

ASP.NET 4.0

Avançado

TMSASP35AV
Novembro/2010

Apostila desenvolvida especialmente para a Target Informática Ltda.

Sua cópia ou reprodução é expressamente proibida.

Novembro/2010

Sumário

1. Segurança.....	1
Objetivos	2
Conceitos de Segurança.....	3
Habilitando o site para utilizar o gerenciamento de segurança do Asp.Net 4.0.....	5
Entendendo o gerenciamento da segurança	6
Criando Usuários e explorando a estrutura criada	8
Validando usuários com o Login	11
Utilizando o ChangePassword para trocar as senhas	12
Entendendo e utilizando os controles LoginStatus, LoginName e LoginView ...	13
Gerenciamento Manual de usuários	17
Classes Membership e MembershipUser	18
Utilizando Grupos	19
Exercícios	23
Espaço para Anotações.....	24
 2. Globalização e Localização.....	 25
Objetivos	26
O que é globalização e localização?	27
Localizando valores em uma aplicação asp.Net	28
Traduzindo interfaces do usuário com o asp.net	32
Exercícios	38
Espaço para Anotações.....	39
 3. Utilizando Componentes no ASP.NET	 40
Objetivos	41
O que são componentes?	42
Utilizando uma DLL externa	43
Private Assemblies e Shared Assemblies.....	46
Analizando as referências	46
Espaço para Anotações.....	48
 4. Web Parts	 50
Objetivos	51
O que são web parts ?	52
A arquitetura de Web Parts	53
O WebPartManager e as WebZones.....	53
Utilizando as Web Parts	55
Construindo Web Parts.....	61
Exercícios	67
Espaço para Anotações.....	68
 5. LINQ	 69
Objetivos	70
O que é o LINQ	71

LINQ TO OBJECTS	73
LINQ to SQL.....	78
Entendendo o DataContext	78
Utilizando LINQ to SQL em nossa aplicação.....	79
Utilizando LINQ to SQL em 3 camadas	86
Exercícios	97
Espaço para Anotações.....	98
 6. Manutenção de Estado	99
Session	101
HiddenField.....	104
ViewState.....	104
QueryString	105
Cookies	106
Application.....	106
Exercícios	109
Espaço para Anotações.....	110
 7. Web Services	111
Objetivos	112
Web Services.....	113
Criando e utilizando um Web Service	115
Chamando um XML Web Service por HTTP	117
Utilizando um WebService externo.....	119
Exercícios	123
Espaço para Anotações.....	124
 8. Utilizando AJAX no ASP.NET	125
Objetivos	126
O que é AJAX ?	127
Por que usar AJAX no ASP.NET?	127
Utilizando Ajax em uma aplicação Web	127
O controle ScriptManager.....	129
O controle UpdatePanel.....	130
O Controle UpdateProgress	131
O Controle Timer	132
Triggers	133
Inclusão de Scripts via ScriptManager	134
AJAX Control Toolkit	136
Utilizando controles do Ajax Control Toolkit	138
ValidatorCalloutExtender	138
CollapsiblePanelExtender	140
PopupControlExtender.....	141
Espaço para anotações	144
 9. Reflection	145
Objetivos	146

O que é Reflection?.....	147
Funcionalidades e ganhos de utilizar reflexão em nossa aplicação	147
O tipo Assembly.....	147
O tipo AssemblyName.....	148
O tipo Module.....	149
Trabalhando com Tipos.....	150
Recuperando um objeto Type	150
Utilizando Reflection em nossa aplicação.....	151
Espaço para anotações	156
10. Distribuindo sua Aplicação.....	157
Objetivos	158
Distribuindo sua Aplicação ASP.NET.....	159
Publicando sua aplicação	160
Espaço para Anotações.....	162

1. Segurança

Objetivos

Ao final deste capítulo você estará apto à:

- Conhecer o sistema de segurança de sites asp.net implementados pelo .net framework 4.0.
- Entender as diferenças entre as principais formas de autenticação disponíveis no asp.net.
- Utilizar os controles de segurança do asp.net.
- Gerenciar grupos de usuários e atribuir características de permissão aos grupos.

Conceitos de Segurança

Garantir a segurança de sites é uma questão complexa e crítica para os desenvolvedores de web sites. Para garantir a segurança é necessário um planejamento cuidadoso, tanto os administradores do web site como os programadores devem ter um claro entendimento das opções de segurança disponíveis.

No asp.net clássico e em outras linguagens de programação todo o controle de segurança e de acesso aos sites, bem como o gerenciamento de usuários e seus respectivos privilégios de acessos, ficam atrelados ao trabalho do desenvolvedor responsável por criar um mecanismo que gerencie toda essa informação e esse controle.

O Asp.net 4.0 trabalha em conjunto com o Microsoft .Net Framework 4.0 e o Microsoft Internet Information Service (IIS) para ajudar a fornecer segurança em nossas aplicações Web. Desde a versão do 1.1 do .Net Framework já existia uma implementação para o gerenciamento de segurança de nossas aplicações web, mas muitas coisas evoluíram.

Na versão 4.0 do .net framework temos duas funções principais de segurança:

- Authorization.
- Authentication.

A função Authorization Limita direitos de acesso, por permitir e negar permissões, para uma identidade autenticada e pré-estabelecida

A função Authentication ajuda a verificar se o usuário é realmente quem ele diz ser. A aplicação obtém credenciais de um usuário, como nome e senha, e validades as credenciais contra alguma autoridade. Se as credencias forem validadas então o usuário realmente é quem ele diz ser.

Geralmente a função authorization é utilizada para identificar quem tem o privilégio de acessar determinados arquivos via web. Já a função Authentication, por ser mais dinâmica é amplamente utilizada como o método de gerenciar o acesso de usuários em nossas aplicações web. Em função de ser mais utilizada, vamos direcionar nossos estudos sobre a função authentication.

O Asp.Net implementa a authentication através de authentication providers, que contém o código necessário para autenticar as credencias do usuário solicitante. Junto com o asp.net temos dois authentication providers pré implementados.

- Windows Authentication Provider.

Segurança

- Forms Authentication Provider.

A autenticação baseada no Windows Authentication Provider utiliza o usuário windows corrente no IIS como o usuário autenticado em uma aplicação Asp.net. Ela é bem utilizada para aplicativos asp.net do tipo Intranet para aproveitar as credencias do usuário para ter permissão em determinados diretórios.

A autenticação baseada no Forms Authentication Provider permite autenticar usuários usando seu próprio código e depois manter um token de autenticação em um cookie ou no URL da página.

Para usar a autenticação baseada em formulários você cria uma pagina de login que irá coletar as credencias dos usuários e deve incluir código para validar as credencias. Normalmente a solução asp.net é configurada para direcionar á pagina de login sempre que o usuário tentar entrar em uma página que só seja disponibilizada após o usuário ser autenticado. Após a autenticação, se o usuário for um usuário com permissão então ele é direcionado para a página que ele tentou entrar antes, senão ele tem o acesso barrado até que ele se autentique.

O modelo de autenticação por formulários (Forms Authentication) é amplamente utilizado em desenvolvimentos de aplicações web.

Uma conveniente maneira para trabalhar com Forms Authentication é utilizar Asp.Net MemberShip e Asp.Net Login Controls. O Asp.Net MemberShip permite armazenar e gerenciar informações de usuários e inclui métodos para autenticar usuários.O Asp.Login Controls são controles fornecidos pelo Asp.Net que trabalham em conjunto com o Asp.Net MemberShip. Os controles encapsulam a lógica para solicitar aos usuários credenciais, validar os usuários, recuperar senhas e outras ações vinculadas com o gerenciamento de usuários. O resultado disso é que o Asp.Net MemberShip e os Asp.Net Login Controls fornecem uma camada de abstração sobre a autenticação por formulários. Esses recursos substituem a maior parte ou todo o trabalho que você normalmente teria que fazer para utilizar a autenticação por formulários.

