido, o campo DataDeEntrega, o campo Frete e o campo PaísDeDestino da tabela Pedidos. Criar uma coluna adicional que calcula o número de dias entre a DataDeEntrega e a DataDoPedido. Chamar esta coluna de Dias_Ped_Entr.

```
SELECT NúmeroDoPedido, DataDoPedido, DataDeEntrega, PaísDeDestino, Frete,  
DataDeEntrega-DataDoPedido AS Dias_Ped_Estr FROM Pedidos  
  
WHERE (DataDoPedido Between '1/1/95' And "8/31/95")  
  
AND  
  
(PaísDeDestino In ('Brasil', 'Argentina', 'Alemanha', 'Canadá'))  
  
ORDER BY Pedidos.NúmeroDoPedido
```

Veja que a coluna calculada DataDeEntrega-DataDoPedido está junto com a listagem dos campos no início da instrução Select. Além disso foi utilizada a palavra AS para atribuir um nome (apelido) para esta coluna calculada. Este nome é o que será utilizado para fazer referência à coluna, em uma página ASP.NET ou um programa desenvolvido em Visual Basic, por exemplo.

Você também pode classificar a listagem em ordem crescente ou decrescente de um campo calculado. Por exemplo, se você quisesse classificar a listagem do item anterior, em ordem crescente, do número de dias entre a DataDeEntrega e a DataDoPedido, bastaria utilizar a seguinte instrução SQL:

```
SELECT NúmeroDoPedido, DataDoPedido, DataDeEntrega, PaísDeDestino, Frete,  
DataDeEntrega-DataDoPedido AS Dias_Ped_Estr  
  
FROM Pedidos  
  
WHERE (DataDoPedido Between '1/1/95' And '8/31/95')  
  
AND  
  
(PaísDeDestino In ('Brasil', 'Argentina', 'Alemanha', 'Canadá'))  
  
ORDER BY DataDeEntrega-DataDoPedido
```

Exemplo 9: Alterar a instrução SQL anterior, eliminando os critérios para a DataDoPedido e para o PaísDeDestino. Colocar um novo critério para PaísDeDestino, onde sejam exibidos apenas os pedidos cujo PaísDeDestino tem a Primeira Letra na faixa de A até M. Utilize o operador Like.

```
SELECT NúmeroDoPedido, DataDoPedido, DataDeEntrega, PaísDeDestino, Frete,  
DataDeEntrega-DataDoPedido AS Dias_Ped_Estr  
  
FROM Pedidos  
  
WHERE PaísDeDestino Like '[A-M]%'  
  
ORDER BY PaísDeDestino
```

Observe a utilização do Operador Like no critério de Pesquisa para esta consulta. Nunca é demais salientar que a utilização das instruções SQL juntamente com as funções e operadores como o Like, nos fornece um amplo conjunto de possibilidades. O caractere % é utilizado como um caractere curinga. Por exemplo, se especificarmos o seguinte critério:

```
WHERE NomeDoCliente Like 'João%'
```

serão retornados todos os registros para Clientes cujo nome inicia com João. O caractere curinga % significa, em termos simples: “qualquer coisa”, ou seja, o critério especifica que o nome inicie com João, não importando (qualquer coisa) o que vier depois.

Até agora estivemos trabalhando com Instruções que selecionam registros de uma única tabela. Porém é bastante comum criarmos instruções SQL baseadas em duas ou mais tabelas. Quando criamos instruções SQL que buscam dados em duas ou mais tabelas, dizemos que está sendo feito um Join entre as duas tabelas.

Normalmente este “Join” (ou ligação) é feito através de um campo comum às duas tabelas. Por exemplo, NúmeroDoPedido na tabela Pedidos e NúmeroDoPedido na tabela Detalhes do Pedido. Outro exemplo, CódigoDoCliente na tabela Pedidos e CódigoDoCliente na tabela Clientes. Pode acontecer de termos consultas que trabalham com três ou mais Tabelas; neste caso teremos diversos Joins. O número de Joins sempre é igual ao número de tabelas menos um. Por exemplo, se a nossa consulta acessar dados de quatro tabelas, teremos três joins.

IMPORTANTE: Aqui cabe um aviso importante para os usuários do Microsoft Access. No Microsoft Access, o caractere curinga é o *, já no Microsoft SQL Server 2000, utilizamos como caractere curinga o %, o qual também é utilizado na construção de páginas ASP.NET.

Agora passaremos a explorar na Prática, através de exemplos, a construção de Instruções SQL que trabalham com duas ou mais tabelas.

Exemplo 10: Criar uma instrução SQL que selecione os seguintes campos:

- ◆ NúmeroDoPedido da tabela Pedidos
- ◆ DataDoPedido da tabela Pedidos
- ◆ PaísDeDestino da tabela Pedidos
- ◆ Frete da tabela Pedidos
- ◆ CódigoDoProduto da tabela DetalhesdoPedido
- ◆ PreçoUnitário da tabela DetalhesdoPedido
- ◆ Quantidade da tabela DetalhesdoPedido

Além disso, as tabelas Pedidos e Detalhes do Pedido estão relacionadas pelo campo NúmeroDoPedido, através de um relacionamento do tipo Um para Vários.

Classificar a listagem em ordem crescente do Número do Pedido.