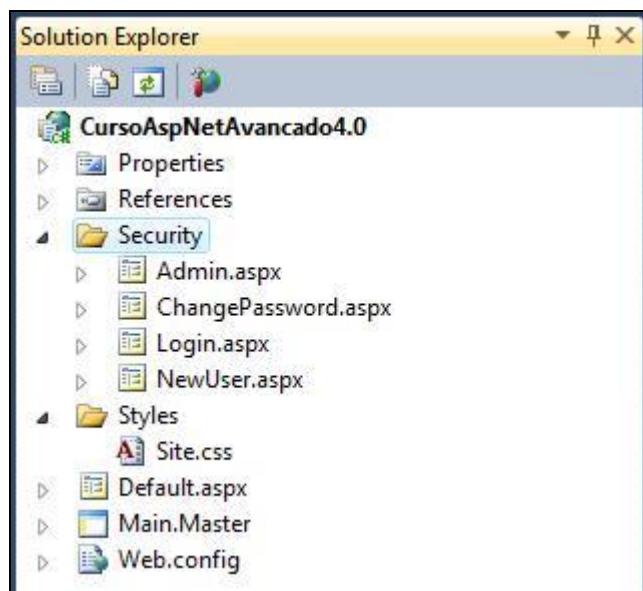
Habilitando o site para utilizar o gerenciamento de segurança do Asp.Net 4.0

Ao criarmos um projeto no Visual Studio 2010 podemos utilizar o template “Asp.Net Web Application”. Esse template irá criar uma aplicação asp.net com as páginas responsáveis por implementar a segurança dentro da pasta chamada Account. O projeto já estará configurado para utilizarmos o gerenciamento de segurança do asp.net.

Mas para entendermos como implementar de uma forma eficiente a autenticação baseada em formulários, a aplicação base de nosso curso é baseada no template “Asp.Net Empty Web Application” que é um template que não cria nenhuma página pré definida em nossa aplicação. Esse template foi utilizado para que nós mesmos possamos estabelecer a estrutura de gerenciamento da segurança de nossa aplicação.

Vamos criar uma pasta chamada “Security” em nossa aplicação e vamos inserir 4 web forms dentro dessa pasta: Admin, Login, NewUser e ChangePassword.

A estrutura deve ficar conforme a figura abaixo.



Após termos criado essa estrutura, com a pagina Default.aspx setada como pagina inicial, vamos rodar nossa aplicação e note que podemos navegar livremente em nossa aplicação.

Para que o nosso site utilize a autenticação por formulários é necessário que seja setada uma configuração no arquivo web.config de nossa aplicação asp.net.

Segurança

Devemos configurar o web.config da seguinte maneira:

```
<authentication mode="Forms">
  <forms name=".ASPXAUTH" loginUrl("~/Security/Login.aspx" defaultUrl="Default.aspx" ></forms>
</authentication>
```

Essa configuração deve ser inserida dentro da Tag <system.web> do arquivo web.config. Nesta configuração informamos que o modo de autenticação será "Forms" e que o cookie de autenticação será ".ASPXAUTH"(o cookie padrão de autenticação do asp.net). Setamos também o formulário responsável pela autenticação do usuário e o formulário default de nossa aplicação.

Agora vamos rodar nossa aplicação, com qualquer página setada como página (a pagina Admin.aspx por exemplo). Note que continuamos navegando livremente por qualquer página de nossa aplicação web. Isso ocorre porque nós definimos o método de autenticação de nosso site, mas não restringimos ainda o acesso ao site.

Para que possamos restringir o acesso a nossa aplicação devemos adicionar a tag `authorization` no web.config logo abaixo da tag `authentication`, dentro da tag principal <system.web>. Dentro dessa tag é configurado o nível de autorização de acesso que os usuários terão em nossa aplicação web.

Para desabilitar o aceso de usuários que não estejam autenticados em nosso site devemos adicionar ao arquivo web.config o seguinte código:

```
<authorization>
  <deny users="?">
</authorization>
```

Após termos adicionado essa configuração no web.config, defina a página Default.aspx como a página inicial de nossa aplicação e rode o sistema. Iremos perceber agora dois comportamentos diferentes do que estava ocorrendo:

- Você foi redirecionado para a página login.aspx. Isto ocorre porque está é a página que nós configuramos como a página onde o usuário irá se autenticar para ter acesso ao nosso sistema e como nós restringimos o acesso a usuários desconhecidos, agora somente usuários autenticados poderão navegar dentro de nosso web site.
- Na url, o asp.net adicionou informações da página que foi tentado o acesso. Isso é necessário para que o asp.net saiba para qual página ele deve redirecionar o site logo apóis.

Entendendo o gerenciamento da segurança

Agora que já trancamos o acesso de qualquer pessoa que não se autentique em nossa aplicação devemos criar uma funcionalidade que disponibilize a usuários que criem contas para acesso em nosso site. Dependendo

Segurança

do tipo de aplicação essa página só será acessada por administradores de sistema ou pode ser uma página que tenha livre acesso. Um exemplo prático seria um sistema de um fórum onde para ter acesso o usuário deve ser cadastrado no sistema. Dessa forma a página de criação de usuários teria livre acesso e as páginas do fórum só seriam acessadas por usuários logados.

Na montagem das páginas de nossa aplicação nós definimos uma página responsável por criar novos usuários chamada “NewUser.aspx”. Vamos configurar em nossa aplicação que o acesso à página de criação de usuários pode ser pública, como no exemplo do fórum.

Para que possamos configurar uma página com acesso público devemos adicionar uma nova chave após o fechamento do nó `<system.web>`, a chave `location`. Dentro da chave `location` iremos apontar o caminho da página que ela irá gerenciar e iremos criar, dentro da chave `location` uma nova referência ao `<system.web>` e iremos configurar uma nova `authorization`, que gerenciará somente a página apontada pela `location`.

Uma das grandes desvantagens de trancar o acesso ao conteúdo de nosso site, como fizemos em nossa aplicação, é que todo o conteúdo fica desabilitado para interação com o usuário. Você pode confirmar esse problema, se rodar a aplicação novamente e limpar o cache do navegador (ctrl + f5 no ie). Após limparmos o cache do navegador iremos perceber que toda a formatação de layout de nosso site realizada a partir dos arquivos css foi perdida. Isso ocorre porque o usuário ainda não se autenticou e está proibido de acessar o conteúdo da pasta que armazena os arquivos css. Para contornar esse problema devemos habilitar o acesso de todos os usuários para as pastas que contêm arquivos css, javascripts e outros conteúdos que o sistema deve utilizar sem que o usuário esteja logado no sistema.

Para habilitarmos o acesso de usuários não autenticados a página de criação de usuários e aos arquivos de css devemos inserir o seguinte código no arquivo `web.config`:

```
<location path="Styles">
  <system.web>
    <authorization>
      <allow users="*"/>
    </authorization>
  </system.web>
</location>

<location path="Security/NewUser.aspx">
  <system.web>
    <authorization>
      <allow users="*"/>
    </authorization>
  </system.web>
</location>
```

Após termos inserido esse código no `web.config` vamos inserir um texto na página de login de nossa aplicação e um hyperlink que irá redirecionar um usuário para a página de criação de contas de usuários. Dessa forma será

Segurança

possível que um usuário que não tenha nenhuma conta possa se registrar em nosso sistema.

```
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">

    <h2>
        Login do Sistema
    </h2>
    <p>
        Se você não é um usuário registrado clique <asp:HyperLink ID="hyperLinkNewUser" runat="server" Text="aqui"
            NavigateUrl "~/Security/NewUser.aspx" ></asp:HyperLink> para gerar uma conta!
    </p>

</asp:Content>
```

Agora vamos rodar nosso sistema com a página Default.aspx setada como página inicial de nossa aplicação. Poderemos ver que automaticamente seremos redirecionados para a página de Login e agora teremos um hyperlink que nos redirecionará para a página de criação de usuários. Podemos reparar que tanto a página de usuários como a configuração de layout (css) voltaram a ser acessíveis por usuários que não estão logados em nosso sistema.