Para resolver este exemplo podemos utilizar a seguinte instrução SQL:

```
SELECT Pedidos.NúmeroDoPedido, Pedidos.DataDoPedido, Pedidos.PaísDeDestino,
Pedidos.Frete, DetalhesdoPedido.CódigoDoProduto, DetalhesdoPedido.PreçoUnitário,
DetalhesdoPedido.Quantidade

FROM Pedidos
INNER JOIN
DetalhesdoPedido
ON
Pedidos.NúmeroDoPedido = DetalhesdoPedido.NúmeroDoPedido
ORDER BY Pedidos.NúmeroDoPedido
```

Primeiro devemos observar que, pelo fato de estarmos tratando com dados de duas tabelas, estamos utilizando a nomenclatura completa, isto é, Nome_da_tabela.Nome_do_campo.

Observe a utilização da cláusula INNER JOIN, ligando as tabelas Pedidos e DetalhesdoPedido, através do campo NúmeroDoPedido, conforme especificado na cláusula ON, onde temos o seguinte:

ON

```
Pedidos.NúmeroDoPedido = DetalhesdoPedido.NúmeroDoPedido
```

Observe que esta listagem irá trazer vários registros para cada Pedido, tantos quantos forem os itens de cada pedido.

Mas se, ao invés do CódigoDoProduto, nós quiséssemos que fosse exibida a Descrição do Produto?

Em primeiro lugar esta informação encontra-se na tabela Produtos; logo, teremos que adicionar a Tabela Produtos à nossa consulta, a qual irá se ligar à tabela Detalhes Do Pedido através do campo CódigoDoProduto; logo teremos mais um Join. Para três tabelas teremos dois Joins.

Após adicionar a tabela Produtos e substituir o campo CódigoDoProduto pelo campo NomeDoProduto, a nossa instrução SQL deve ficar conforme indicado no seguinte comando:

```
SELECT Pedidos.NúmeroDoPedido, Pedidos.DataDoPedido, Pedidos.PaísDeDestino,
Pedidos.Frete, Produtos.NomeDoProduto, DetalhesdoPedido.PreçoUnitário,
DetalhesdoPedido.Quantidade

FROM Produtos

INNER JOIN

(Pedidos INNER JOIN DetalhesdoPedido

ON

Pedidos.NúmeroDoPedido = DetalhesdoPedido.NúmeroDoPedido)

ON

Produtos.CódigoDoProduto=DetalhesdoPedido.CódigoDoProduto

ORDER BY Pedidos.NúmeroDoPedido
```

Observe que neste caso temos um INNER JOIN dentro do outro. Dentro do parênteses é feita a ligação entre as tabelas Pedidos e DetalhesdoPedido, através do campo NúmeroDoPedido, e externamente é feita a ligação entre as tabelas Produtos e DetalhesdoPedido, através do campo NúmeroDoPedido.

Podemos utilizar diversos níveis de INNER JOIN, embora esta não seja uma prática recomendada, pois, se aumentarmos muito os níveis de INNER JOIN, posso ter como resultado pesquisas mais lentas em consequência do aumento da complexidade das consultas. Até 3 ou 4 níveis é considerado normal, acima disso preciso repensar a maneira de construir a consulta.

Exemplo 11: Alterar a instrução SQL do item anterior para que somente sejam exibidos os pedidos para os produtos cujo NomeDoProduto inicie com uma letra na faixa de “A” até “J”. Tirar a classificação do campo NúmeroDoPedido e classificar em ordem crescente do campo NomeDoProduto.

```

SELECT Pedidos.NúmeroDoPedido, Pedidos.DataDoPedido, Pedidos.PaísDeDestino,
Pedidos.Frete, Produtos.NomeDoProduto, DetalhesdoPedido.PreçoUnitário,
DetalhesdoPedido.Quantidade

FROM Pedidos

INNER JOIN

(Produtos INNER JOIN DetalhesdoPedido

ON

Produtos.CódigoDoProduto=DetalhesdoPedido.CódigoDoProduto)

ON

Pedidos.NúmeroDoPedido = DetalhesdoPedido.NúmeroDoPedido

WHERE (Produtos.NomeDoProduto Like '[A-J]%' )

ORDER BY Produtos.NomeDoProduto

```

Observe, além dos dois INNER JOIN, a utilização da cláusula WHERE em conjunto com Operador LIKE para especificar o critério desejado.

Até este momento estivemos construindo Instruções SQL que executam consultas no banco de dados. Especificamos quais campos serão exibidos, critérios de filtragem para estes campos e uma ordem de classificação. A partir de agora aprenderemos a utilizar Instruções SQL para a construção de outros tipos de operações, as quais realizam alterações e inclusões em tabelas de banco de dados.

A Instrução UPDATE

A instrução UPDATE é utilizada para alterar informações em um banco de dados. Poderíamos, por exemplo, criar um formulário onde o usuário pode alterar os seus dados cadastrais. Primeiro os dados são recuperados a partir do banco de dados, e em seguida as alterações são enviadas de volta para o banco de dados, através de uma instrução UPDATE. Vamos estudar esta instrução através de exemplos, assim como fizemos com a instrução SELECT.

Exemplo 1: Criar uma instrução SQL que aumenta o PreçoUnitário em 20 % na tabela DetalhesdoPedido, devido a um ajuste na moeda e uma desvalorização em relação ao Dólar.

```

UPDATE DetalhesdoPedido

SET PreçoUnitário = PreçoUnitário*1.2

```

Observe a simplicidade da instrução SQL. Utilizamos uma instrução UPTADE, seguida do nome da tabela onde será feita a atualização. Em seguida uma instrução SET com a expressão de atualização para aumentar em 20 % o PreçoUnitário em todos os registros da tabela DetalhesdoPedido.

Vamos alterar a instrução anterior, para incluir um critério na consulta de atualização.