O próximo passo é inserirmos a funcionalidade de criação de usuários, porque até agora apenas disponibilizamos a navegação até a página de criação de usuários, mas a página ainda não possui nenhuma funcionalidade.

Criando Usuários e explorando a estrutura criada

O Visual Studio 2010 disponibiliza o controle CreateUserWizard. Esse controle é responsável por toda a criação de novos usuários que estarão vinculados com nossa aplicação. Todo o gerenciamento de criação de usuários está encapsulado dentro do controle.

Para utilizarmos esse controle basta adicionarmos o controle em nossa página de criação de usuários. O controle possui diversas propriedades de customização. É interessante que você dê uma olhada nas propriedades do controle para ter uma idéia da capacidade de personalização.

Seguindo a construção de nossa aplicação, agora vamos adicionar o controle CreateUserWizard ao formulário NewUser.aspx. O código do formulário deve ficar conforme a figura abaixo.

Segurança

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Main.Master" AutoEventWireup="true" CodeBehind="NewUser.aspx.cs"
Inherits="CursoAspNetAvancado4._0.Security.NewUser" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
<h2>
    Cadastro de usuários
</h2>
<br />
<asp:CreateUserWizard ID="CreateUserWizard1" runat="server">
    <WizardSteps>
        <asp:CreateUserWizardStep ID="CreateUserWizardStep1" runat="server">
        </asp:CreateUserWizardStep>
        <asp:CompleteWizardStep ID="CompleteWizardStep1" runat="server">
        </asp:CompleteWizardStep>
    </WizardSteps>
</asp:CreateUserWizard>
</asp:Content>
```

Após ter inserido esse código no NewUser.aspx, coloque o formulário em modo design, clique com o botão direito sobre o controle, selecione a opção Auto Format e selecione o schema Professional e aplique. Com isso iremos deixar o nosso controle com uma aparência pré-definida.

Agora vamos rodar a nossa aplicação, iremos ir direto para a página de login, utilizaremos o atalho para navegar até a página de criação de usuários e iremos realizar um cadastro de nosso usuário para navegar em todo o nosso sistema.

O preenchimento do controle será como o da figura abaixo.

The screenshot shows a user registration form titled "CADASTRO DE USUÁRIOS". The form is titled "Sign Up for Your New Account". It contains the following fields:

- User Name: Marcelo
- Password: (redacted)
- Confirm Password: (redacted)
- E-mail: marcelorosenthal@hotmail.com
- Security Question: Nome do Cachorro
- Security Answer: Sofia

At the bottom is a "Create User" button.

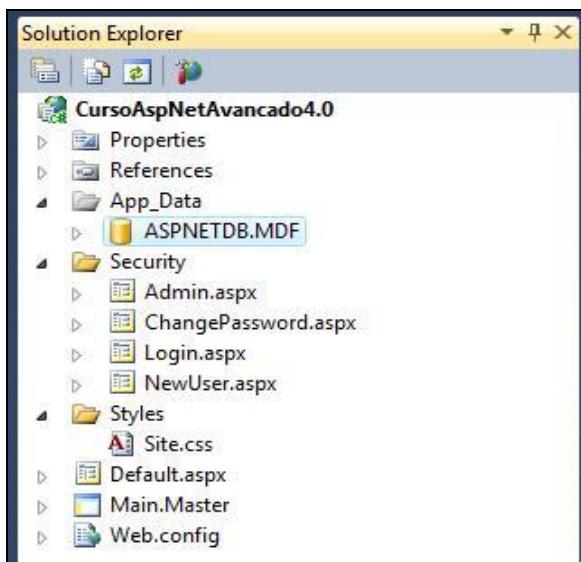
Após termos criado o usuário o controle muda de aparência e informa que o usuário foi criado com sucesso!

The screenshot shows a confirmation message on the "CADASTRO DE USUÁRIOS" page. The message is "Complete" and "Your account has been successfully created." At the bottom is a "Continue" button.

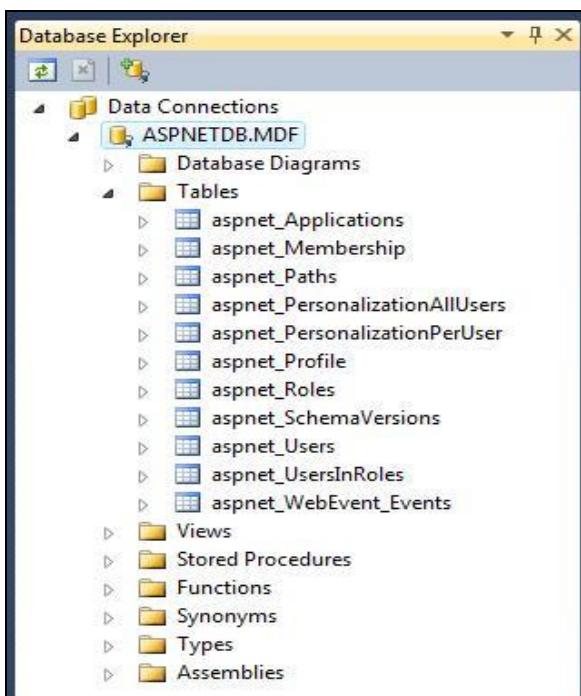
Segurança

Após termos criado o usuário vamos voltar em nossa aplicação e vamos explorar a estrutura criada pelo asp.net para gerenciamento dos dados referentes aos usuários do sistema.

Para explorarmos a estrutura criada devemos ir ao Solution Explorer e expandir a pasta App_Data, note que dentro desta pasta foi criado um banco SQL Express chamado ASPNETDB.MDF. Dentro deste banco estarão as tabelas responsáveis por armazenar todos os usuários e informações relacionadas à segurança em nossa aplicação.



Ao darmos um duplo clique sobre o banco de dados será aberta a janela Database Explorer. Dessa forma poderemos analisar toda a estrutura de tabelas de nosso banco de dados. Visualize os dados da tabela aspnet_Users e perceba que os dados do usuário cadastrado estão armazenados dentro desta tabela.



Segurança

Uma das grandes limitações de trabalhar com o modelo de gerenciamento de segurança do asp.net é que ele trabalha em conjunto com o banco ASPNETDB.MDF e muitas vezes queremos que a segurança trabalhe em conjunto com nossa aplicação. Para contornar esse problema devemos implementar um Membership Provider apontado para a nossa base de dados. Quando implementamos um MemberShip Provider estamos sobreescrevendo as funcionalidades de gerenciamento de segurança do asp.net e estamos adotando a nossa implementação como o gerenciador de segurança. Para mais detalhes de como implementar um MemberShip Provider consulte o seguinte link: <http://msdn.microsoft.com/en-us/library/6tc47t75.aspx>.

Validando usuários com o Login

Após termos criado um usuário para acessar nossa aplicação devemos agora preparar uma tela de login, onde o usuário irá entrar com suas credenciais e o sistema irá validar se ele realmente é quem ele diz ser e caso a resposta seja positiva o sistema deva permitir o livre acesso do usuário a áreas que a ela sejam permitidas.

Para executar o processo de validação do usuário o asp.net disponibiliza o controle Login. Toda a lógica de controle de acesso do usuário está encapsulada dentro do controle.

Para implementarmos essa funcionalidade em nossa aplicação, vamos na página Login.aspx e vamos arrastar um controle login para dentro de nosso form, abaixo da tag de fechamento do parágrafo (

</p>). Para deixarmos o visual com um aspecto legal é interessante adicionar uma linha (
) entre o final do parágrafo e o nosso controle. Após termos inserido o controle vamos definir o estilo do controle como Professional. O resultado deve ficar igual à figura abaixo.



Vamos rodar o sistema, ao digitarmos o usuário e a senha correta seremos direcionados para a página Default.aspx, caso digitemos usuário ou senha errado, ficaremos trancados na tela de login.

Com esse controle o asp.net gerencia todo o controle de acesso ao nosso sistema e não precisamos programar nenhuma linha.

Utilizando o ChangePassword para trocar as senhas

Outra funcionalidade importante que todo o gerenciamento de segurança deve possuir é a possibilidade de trocar as senhas dos usuários da aplicação.

O asp.net disponibiliza o controle ChangePassword. Esse controle é responsável por gerenciar a troca de senhas dos usuários. Para utilizá-lo basta arrastar o controle para dentro de um web form. Toda a regra de negócio já está encapsulada dentro do controle. Nós não precisamos escrever nenhuma linha de código para que o controle funcione.

Para implementarmos essa funcionalidade em nossa aplicação, vamos na página ChangePassword.aspx e vamos inserir o controle em nosso formulário. O código do web form deve ficar conforme abaixo.