Exemplo 2: Alterar a instrução SQL do item anterior para que somente sejam aumentados em 20%, o campo PreçoUnitário dos registros cujo PreçoUnitário for maior ou igual a R\$ 20,00.

```
UPDATE DetalhesdoPedido  
SET PreçoUnitário = PreçoUnitário*1.2  
WHERE PreçoUnitário>=20
```

Adicionamos uma cláusula WHERE, para atualizar apenas os registros cujo PreçoUnitário seja maior ou igual a R\$ 20,00.

Exemplo 3: Alterar a instrução SQL do Exemplo anterior para que somente sejam aumentados os registros cujo PreçoUnitário for maior ou igual a R\$ 20,00 e cujo NúmeroDoPedido seja menor do que 10500.

```
UPDATE DetalhesdoPedido  
SET PreçoUnitário = PreçoUnitário*1.2  
WHERE (PreçoUnitário>=20)  
AND  
(NúmeroDoPedido<10500)
```

Observe que utilizamos critérios em dois campos (PreçoUnitário e NúmeroDoPedido) e que estes critérios estão ligados por um operador AND, o que significa que um registro somente será atualizado se ele atender aos dois critérios ao mesmo tempo (PreçoUnitário maior ou igual a 20 e NúmeroDoPedido menor do que 10500). Se o registro atender apenas uma das condições, o registro não terá o seu PreçoUnitário atualizado.

Vamos refinar um pouco mais a nossa consulta de atualização. Vamos fazer com que somente sejam atualizados os Pedidos para o ano da DataDoPedido igual a 1995, e ainda iremos manter os critérios adicionados até agora.

Exemplo 4: Alterar a instrução SQL do Exemplo anterior para incluir um critério para que o Ano da DataDoPedido seja 1995. Você terá que incluir a tabela Pedidos, uma vez que o campo DataDoPedido encontra-se nesta tabela.

```
UPDATE Pedidos  
INNER JOIN  
DetalhesdoPedido  
ON  
Pedidos.NúmeroDoPedido = DetalhesdoPedido.NúmeroDoPedido  
SET DetalhesdoPedido.PreçoUnitário = PreçoUnitário*1.2  
WHERE (DetalhesdoPedido.PreçoUnitário>=20)  
AND  
(DetalhesdoPedido.NúmeroDoPedido<10500)  
AND  
Year(DataDoPedido)=1995
```

Temos diversos detalhes interessantes a observar nesta instrução SQL.

Primeiro um INNER JOIN relacionando as tabelas Pedidos e DetalhesdoPedido, através do campo NúmeroDoPedido. Isso mostra que é perfeitamente possível utilizar um INNER JOIN dentro de uma consulta de

atualização. Segundo, temos a utilização da função Year para extrair apenas o ano do campo DataDoPedido e compará-lo com o critério 1995.

Novamente cabe ressaltar que a utilização de Operadores e Funções nos fornece uma grande flexibilidade em termos de construção de nossas consultas.

A Instrução INSERT

A instrução INSERT é utilizada para adicionar registros em uma tabela de um banco de dados. Por exemplo, quando o usuário preenche os dados de um formulário de cadastro e envia estes dados para o banco de dados, podemos utilizar a instrução INSERT para inserir os dados em uma tabela do banco de dados.

Vamos analisar alguns exemplos.

Exemplo 1: Este exemplo cria um novo registro na tabela Funcionários:

```
INSERT INTO Funcionários (Nome, Sobrenome, Cargo)
VALUES ('Paulo', 'Braga', 'Estagiário')
```

Esta instrução insere um registro na tabela Funcionários com os seguintes dados:

- ◆ Nome: Paulo
- ◆ Sobrenome: Braga
- ◆ Cargo: Estagiário

Exemplo 2: Este exemplo seleciona todos os estagiários de uma tabela Estagiários que tenham sido contratados há mais de 30 dias e adiciona os registros selecionados à tabela Funcionários.

```
INSERT INTO Funcionários
SELECT Estagiários.* FROM Estagiários
WHERE DataDaContratação < Date() - 30
```

Observe que utilizamos a função Date() para capturar a data do Sistema e subtraímos 30, para obter apenas os funcionários contratados há mais do que 30 dias.

A Instrução DELETE

A instrução DELETE é utilizada para excluir registros de um banco de dados. Vamos supor que você tenha desenvolvido uma aplicação Web para agenda eletrônica, onde o usuário pode cadastrar contatos, endereços, telefones e compromissos. Podemos criar uma página ASP.NET que permite ao usuário eliminar dados que não sejam mais necessários na agenda. Neste caso podemos fazer uso da instrução DELETE. Vamos a um exemplo prático.

Exemplo 1: Criar uma instrução SQL que elimine todos os Pedidos da tabela Pedidos cujo PaísDeDestino seja a Alemanha.

```
DELETE Pedidos.PaísDeDestino  
FROM Pedidos  
WHERE Pedidos.PaísDeDestino='Alemanha'
```

A instrução é tão simples que praticamente dispensa comentários. A única recomendação importante é que não devemos utilizar uma instrução DELETE, sem a utilização de uma cláusula WHERE. Utilizar um DELETE sem uma cláusula WHERE significa que estaremos eliminando todos os registros da tabela.

Neste item apresentamos uma breve introdução à linguagem SQL, que no Microsoft SQL Server é chamada de Transact-SQL. De maneira alguma foram apresentadas todas as instruções e comandos disponíveis.

Comandos Avançados da Linguagem T-SQL

Neste tópico vamos tratar de alguns comandos e técnicas avançadas da Linguagem T-SQL. Falaremos sobre os seguintes itens:

- ◆ Pesquisando múltiplas tabelas – detalhes e exemplos.
- ◆ Utilizando subconsultas.