```
<%@ Page Title="" Language="C#" MasterPageFile "~/Main.Master" AutoEventWireup="true"
CodeBehind="ChangePassword.aspx.cs"
Inherits="CursoAspNetAvancado4._0.Security.ChangePassword" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
<h2>
    Trocar Senha
</h2>
<br />
<asp:ChangePassword ID="chgPswPrincipal" runat="server">
</asp:ChangePassword>
</asp:Content>
```

Após ter inserido o controle configure o controle para utilizar o estilo Professional. Devemos também inserir um atalho de navegação para a tela de trocar senha em nossa aplicação. Para isso insira o código abaixo na página Main.master logo abaixo do fechamento da div que utiliza a class "title".

```
<div class="loginDisplay">
    <asp:HyperLink ID="hyperLinkNewUser" runat="server" Text="Trocar Senha"
        NavigateUrl="~/Security/ChangePassword.aspx" ></asp:HyperLink>
</div>
```

Agora vamos rodar nossa aplicação, note que somente conseguiremos navegar para a página de trocar a senha após termos logado no sistema. O resultado dá página de trocar a senha deve ser igual a figura abaixo.



Segurança

Após termos realizado a alteração de uma senha o controle deve apresentar uma mensagem informando que a alteração foi realizada com sucesso.

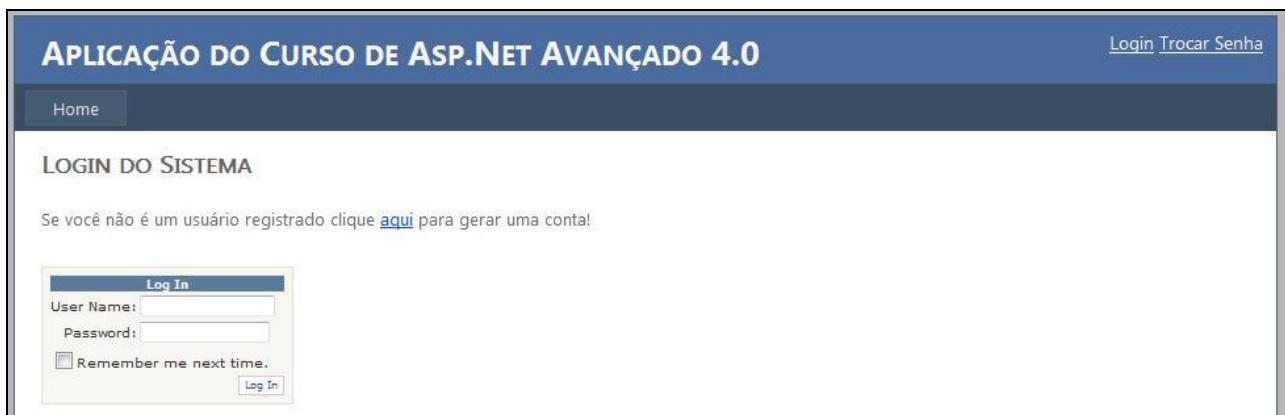
Um comportamento que nossa aplicação está tendo que não é legal é sempre apresentar para o usuário a opção de trocar a senha, mesmo que o usuário não esteja logado. O correto é apresentar essa opção somente para quem já se logou no sistema. Veremos adiante como podemos resolver esse problema.

Entendendo e utilizando os controles LoginStatus, LoginName e LoginView

Podemos ver em nossa aplicação que ainda não criamos nenhum atalho a tela de login e nem uma opção para o usuário efetuar o logout do sistema. Para implementarmos essa funcionalidade podemos utilizar o controle LoginStatus que o asp.net fornece.

O controle LoginStatus funciona como um atalho para o login ou para o logout do usuário. O controle funciona com duas visualizações: Logged Out, que funciona quando o usuário não está autenticado e a visualização Logged In que funciona quando o usuário já se autenticou em nosso sistema.

Para entendermos o funcionamento do LoginStatus vamos utilizá-lo em nossa aplicação. Vamos na página Main.master e vamos adicionar um controle LoginStatus logo acima do hyperlink de trocar senha. No controle LoginStatus vamos definir a visão como LoggedOut e vamos rodar nossa aplicação.



Note que enquanto não nos logamos no sistema o texto que o controle apresenta é “Login” e sempre que clicarmos no controle seremos redirecionados para a página de login. Após termos nos logado o controle irá apresentar o texto Logout. Quando clicarmos no controle LoginStatus e ele estiver apresentando a opção de logout o nosso cookie de autenticação será excluído e seremos redirecionados para a página de login para novamente nos autenticarmos em nossa aplicação.

The screenshot shows a web application titled "APLICAÇÃO DO CURSO DE ASP.NET AVANÇADO 4.0". The main content area displays the message "BEM-VINDO A PÁGINA PRINCIPAL DE NOSSA APLICAÇÃO!" and a note stating "No curso de asp.net avançado iremos ver algumas das principais funcionalidades do desenvolvimento web.". The top right corner has links for "Logout" and "Trocar Senha". A navigation bar at the top includes a "Home" link.

Outra funcionalidade importante em aplicações web é apresentar o nome do usuário que está logado no sistema. Para isso temos o controle `LoginName`. Esse controle simplesmente apresenta o nome do usuário autenticado em nossa aplicação.

Para termos um efeito mais profissional em nossa aplicação utilizamos a propriedade `formatString` para mesclar textos juntamente com o nome do usuário.

Vamos implementar essa funcionalidade em nossa aplicação. Vamos na página `Main.master` e vamos adicionar o seguinte código logo acima do controle `LoginStatus`.