Pesquisando Dados em Múltiplas Tabelas – Detalhes e Exemplos

É comum a situação em que, para construir uma consulta, temos que acessar dados de duas ou mais tabelas. Por exemplo se quisermos uma listagem com o nome do funcionário, o número do pedido, a data do pedido e o país de destino, teremos que utilizar acessar dados das seguintes tabelas do Banco de dados Northwind:

- ◆ Clientes
- ◆ Pedidos
- ◆ Detalhes do Pedido

Para que possamos acessar dados de duas ou mais tabelas, em uma consulta, utilizamos uma cláusula JOIN. Um JOIN é uma operação que nos permite acessar dados de duas ou mais tabelas. O JOIN é efetuado com base em uma coluna que seja comum a duas ou mais tabelas. Por exemplo, se fizermos um JOIN entre as tabelas Pedidos e Detalhes do Pedido, o JOIN será efetuado através da coluna NúmeroDoPedido, que é a coluna que relaciona as duas tabelas. Na tabela Detalhes do Pedido, o campo NúmeroDoPedido é utilizado para indicar a qual pedido pertence um determinado item.

Vamos a um exemplo de utilização do JOIN.

Exemplo: Escrever um comando SQL que retorne os campos indicados na Tabela III.2.

Tabela III.2 Campos – Tabelas

Campo	Tabela
NúmeroDoPedido	Pedidos
DataDoPedido	Pedidos
CódigoDoProduto	Detalhes do Pedido
PreçoUnitário	Detalhes do Pedido
Quantidade	Detalhes do Pedido

Esta listagem irá retornar uma linha para cada item de cada pedido, com os campos indicados na Tabela III.2. Por exemplo, se um campo tiver 5 itens serão retornadas 5 linhas, com o mesmo valor para o campo NúmeroDoPedido e OrderDate e diferentes valores para os demais campos. Para construir a listagem solicitada, utilizamos o seguinte comando:

```

SELECT      [Pedidos].[NúmeroDoPedido], [Pedidos].[OrderDate],
            [Detalhes do Pedido].[ProductID],
            [Detalhes do Pedido].[UnitPrice], [Detalhes do Pedido].[Quantity]
FROM        [Detalhes do Pedido]
JOIN
            [Pedidos]
ON
            [Detalhes do Pedido].[NúmeroDoPedido] = [Pedidos].[NúmeroDoPedido]

```

Observe que estamos utilizando a sintaxe [NomeDaTabela].[NomeDoCampo].

O resultado parcial é indicado na listagem a seguir:

NúmeroDoPedido	DataDoPedido	Código	PreçoUnitário
10248	4/7/1996	11	14 12
10248	4/7/1996	42	9,8 10
10248	4/7/1996	72	34,8 5
10249	5/7/1996	14	18,6 9
10249	5/7/1996	51	42,4 40
10250	8/7/1996	41	7,7 10
10250	8/7/1996	51	42,4 35
10250	8/7/1996	65	16,8 15
...

Observe que, para cada pedido, aparecem tantas linhas quantas forem os itens do pedido.

Vamos analisar a sintaxe básica do JOIN.

```
SELECT Campo1, Campo2, ..., Campon  
FROM     Tabela1  
JOIN  
Tabela2  
ON  
[Tabela1].[CampoComun]= [Tabela2].[CampoComun]
```

Além do JOIN ligando as duas tabelas, devemos especificar qual o campo em comum nas duas tabelas. Isto é feito pela cláusula ON.

Tipos de JOIN

Existem três tipos de JOIN, conforme listado a seguir:

- ◆ INNER JOIN: Este é o JOIN padrão, isto é, se não especificarmos o tipo de JOIN a ser utilizado, será utilizado um INNER JOIN. A característica do INNER JOIN é que somente são retornados os registros que têm valores coincidentes nas duas ou mais tabelas. Por exemplo, se tivermos um INNER JOIN ligando as tabelas Clientes e Pedidos, somente serão retornados os registros para os clientes que têm algum pedido na tabela Pedidos. Clientes que não efetuaram pedidos (portanto não têm registros coincidentes na tabela Pedidos) não farão parte do resultado da operação INNER JOIN. A seguir um exemplo de utilização de um INNER JOIN, entre as tabelas Clientes e Pedidos:

```
SELECT  [Clientes].[CódigoDoCliente], [Clientes].[NomeDaEmpresa],  
        [Clientes].[Cidade], [Pedidos].[NúmeroDoPedido],  
        [Pedidos].[DataDoPedido]  
FROM    [Clientes]  
INNER JOIN  
        [Pedidos]  
ON  
        [Clientes].[CódigoDoCliente]= [Pedidos].[CódigoDoCliente]
```

Este comando retorna todos os clientes que têm pedidos e os respectivos pedidos. Se um cliente tiver 10 pedidos, retornarão 10 registros para este cliente, um para cada pedido. Clientes que não têm pedido, não aparecerão na listagem.

- ◆ LEFT OUTER JOIN: Este tipo retorna todos os registros da primeira tabela e os registros relacionados da segunda tabela. No nosso exemplo, na listagem anterior, se trocarmos o INNER JOIN por um LEFT OUTER JOIN, serão retornados todos os Clientes e os pedidos relacionados, e também os clientes que não possuem pedidos. Para aqueles clientes que não possuem pedidos, os campos NúmeroDoPedido e DataDoPedido terão o valor Null. A seguir temos o comando que utiliza um LEFT OUTER JOIN:

```

SELECT      [Clientes].[CódigoDoCliente], [Clientes].[NomeDaEmpresa],
            [Clientes].[Cidade], [Pedidos].[NúmeroDoPedido],
            [Pedidos].[DataDoPedido]
FROM        [Clientes]
LEFT OUTER JOIN
            [Pedidos]
ON
[Clientes].[CódigoDoCliente]= [Pedidos].[CódigoDoCliente]
ORDER BY [Clientes].[CódigoDoCliente]

```

Na Figura III.1 podemos observar que foram retornados inclusive os clientes que não possuem pedidos, o que é indicado pelos valores NULL nos campos NúmeroDoPedido e OrderDate.