```
<div class="loginDisplay">
    <asp:LoginName ID="lgnNome" runat="server" FormatString="Seja bem-vindo {0} - " />
    <asp:LoginStatus ID="LoginStatus1" runat="server" />
    -
    <asp:HyperLink ID="hyperLinkNewUser" runat="server" Text="Trocar Senha "
        NavigateUrl "~/Security/ChangePassword.aspx"></asp:HyperLink>
</div>
```

Agora vamos rodar nossa aplicação, enquanto não estivermos logado nenhuma mensagem será apresentada, mas após nos logarmos em nosso sistema será apresentado à mensagem: "Seja bem-vindo (nome do usuário que se logou)".

The screenshot shows the same application as before, but now personalized for the user "Marcelo". The top right corner shows "Seja bem-vindo Marcelo - Logout - Trocar Senha". The main content area displays the message "BEM-VINDO A PÁGINA PRINCIPAL DE NOSSA APLICAÇÃO!" and the note about the advanced course. The navigation bar at the top includes a "Home" link.

Podemos perceber que a opção de `Trocar Senha` sempre fica visível, mesmo que o usuário não esteja autenticado. Já levantamos essa questão e apontamos como uma falha de design em nossa aplicação. Para resolvermos isso podemos utilizar o controle `LoginView`.

Segurança

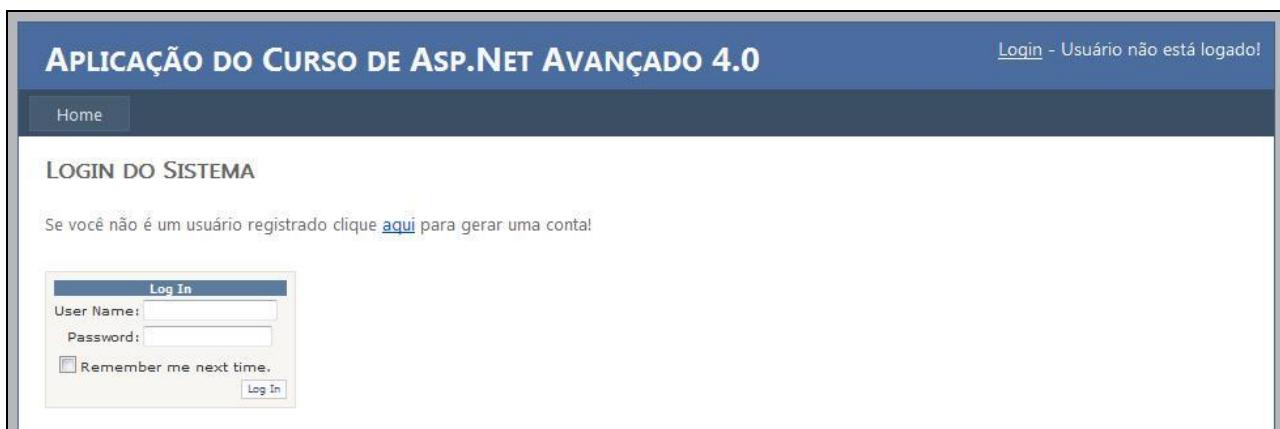
O controle LoginView tem como objetivo exibir determinado conteúdo quando o usuário estiver autenticado e outro conteúdo enquanto o usuário não se autenticar em nosso sistema.

O controle é formado por dois templates: Anonymous e Loggedin. No template Anonymous colocaremos o conteúdo que queremos que seja apresentado quando o usuário não esteja autenticado e no template Loggedin colocaremos o conteúdo que queremos que seja apresentado quando o usuário se autenticar.

Para entendermos melhor esse funcionamento vamos em nossa aplicação, na página Main.master e vamos alterar o conteúdo que está dentro da div que utiliza a class "loginDisplay". Vamos adicionar um loginView e dentro do template anonymous vamos adicionar uma label que irá informar para o usuário que não tem nenhum usuário logado no sistema e dentro do template Loggedin vamos colocar o nosso hyperlink de trocar a senha. O código deve ficar conforme abaixo:

```
<div class="loginDisplay">
    <asp:LoginName ID="lgnNome" runat="server" FormatString="Seja bem-vindo {0} - " />
    <asp:LoginStatus ID="LoginStatus1" runat="server" />
    -
    <asp:LoginView ID="lvwVisao" runat="server">
        <AnonymousTemplate>
            <asp:Label ID="lblNaoLogado" runat="server" Text="Usuário não está logado!" />
        </AnonymousTemplate>
        <LoggedInTemplate>
            <asp:HyperLink ID="hyperLinkNewUser" runat="server" Text="Trocara Senha"
                NavigateUrl("~/Security/ChangePassword.aspx")></asp:HyperLink>
        </LoggedInTemplate>
    </asp:LoginView>
</div>
```

Agora vamos rodar a nossa aplicação. Enquanto o usuário não se loga o loginview irá exibir a label informando que o usuário não se logou.



Após o usuário ter se logado o loginview irá apresentar o hyperlink de trocar a senha.

The screenshot shows a web application interface. At the top, there's a blue header bar with the text "APLICAÇÃO DO CURSO DE ASP.NET AVANÇADO 4.0". To the right of this, in a smaller font, is "Seja bem-vindo Marcelo - [Logout](#) - [Trocar Senha](#)". Below the header is a dark grey navigation bar containing a single item, "Home". The main content area has a white background. It features a bold heading "BEM-VINDO A PÁGINA PRINCIPAL DE NOSSA APLICAÇÃO!" followed by a descriptive paragraph: "No curso de asp.net avançado iremos ver algumas das principais funcionalidades do desenvolvimento web.".

Gerenciamento Manual de usuários

Vimos até agora como podemos implementar segurança em nossas aplicações asp.net utilizando os controles disponibilizados pelo asp.net. Os controles encapsulam todas as regras de negócio e funcionam de uma forma pré-definida em seu desenvolvimento.

Muitos programadores simplesmente não gostam de deixar o gerenciamento de uma funcionalidade tão importante, como a segurança do site, ao cargo de controles em que o programador não consegue gerenciar todo o fluxo de trabalho. Pensando em disponibilizar uma maior flexibilidade ao programador o .net framework disponibiliza todas as funcionalidades de seus controles dentro de classes que estão na namespace System.Web.Security. Dessa forma tudo que vimos até agora podemos implementar manualmente.

Na tabela abaixo é apresentado algumas das classes disponíveis no System.Web.Security e suas funcionalidades.

Classe	Funcionalidade
Membership	Responsável por validar credenciais de usuários e gerenciar configurações de usuários.
MembershipCreateUserException	Armazena os possíveis erros ao criar novos usuários.
MembershipPasswordException	Armazena os possíveis erros referentes a senhas dos usuários.
MembershipProvider	Define os contratos de implementação do membership provider do asp.net com providers customizados.
MembershipUser	Responsável por buscar e atualizar informações do usuário na base de dados.
Roles	Gerencia a participação do usuário em grupos de funções.

Em função de possuirmos muitas classes nesse namespace estamos citando apenas algumas classes. Para um estudo mais aprofundado de todas as opções acesse: <http://msdn.microsoft.com/en-us/library/k5ss5tk.aspx>

Classes Membership e MembershipUser

Essas duas classes podem ser consideradas as mais importantes quando queremos implementar manualmente o gerenciamento de segurança. Abaixo vamos listar alguns dos principais métodos dessas classes.

- Classe Membership
 - CreateUser – Cria um novo usuário.
 - DeleteUser – Exclui um usuário.
 - FindUserByEmail – Localiza usuários a partir do email cadastrado.
 - FindUserByName – Localiza usuário a partir do nome.
 - GeneratePassword – Gera um senha randômica.
 - GetAllUsers – Retorna uma coleção com todos os usuários.
 - UpdateUser – Atualiza as informações do usuário.
 - ValidateUser – Validad se o nome do usuário e a senha são válidos.
- Para mais informações sobre a classe Membership acesse o seguinte link: <http://msdn.microsoft.com/en-us/library/system.web.security.membership.aspx>
- Classe MembershipUser
 - CreationDate – Data e hora de criação do usuário.
 - Email – Retorna ou seta o email de um determinado usuário.
 - IsOnline – Indica se o usuário está online.
 - PasswordQuestion – Retorna a questão da senha que o usuário cadastrou.
 - ChangePassword – Altera a senha de um usuário.
 - ResetPassword – Reseta a senha do usuário, gerando automaticamente uma nova senha.
- Para mais informações sobre a classe MembershipUser acesse o seguinte link: <http://msdn.microsoft.com/en-us/library/system.web.security.membershipuser.aspx>.

Segurança

Utilizando Grupos

Estudamos os métodos de gerenciamentos de usuários implementados com controles e com o auxilio de classes manuais, mas ainda falta entendermos como funcionam os Grupos(roles) dos usuários.

Grupos servem para definir perfis dos usuários que estão vinculados a ele. Por exemplo, normalmente em um sistema temos o grupo administrativo que tem acesso a todas as funcionalidades do sistema e temos o grupo operacional onde os usuários deste grupo têm acesso a algumas páginas que usuários não autenticados não possuem e não tem acesso a algumas páginas que somente usuários do grupo administrativo têm.

O gerenciamento de grupos de usuários não é realizado com controles. Deve ser implementado pelo programador. O fluxo de trabalho é de gerar grupos e associar o grupo com os usuários que devem fazer parte dele.

Para entendermos o funcionamento dos grupos de usuários vamos implementá-lo em nossa aplicação.

Primeiramente devemos habilitar em nossa aplicação o gerenciamento de grupos de usuários. Por padrão essa opção é desabilitada. Para isso devemos inserir o seguinte código no web.config de nossa aplicação dentro da tag <system.web>.

```
<roleManager enabled="true" />
```

Após termos configurado nossa aplicação para utilizar o gerenciamento de grupos vamos adicionar uma página, na pasta Security, chamada cadGrupos.aspx e vamos incluir uma opção no menu de acesso a essa página.

O comportamento que queremos em nossa aplicação é criar um grupo chamado admin e somente usuários vinculados com o grupo admin possuam acesso a página “admin.aspx” que está dentro da pasta security em nossa aplicação.

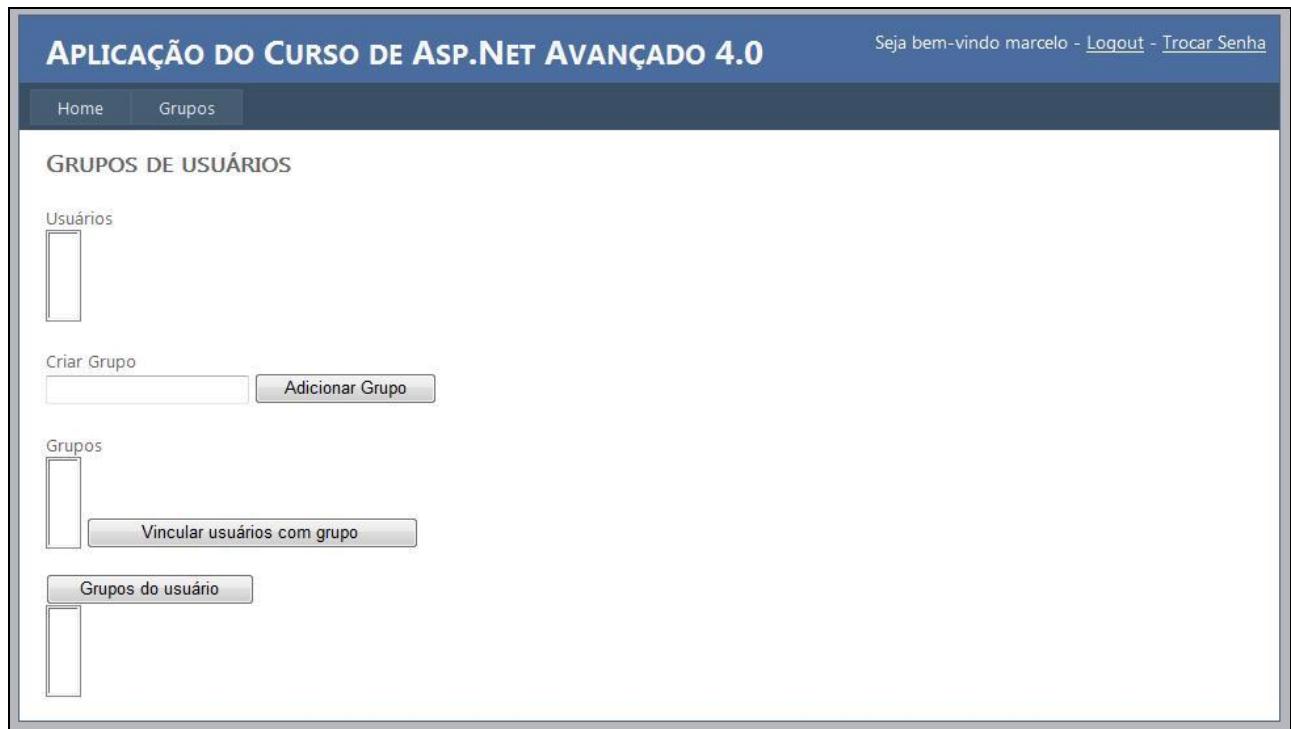
Para isso vamos criar na página cadGrupos.aspx as seguintes funcionalidades:

- Uma listagem apresentando todos os usuários cadastrados.
- Uma caixa de texto e um botão onde o usuário irá digitar o nome do grupo e o botão irá criar esse grupo.
- Uma listagem que irá apresentar todos os grupos criados.
- Um botão que irá associar os usuários do sistema com os grupos disponíveis.

Para conseguirmos essa interface vamos inserir o seguinte código na página cadGrupos.aspx.

```
<%@ Page Title="" Language="C#" MasterPageFile "~/Main.Master" AutoEventWireup="true" CodeBehind="cadGrupos.aspx.cs"
Inherits="CursoAspNetAvancado4._0.Security.cadGrupos" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
    <h2>
        Grupos de usuários
    </h2>
    <p>
        <asp:Label ID="Label1" runat="server" Text="Usuários"></asp:Label>
        <br />
        <asp:ListBox ID="lstUsuarios" runat="server"></asp:ListBox>
    </p>
    <p>
        <asp:Label ID="Label2" runat="server" Text="Criar Grupo"></asp:Label>
        <br />
        <asp:TextBox ID="txtNomeGrupo" runat="server"></asp:TextBox>
        <asp:Button ID="btnAddGrupo" runat="server" Text="Adicionar Grupo" />
    </p>
    <p>
        <asp:Label ID="Label3" runat="server" Text="Grupos"></asp:Label>
        <br />
        <asp:ListBox ID="lstGrupos" runat="server"></asp:ListBox>
        <asp:Button ID="btnVincularUsuarioGrupo" runat="server" Text="Vincular usuários com grupo" />
    </p>
    <p>
        <asp:Button ID="btnShowGrupos" runat="server" Text="Grupos do usuário" />
        <br />
        <asp:ListBox ID="lstGruposUsuario" runat="server"></asp:ListBox>
    </p>
</asp:Content>
```

Quando renderizarmos nossa página ela deve ficar com o seguinte aspecto.



O próximo passo é carregarmos a listBox de usuários com todos os usuários disponíveis e gerarmos um método que irá carregar a listBox de grupos com todos os grupos disponíveis.

Segurança

Vamos inserir o seguinte código na página cadGrupos.aspx.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        this.lstUsuarios.DataSource = System.Web.Security.Membership.GetAllUsers();
        this.lstUsuarios.DataBind();

        Grupos_Databind();
    }
}
protected void Grupos_Databind()
{
    this.lstGrupos.DataSource = System.Web.Security.Roles.GetAllRoles();
    this.lstGrupos.DataBind();
}
```

Note que estamos atualizando a listBox de usuários sempre que carregamos a pagina e colocamos o carregamento da listBox de grupos no método “Grupos_Databind”. Criamos o método “Grupos_Databind” para que esse método possa ser reaproveitado em mais de uma situação em nossa página.

O próximo passo é criarmos o código do botão “Adicionar Grupo”. Vamos dar um duplo clique no botão e vamos inserir o código abaixo.

```
protected void btnAddGrupo_Click(object sender, EventArgs e)
{
    System.Web.Security.Roles.CreateRole(this.txtNomeGrupo.Text);
    Grupos_Databind();
    this.txtNomeGrupo.Text = string.Empty;
}
```

Esse código irá chamar o método CreateRole passando o nome do grupo que foi digitado na TextBox, dessa forma criando um novo grupo. Após criar o grupo o método chama o método para atualizar a listBox de grupos e limpa a TextBox.