	CustomerID	CompanyName	City	OrderID	OrderDate
1	ALFKI	Alfreds Futterkiste	Berlin	NULL	NULL
2	ANATR	Ana Trujillo Empared...	México D.F.	NULL	NULL
3	ANTON	Antonio Moreno Taquería	México D.F.	NULL	NULL
4	AROUT	Around the Horn	London	NULL	NULL
5	BERGS	Berglunds snabbköp	Luleå	10278	1996-08-12
6	BERGS	Berglunds snabbköp	Luleå	10280	1996-08-14
7	BERGS	Berglunds snabbköp	Luleå	10384	1996-12-16
8	BERGS	Berglunds snabbköp	Luleå	10444	1997-02-12

Figura III.1: Utilizando LEFT OUTER JOIN.

- ♦ **RIGHT OUTER JOIN:** Este tipo retorna todos os registros da segunda tabela e os registros relacionados da primeira tabela. Um exemplo de RIGHT OUTER JOIN:

```

SELECT      [Clientes].[CódigoDoCliente],
            [Clientes].[NomeDaEmpresa],
            [Clientes].[Cidade], [Pedidos].[NúmeroDoPedido],
            [Pedidos].[DataDoPedido]
FROM        [Clientes]
RIGHT OUTER JOIN
            [Pedidos]
ON
[Clientes].[CódigoDoCliente]=
[Pedidos].[CódigoDoCliente]
ORDER BY [Clientes].[CódigoDoCliente]

```

IMPORTANTE: Se você utilizar o Banco de dados Northwind que vem com o SQL Server 2000, serão retornados todos os clientes, pois existem pedidos para todos. Para poder exemplificar neste exercício, eu exclui os pedidos para os clientes com os seguintes códigos: ALFKI, ANATR, ANTON, AROUT.

Na Figura III.2 podemos observar que foram retornados todos os pedidos da tabela pedidos, mas apenas os clientes que possuem pedidos, isto é, todos os registros da segunda tabela (Pedidos) e apenas os registros relacionados da primeira tabela (Clientes).

The screenshot shows the SQL Query Analyzer interface. The query window displays the following T-SQL code:

```

SELECT      [Customers].[CustomerID], [Customers].[CompanyName],
            [Customers].[City], [Orders].[OrderID], [Orders].[OrderDate]
FROM        [Customers] RIGHT OUTER JOIN
            [Orders]
ON         [Customers].[CustomerID] = [Orders].[CustomerID]
ORDER BY   [Customers].[CustomerID]
  
```

The results grid shows the following data:

	CustomerID	CompanyName	City	OrderID	OrderDate
1	ALFKI	Alfreds Futterkiste	Berlin	10643	1997-08-25
2	ALFKI	Alfreds Futterkiste	Berlin	10692	1997-10-03
3	ALFKI	Alfreds Futterkiste	Berlin	10702	1997-10-13
4	ALFKI	Alfreds Futterkiste	Berlin	10835	1998-01-15
5	ALFKI	Alfreds Futterkiste	Berlin	10952	1998-03-16
6	ALFKI	Alfreds Futterkiste	Berlin	11011	1998-04-09

Figura III.2: Utilizando RIGHT OUTER JOIN.

JOIN com Mais do que Duas Tabelas

Também podemos fazer um JOIN com mais do que duas tabelas. Sempre o número de JOINs, será igual ao número de tabelas menos um. Por exemplo, se estivermos relacionando 4 tabelas, teremos três JOINs; se estivermos relacionando três tabelas, teremos dois JOINs, e assim por diante.

Exemplo: Criar uma listagem com os campos indicados na Tabela III.3

Tabela III.3 Campos – Tabelas

Campo	Tabela
CódigoDoCliente	Clientes
NomeDaEmpresa	Clientes
NúmeroDoPedido	Pedidos
DataDoPedido	Pedidos
TotalPedido	$([Quantidade]*[PreçoUnitário])*(1-[Desconto])$

Teremos uma listagem com o código e o nome do cliente, o número, a data e o total do pedido, sendo que o total é calculado a partir de dados da tabela Detalhes do Pedido. Com isso estamos acessando campos de três tabelas: Clientes, Pedidos e Detalhes do Pedido. Três tabelas = dois joins.