Roda sua aplicação e crie um grupo chamado admin.

O próximo passo é criarmos a funcionalidade de vincular usuários com grupos de usuários. Para isso vamos inserir o código abaixo no evento click do botão “Vincular usuários com grupo ((btnVincularUsuarioGrupo)).

```
protected void btnVincularUsuarioGrupo_Click(object sender, EventArgs e)
{
    System.Web.Security.Roles.AddUserToRole(this.lstUsuarios.SelectedItem.Text,
        this.lstGrupos.SelectedItem.Text);
}
```

Segurança

Neste código é pego o nome do usuário que está selecionado na listBox de usuários e o nome do grupo que está selecionado na listBox de Grupos e é realizado a vinculação entre o usuário e o grupo através do método AddUserToRole.

O próximo passo é Adicionar o código do botão Grupos do usuário, que irá apresentar todos os grupos vinculados com o usuário que está selecionado na listBox de usuários. Para isso vamos inserir o seguinte código do evento click do botão grupos de usuário.

```
protected void btnShowGrupos_Click(object sender, EventArgs e)
{
    this.lstGruposUsuario.DataSource = System.Web.Security.Roles.GetRolesForUser(this.lstUsuarios.SelectedItem.Text);
    this.lstGruposUsuario.DataBind();
}
```

Agora vamos rodar nossa aplicação e vamos criar um novo usuário, este usuário não irá ser vinculado com nenhum grupo e irá servir para testarmos se realmente a aplicação irá trancar o acesso de usuários que não estão vinculados com o grupo admin podem ter acesso a pagina Admin.aspx. Após termos criado o novo usuário vamos na página de grupos e vamos vincular o usuário que estamos utilizando em nossa aplicação com o grupo Admin. Após termos efetuado o vinculo selecione o primeiro usuário na caixa de listagem e clique em grupos de usuário e veja o grupo admin sendo mostrado como uma grupo vinculado, após selecione o novo usuário e clique em grupos de usuários e note que nenhum grupo será apresentado.

Agora temos que garantir que somente usuários do grupo admin possam acessar a página Admin.aspx, para isso vamos adicionar o seguinte código em nosso web.config, logo abaixo do fechamento da tag location referente a página “Security/newUser.aspx”.

```
<location path="Security/Admin.aspx">
    <system.web>
        <authorization>
            <allow roles="admin"/>
            <deny users="*"/>
        </authorization>
    </system.web>
</location>
```

A estrutura do nó é muito semelhante a estrutura criada para gerenciar a pagina NewUser.aspx. A diferença é que aqui negamos todos os acessos a página menos os de usuários que pertencem ao grupo admin.

Agora crie um atalho para a página Admin.aspx no menu da Master Page, rode o sistema, primeiro acesse com o novo usuário criado e note que não consegue acessar a página. Troque o usuário logado para o usuário que está vinculado ao grupo admin e perceba que agora é possível acessar a página.

Exercícios

1. Crie uma aplicação web que implemente os controles de segurança do asp.net 4.0.
2. Refaça os controles de segurança de sua aplicação gerenciando todas as funcionalidades de segurança manualmente.
3. Vincule seus usuários com grupos pré-definidos e apresente conteúdo personalizado para os usuários em virtude dos grupos que eles estão vinculados. Lembre-se que um usuário pode estar vinculado com mais de um grupo.

Espaço para Anotações

2. Globalização e Localização

Objetivos

Ao final deste capítulo você estará apto à:

- Conhecer os recursos de localização e globalização do ASP.Net.
- Formatar os dados de suas páginas asp.net respeitando a cultura do browser.
- Utilizar os recursos de tradução local e global em sua aplicação ASP.Net

O que é globalização e localização?

Globalização é o processo de criar uma aplicação que atenda as necessidades de usuários de múltiplas culturas. Este processo envolve muito mais do que simplesmente traduzir elementos de interface do usuário da aplicação em múltiplas linguagens, ele também inclui usar o sinal da moeda corrente, formato de data/hora, calendário, etc. Acomodar essas diferenças culturais em uma aplicação é chamado de localização.

Localizar uma aplicação ASP.NET significa ajustar a aplicação a cultura corrente do browser do usuário, fazendo com que dados sejam apresentados em um formato familiar do usuário. Abaixo alguns exemplos de dados que sofrem interferência pela localização:

- Formato de Data
 - Cada país possui uma notação padrão para datas. Exemplo: Brasil utiliza o padrão dd/mm/aaaa; EUA utiliza mm/dd/aaaa
- Idioma
 - Seria como traduzir o seu aplicativo para dois ou mais idiomas.
- Formato de moeda
 - Tal qual o formato de data, cada país tem sua notação padrão para moeda. Por exemplo, Brasil utiliza R\$999.999,99; EUA utiliza U\$999,999.99

Existem dois tipos de cultura a serem consideradas em um aplicativo ASP.NET:

- Culture - Responsável pela formatação de dados a serem utilizados pelo sistema (data, moeda etc.);
- UICulture - Responsável pela formatação dos recursos a serem apresentados na interface com o usuário, isso é, tradução dos dados existentes no Web Form (conteúdo e propriedades de controles como: labels, botões, títulos, tooltips etc.).

O .Net Framework 4.0 suporta um mínimo de 354 culturas diferentes, tendo evoluído de um mínimo de 203 culturas no .net framework 3.5. Muito dessas culturas são culturas neutras que foram incluídas para adicionar informações a culturas pais.

Localizando valores em uma aplicação asp.Net

Para que uma aplicação asp.net exiba para o usuário dados localizados em uma determinada cultura é necessário que seja configurado o parâmetro `culture`. Este parâmetro é responsável por informar ao asp.net como ele deve proceder em relação a dados de localização dentro da aplicação.

O parâmetro `culture` pode ser configurado das seguintes maneiras:

- Detectar automaticamente a cultura do usuário;
- Detectar automaticamente a cultura do usuário e caso não gerencie a cultura corrente utilizar uma cultura padrão;
- Forçar a utilização de uma determinada cultura.

Agora para alcançarmos um entendimento da utilização de culturas, vamos configurar nossa aplicação para trabalhar com culturas.

Como nossa aplicação trabalha com Master.Pages vamos colocar um controle na master que estará integrado com as culturas e outro controle na página Default.aspx. Vamos colocar em dois lugares diferentes para acompanharmos os cuidados que devemos ter ao utilizar culturas nas Master.Pages.

Primeiro passo devemos adicionar o seguinte código em nosso arquivo Site.css.

```
.title h2
{
    color:White;
    font-size:1em;
}
```

Após termos inserido o tratamento visual para a tag `h2` dentro da `div Title`, vamos atualizar o código da `div Title` do Main.master conforme abaixo.

```
<div class="title">
    <h1>
        Aplicação do Curso de Asp.Net Avançado 4.0
    </h1>
    <h2>
        <asp:Label runat="server" ID="lblMsgData" Text="Data: " />
        <asp:Label runat="server" ID="lblData" Text="" />
    </h2>
</div>
```

O próximo passo é inserirmos o seguinte código na pagina de códigos do Main.master.

```
protected void Page_Load(object sender, EventArgs e)
{
    this.lblData.Text = System.DateTime.Now.ToString();
}
```

Com estas alterações sempre que entramos em nossa aplicação iremos apresentar a data atual no cabeçalho de nossa Main.master.