A seguir temos o comando que retorna a listagem solicitada:

```

SELECT      [Clientes].[CódigoDoCliente], [Clientes].[NomeDaEmpresa],
            [Pedidos].[NúmeroDoPedido], [Pedidos].[OrderDate],
            SUM(([Detalhes do Pedido].[Quantidade] * [Detalhes do
            Pedido].[PreçoUnitário])
            * (1 - dbo.[Detalhes do Pedido].Desconto)) AS TotalPedido
FROM [Detalhes do Pedido]
INNER JOIN
    [Pedidos]
ON
[Detalhes do Pedido].[NúmeroDoPedido] = [Pedidos].[NúmeroDoPedido]
INNER JOIN
    [Clientes]
ON
[Pedidos].[CódigoDoCliente] = [Clientes].[CódigoDoCliente]
GROUP BY
    [Clientes].[CódigoDoCliente], [Clientes].[NomeDaEmpresa],
    [Pedidos].[NúmeroDoPedido], [Pedidos].[OrderDate]
ORDER BY
    [Clientes].[CódigoDoCliente]

```

Na listagem a seguir, temos a parte inicial do resultado retornado:

Código	NomeDaEmpresa	Número	OrderDate	TotalPedido
BERGS	Berglunds snabbköp	10278	12/8/1996	1.488,80
BERGS	Berglunds snabbköp	10280	14/8/1996	613,20
BERGS	Berglunds snabbköp	10384	16/12/1996	2.222,40
BERGS	Berglunds snabbköp	10444	12/2/1997	1.031,70
BERGS	Berglunds snabbköp	10445	13/2/1997	174,90
BERGS	Berglunds snabbköp	10524	1/5/1997	3.192,65
...

Vamos fazer alguns comentários sobre o comando anterior.

Primeiro vamos considerar o trecho onde é feito o JOIN entre as três tabelas. Na verdade o JOIN é feito sempre entre duas tabelas, dizemos duas a duas, conforme indicado no trecho a seguir:

```

FROM [Detalhes do Pedido]
INNER JOIN
    [Pedidos]

```

ON

[Detalhes do Pedido]. [NúmeroDoPedido] = [Pedidos]. [NúmeroDoPedido]

INNER JOIN

[Clientes]

ON

[Pedidos]. [CódigoDoCliente] = [Clientes]. [CódigoDoCliente]

Primeiro fizemos o JOIN entre as tabelas [Detalhes do Pedido] e [Pedidos], através do campo [NúmeroDoPedido]. Depois é feito o JOIN entre as tabelas [Pedidos] e [Clientes], através do campo CódigoDoCliente.

Também utilizamos GROUP BY, para agrupar os diversos itens de cada pedido e a função SUM, para calcular a soma de todos os itens do pedido. No final ordenamos pelo código do cliente – [CódigoDoCliente].

Utilizando Subconsultas

Uma subconsulta é um comando SELECT dentro de outro comando SELECT, ou seja, uma consulta dentro da outra. Uma consulta pode ser utilizada em qualquer local onde uma expressão seja aceita.

Em alguns casos, podemos obter os mesmos resultados utilizando JOINs ao invés de subconsultas. Um detalhe para o qual devemos estar atentos quando utilizamos subconsultas é em relação ao desempenho. De uma forma geral, devemos utilizar subconsultas quando quisermos dividir uma consulta complexa, em uma série de passos mais simples, mas que combinados geram o resultado da consulta complexa.

Alguns detalhes a serem considerados quando utilizamos subconsultas:

- ◆ Subconsultas devem estar dentro de parênteses.
- ◆ Podemos ter diversos níveis de subconsultas, isto é, uma consulta dentro da outra. Porém mais uma vez quero chamar atenção para a questão do desempenho.

Para vermos as subconsultas em ação, vamos a um exemplo.

Exemplo: Criar uma listagem onde são exibidos os campos NúmeroDoPedido da tabela Pedidos, o campo DataDoPedido da tabela Pedidos e o maior preço unitário dos itens que fazem parte do pedido. Para criar esta consulta, utilizamos o seguinte comando:

```
SELECT [Pedidos]. [NúmeroDoPedido], [Pedidos]. [DataDoPedido],
       (SELECT MAX([Detalhes do Pedido]. [PreçoUnitário])
        FROM [Detalhes do Pedido]
        WHERE [Pedidos]. [NúmeroDoPedido] = [Detalhes do Pedido]. [NúmeroDoPedido])
AS MaiorPreçoUnitário
FROM [Pedidos]
```

A seguir temos os registros iniciais retornados pela consulta anterior:

Número	OrderDate	MaiorPreçoUnitário
10248	1996-07-04	348.000
10249	1996-07-05	424.000
10250	1996-07-08	424.000
10251	1996-07-08	168.000
10252	1996-07-09	648.000
10253	1996-07-10	160.000
...

Observe a subconsulta

```
(SELECT MAX([Detalhes do Pedido].[PreçoUnitário]) FROM [Detalhes do Pedido]
 WHERE [Pedidos].[NúmeroDoPedido] = [Detalhes do Pedido].[NúmeroDoPedido]) AS
MaiorPreçoUnitário
```

Esta consulta pesquisa os diversos itens para cada pedido e retorna o maior preço unitário.

Outro uso típico de uma subconsulta é como parâmetro da filtragem da consulta. Considere o comando a seguir:

```
SELECT NúmeroDoPedido, CódigoDoCliente, DataDoPedido As MaisRecente
FROM Pedidos
WHERE DataDoPedido=(Select MAX(DataDoPedido) FROM Pedidos)
```

Este comando retorna os pedidos mais recentes, ou seja, para o valor máximo da data. O valor máximo da data significa a data mais recente no banco de dados. O resultado desta consulta é o seguinte:

Número	Código	MaisRecente
11077	RATTC	1998-05-06
11076	BONAP	1998-05-06
11075	RICSU	1998-05-06
11074	SIMOB	1998-05-06

A subconsulta determina a data mais recente, data esta que é utilizada como critérios para a cláusula WHERE. Em resumo, o resultado retornado pela subconsulta é passado para a cláusula WHERE.

Se trocarmos o MAX por MIN, teremos uma listagem dos pedidos mais antigos. Execute o seguinte comando:

```
SELECT NúmeroDoPedido, CódigoDoCliente, DataDoPedido As MaisAntigo
FROM Pedidos
WHERE DataDoPedido=(Select MIN(DataDoPedido) FROM Pedidos)
```

Com este comando obtemos o seguinte resultado:

Número	Código	MaisRecente
10248	VINET	1996-07-04

Temos um tipo especial de subconsulta chamada de “Subconsulta correlacionada – Correlated subqueries”. Neste tipo de subconsulta a subconsulta utiliza informações da consulta principal e é executada para cada linha retornada pela consulta externa. Neste tipo de subconsulta é que temos que tomar cuidados com o desempenho. Por exemplo, se a consulta principal retorna 5000 registros, a subconsulta será executada 5000 vezes.

Mais uma vez vamos utilizar um exemplo para ilustrar este conceito.

Exemplo: Criar uma consulta que liste o nome de todos os funcionários que fizeram pedidos para o Brazil, em que o campo DataDoPedido tem o ano de 1997. Para obter esta listagem, utilizamos o seguinte comando:

```
SELECT  
    [Funcionários].[Nome] + ' ' + [Funcionários].[Sobrenome] AS NomeCompleto,  
    CódigoDoFuncionário FROM Employees  
  
WHERE EXISTS  
  
(SELECT * FROM [Pedidos]  
  
    WHERE  
  
        [Funcionários].[CódigoDoFuncionário]=[Pedidos].[CódigoDoFuncionário]  
        AND Year([DataDoPedido])='1997')  
  
ORDER BY [Funcionários].[Nome]
```

Vamos explicar o funcionamento desta subconsulta em quatro passos:

1. Os valores obtidos pela consulta externa (primeiro SELECT) são passados para a consulta interna. Lembre que para cada registro obtido pela consulta externa, a consulta interna é executada.
2. A consulta interna (segundo SELECT) utiliza o valor passado pela consulta externa e executa o comando SELECT da subconsulta.
3. Após a execução, a subconsulta retorna o seu resultado para a consulta principal.
4. Com base no valor retornado pela subconsulta, em conjunto com a função EXISTS, o registro retornado pela consulta externa é incluído ou descartado do resultado final. O próximo registro retornado pela consulta externa é passado para a consulta interna, a qual executa novamente. E o processo continua, até que todos os registros da consulta principal tenham sido passados para a consulta interna. Desta maneira vemos que a consulta interna é executada tantas vezes, quantos forem os registros retornados pela consulta externa.

A seguir temos os resultados do comando deste exemplo:

NomeCompleto	Código
Andrew Fuller	2
Anne Dodsworth	9
Janet Leverling	3
Laura Callahan	8
Margaret Peacock	4
Michael Suyama	6

Nancy Davolio	1
Robert King	7
Steven Buchanan	5

A correta utilização de subconsultas pode facilitar a obtenção de resultados aparentemente complexos de se obter com um único comando SELECT. Porém, mais uma vez, vou ressaltar o cuidado em relação ao desempenho, quando tratamos com subconsultas.

Conclusão

Neste anexo apresentamos uma noção básica da Linguagem SQL.

Aprendemos a utilizar diversos elementos da linguagem, como por exemplo:

- ◆ A instrução Select e as suas diversas cláusulas
- ◆ A instrução Update
- ◆ A instrução Insert INTO
- ◆ A instrução Delete
- ◆ O operador Like
- ◆ As cláusulas WHERE e ORDER BY
- ◆ O operador In
- ◆ Os operadores AND e OR

Também apresentamos alguns conceitos avançados do SQL, como por exemplo:

- ◆ Utilização de JOIN
- ◆ Subconsultas

Introdução

Neste anexo apresento uma lista de sites com informações sobre .NET, C# e ASP.NET. Nestes sites você encontrará informações tais como:

- ◆ Artigos.
- ◆ Exemplos de código.
- ◆ Componentes e Web Services gratuitos.
- ◆ Componentes e Web Services pagos.

Na Tabela IV.1 temos a lista de sites indicados.

Tabela IV.1 Sites sobre ASP.NET.

Site	Descrição
msdn.microsoft.com/net	Site oficial da Microsoft sobre o .NET. Aqui você encontra link para fazer o Download do Framework .NET, link para encomendar os CDs de avaliação do Visual Studio .NET. Também está disponível a documentação atualizada para todas as linguagens e tecnologias do .NET. É, sem dúvida, referência obrigatória para quem está iniciando os estudos de .NET. Eu recomendo.
www.asp.net	Um portal de informações sobre ASP.NET, C# e tecnologias .NET em geral. Contém links para outros excelentes sites. Referência obrigatória. Simplesmente fantástico. Eu recomendo.
www.123aspx.com	Excelente site, com links para artigos, classificados por categoria: Banco de dados, segurança, Web Services, etc. Contém links para diversos sites especializados em ASP.NET. Eu recomendo.
www.aspnextgen.com	Ótimo site com lições gratuitas sobre os aspectos básicos do ASP.NET.
www.asptoday.com	Site pago. A assinatura anual custa U\$ 50,00. Contém uma grande quantidade de artigos e tutoriais sobre ASP.NET.
www.aspfree.com	Link para artigos, sites, componentes. Links para assuntos diversos como SOAP, PHP, XML, C# e .NET em geral.
www.aspnextgen.com	Ótimo site com lições gratuitas sobre os aspectos básicos do ASP.NET.

ANEXO

4

Sites Sobre ASP.NET, C# e XML

Site	Descrição
www.4guysfromrolla.com	Um grande número de artigos com o código disponível. Links para outros sites. Trata de assuntos tais como: ASP, ASP.NET, C#. Fantástico, eu recomendo.
www.411asp.net	O mais completo portal sobre ASP.NET. Centenas de links divididos por categorias: Aplicações, Exemplos e Tutoriais, Códigos, Componentes, Comunidades, Hospedagem com suporte a ASP.NET, etc. Excelente, eu recomendo.
www.developersdex.com	Bom site com informações sobre ASP, ASP.NET, C# (melhor área do site), XML, SOAP, ADSI e VB.
www.dotnetexperts.com	Uma comunidade de especialistas em .NET, onde são publicados artigos sobre ASP.NET, XML, SOAP, C#, além de links para diversos outros sites relacionados.
www.plusasp.com	Site interessante com diversos exemplos práticos. Código dos exemplos disponível. Explicações detalhadas sobre cada exemplo.
www.superexpert.com	Site com tutoriais, artigos e links sobre ASP, ASP.NET e Web Services.
www.superexpert.com	Excelente site. Com muitos artigos, links e informações sobre: ASP, ASP.NET, PHP, Java, Java Script e CGI. Eu recomendo.
www.aspx101.net	Maravilhoso. Artigos, Revistas, links, tutoriais. Tudo muito bem organizado e fácil de localizar. Destaque para informações sobre Web Services. Eu recomendo.
www.asp101.com/aspplus	Bom site. Com links para artigos divididos por categoria. Também contém links para outros sites sobre ASP.NET.
www.aspwire.com	Bom site. Com artigos, dicas e tutoriais. Bom design e navegação intuitiva.
www.15seconds.com/focus/.NET.htm	Lista com diversos artigos gratuitos sobre ASP.NET, XML, Web Services, ADSI e outros assuntos relacionados.
www.devasp.net/net	Site simplesmente fantástico. Links para artigos, tutoriais, código gratuito. Trata de VB.NET, ASP.NET, C#, XML, SOAP, Download de Componentes. Um dos melhores sites de .NET que eu conheço. Eu recomendo.
www.aspnetpro.com	Revista sobre ASP.NET com links para grupos de discussão.

Na Tabela IV.2 temos a lista de sites indicados para C#.

Tabela IV.2 Sites sobre C#.

Site	Descrição
www.csharp.net	Excelente portal sobre C#. Links para artigos, tutoriais, códigos gratuitos e outros sites com informações sobre C#. Excelente site. Eu recomendo.

Site	Descrição
www.csharptoday.com	Site pago. A assinatura anual custa U\$ 50,00. Contém uma grande quantidade de artigos e tutoriais sobre C#.
www.mastercsharp.com	Bom site. Com artigos, e links para C#, ADO.NET, VB.NET e demais tecnologias .NET.
www.csharpindex.com	Excelente. Tutoriais, links para outros sites sobre C#. Exemplos divididos por categorias, componentes, Web Services, XML, SOAP e .NET em geral. Eu recomendo.
www.devdex.com/csharp	Excelente. Tutoriais, links para outros sites sobre C#. Exemplos divididos por categorias, componentes, Web Services, XML, SOAP e .NET em geral. Eu recomendo.
www.csharphelp.com	Ótimo. Suporte a dúvidas, código, exemplos, tutoriais, Base de Conhecimentos sobre C#, links para outros sites e novidades sobre .NET. Eu recomendo.
www.c-sharpcorner.com	Ótimo. Site gratuito para desenvolvedores .NET que utilizam C#. Contém artigos, código-fonte, tutoriais e grupos de discussão.
www.csharp-station.com	Contém links e tutoriais. Bom site.

Outros sites sobre C#

- ◆ www.csharp.com
- ◆ www.csharpfree.com
- ◆ www.csharpgoodies.com
- ◆ www.c-sharpcenter.com
- ◆ www.easycsharp.com/
- ◆ pages.zoom.co.uk/seesharp/
- ◆ ManyQuestions.com
- ◆ www.hitmill.com

Na Tabela IV.3 temos a lista de sites indicados para XML.

Tabela IV.3 Sites sobre XML.

Site	Descrição
www.xml.org	Excelente portal sobre XML. Links para artigos, tutoriais e outros sites sobre XML. Eu recomendo.
www.vbxm.com	Bom site. Links, artigos, tutoriais e muita informação para programadores VB e ASP/ASP.NET, que trabalham com XML.
www.wdvl.com	Excelente site sobre uma variedade de tecnologias para desenvolvimento Web. Excelentes artigos e tutoriais sobre XML.

Site	Descrição
www.hotwired.com/webmonkey/xml/	Links, artigos e ótimos tutoriais. Ideal para quem está iniciando no estudo do XML.
www.w3.org/XML	Site com a especificação oficial para o XML. Eu recomendo.
www.xml.com	Um portal com grande quantidade de informações sobre XML. Excelente. Eu recomendo.
www.xmlpitstop.com	Outro excelente portal com grande quantidade de informações sobre XML. Eu recomendo.
msdn.microsoft.com/xml/default.asp	Site da Microsoft sobre XML. Eu recomendo.

Conclusão

Com o conteúdo apresentado neste livro, mais as referências deste Anexo, o leitor tem condições de entender este novo conceito que está agitando a indústria de desenvolvimento de Software: .NET.

Se você tem sugestões sobre assuntos que gostaria de ver em futuras edições deste livro, ou publicados no site da editora, por favor entre em contato pelo e-mail: webmaster@juliobattisti.com.br

O código com os exemplos deste livro, está disponível para download no seguinte endereço: www.axcel.com.br.

No site do autor você encontrará artigos sobre ASP.NET, C#, Certificações Microsoft, Matemática para Concursos e cursos gratuitos: www.juliobattisti.com.br

Um bom estudo a todos.