Agora o próximo passo é atualizar a página Default.aspx com controles que irão apresentar a data na página. O primeiro passo é atualizar a tag h2 com o seguinte código.

```
<h2>
    Bem-vindo a página principal de nossa aplicação! - Data de entrada: <asp:Label ID="lblData" runat="server" />
</h2>
```

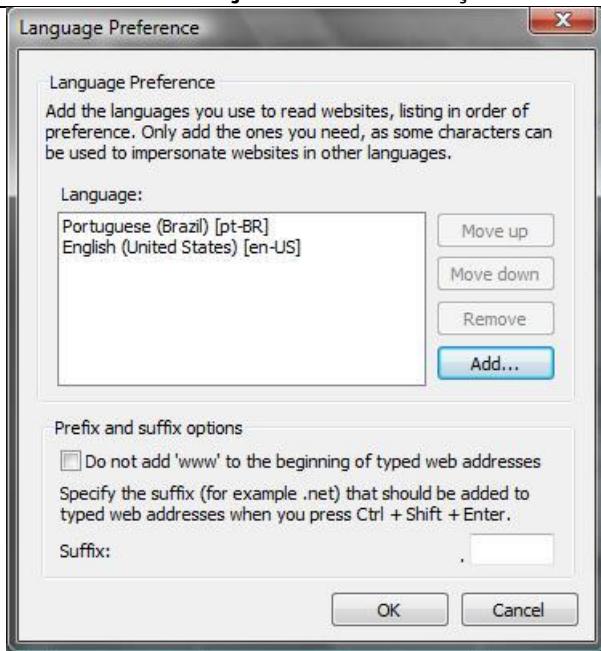
O próximo passo é atualizar o code behind da mesma forma que fizemos com a main.master. Abaixo o código que deve ser inserido na default.aspx.cs.

```
protected void Page_Load(object sender, EventArgs e)
{
    this.lblData.Text = System.DateTime.Now.ToString();
}
```

No exemplo acima inserimos uma label que irá apresentar a data e a hora atual tanto na página default.aspx como na main.master. Ao executarmos a nossa aplicação é possível visualizar a formatação dos dados de acordo com as configurações regionais do servidor em que a aplicação está sendo executada. Mas devemos nos preocupar que em aplicações que atendam diferentes culturas, as configurações regionais do servidor não são as mesmas configurações regionais do usuário. Uma forma de detectarmos qual é a cultura do usuário é a partir da configuração regional do seu browser. Se o browser é um internet Explorer podemos verificar o seu idioma a partir do seguinte caminho:

- Menu ferramentas;
- Opções de internet;
- Aba Geral, botão idiomas;

Globalização e Localização



Vamos definir a língua americana para ser o idioma default do browser e vamos rodar nossa aplicação. Podemos ver que nenhuma mudança de comportamento ocorre quando mudamos o idioma do browser. Para que a aplicação entenda que deve utilizar a cultura do browser do usuário devemos adicionar o parâmetro culture na diretiva da pagina default.aspx. Note que a main.Master não possui o parâmetro culture em suas opções. Isso ocorre porque todas as Master Pages herdam de System.Web.UI.UserControl e as diretivas referentes a globalização estão dentro de SystemWeb.UI.Page.

Para continuarmos nosso exercício devemos configurar Default.aspx conforme a imagem abaixo.

```
<%@ Page Title="Página Principal" Language="C#" MasterPageFile="~/Main.Master" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="CursoAspNetAvancado4._0.Default" Culture="auto" %>
```

Vamos rodar novamente nossa aplicação, note que na tela de login o campo data que está na master Page está mostrando a data com o formato em português (ou a configuração regional do computador que você está utilizando). Após logarmos a aplicação seremos redirecionados para a página Default.aspx e o campo data tanto da página default.aspx como da master Page estarão apresentando o formato da data em inglês (levando em conta que você está usando o idioma inglês em seu browser, caso não esteja utilizando coloque o idioma do browser para inglês).

Navegue entre as opções do menu e note que quando saímos da página default.aspx o campo data da master page muda o formato da data apresentado. Esse comportamento ocorre porque a master page implementa a cultura da página filha que ela está apresentando. Para que não tenhamos esse comportamento devemos configurar a cultura dentro do web.config.

Globalização e Localização

Sempre que configuramos a cultura no web.config, esta configuração irá refletir em todo o sistema. Esta é uma opção muito vantajosa para nós programadores. Imagine a seguinte situação: Temos uma aplicação asp.net que possui aproximadamente 200 web forms. Para setarmos as diretivas de Culture de todos os web forms teríamos que entrar em cada um deles e configurar na sua diretiva de pagina o valor do parâmetro Culture. Fazendo com que tenhamos um grande trabalho manual. Se configurarmos no web.config basta uma única linha de código e toda nossa aplicação irá utilizar as diretivas de cultura especificadas no web.config.

Configuramos o web.config com a seguinte diretiva dentro da tag <system.web>.

```
<system.web>
  <compilation debug="true" targetFramework="4.0" />
  <authentication mode="Forms">
    <forms name=".ASPxAUTH" loginUrl("~/Security/Login.aspx" defaultUrl="Default.aspx" />
  </authentication>
  <authorization>
    <deny users="?" />
  </authorization>

  <roleManager enabled="true" />

  <globalization culture="auto"/>

</system.web>
```

Após termos atualizado nosso web.config rode a aplicação e note que a data apresentada na master page não sofre mais alterações quando mudamos de página. Agora todas as páginas de nosso sistema utilizam a mesma cultura.

Existe situações em que a cultura do browser não é suportada pelo .net framework 4.0. Neste caso a cultura utilizada é a do servidor.

Caso exista a necessidade de forçar uma cultura específica quando o .net framework não suporte a cultura do browser o parâmetro culture será configurado da seguinte maneira.

```
<globalization culture="auto:pt-BR"/>
```

No exemplo acima quando a aplicação não suportar a cultura do browser irá utilizar a cultura portuguesa do Brasil (PT-BR) como cultura default da página.

Traduzindo interfaces do usuário com o asp.net

Um aspecto fundamental para que aplicações multiculturais tenham sucesso na integração com usuários de diferentes culturas é a tradução de todas as interfaces de usuários que são apresentados pela aplicação.

O parâmetro responsável por gerenciar a linguagem de apresentação dos recursos de interface é o `UICulture`.

Podemos definir como recursos de interface do usuário propriedades como `text` e `tooltip` de uma `textBox`. O gerenciamento de `UICulture` irá buscar o valor apropriado em um arquivo de recurso e irá substituir o valor dessas propriedades com o valor contido dentro do arquivo de recurso corresponde a cultura do browser do usuário. Dessa forma é possível que um usuário com seu browser definido com o idioma em inglês visualize a aplicação em inglês e um usuário com o idioma francês visualize a aplicação em francês.

Abaixo vamos realizar um exercício prático para que possamos visualizar uma aplicação sendo apresentada em diversos idiomas. Neste caso vamos preparar algumas partes da aplicação que estamos desenvolvendo nesse curso para apresentar alguns textos de acordo com o idioma do browser do cliente.

O primeiro passo é irmos a nossa página `Default.aspx` e substituir os textos digitados na página por labels. O `aspx` da página deve ficar conforme a figura abaixo.

```
<%@ Page Title="Página Principal" Language="C#" MasterPageFile="~/Main.Master" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="CursoAspNetAvancado4._0.Default" Culture="auto" %>
<asp:Content ID="HeaderContent" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
    <h2>
        <asp:Label ID="lblBemVindo" runat="server" Text="Bem-vindo à página principal de nossa aplicação! - Data de entrada: " ></asp:Label>
        <asp:Label ID="lblData" runat="server" />
    </h2>
    <p>
        <asp:Label ID="lblIntroCurso" runat="server"
            Text="No curso de asp.net avançado iremos ver algumas das principais funcionalidades do desenvolvimento web." ></asp:Label>
    </p>
</asp:Content>
```

Agora nossa página possui duas labels (`lblBemVindo` e `lblIntroCurso`) que apresentam os textos da página em português. Apesar de termos colocado nossos textos dentro de controles `asp.net` da forma como a página está ainda não é possível gerenciar qualquer tradução para as interfaces de nossa página.

O `asp.net` disponibiliza duas formas de traduzir nossas interfaces com o usuário.

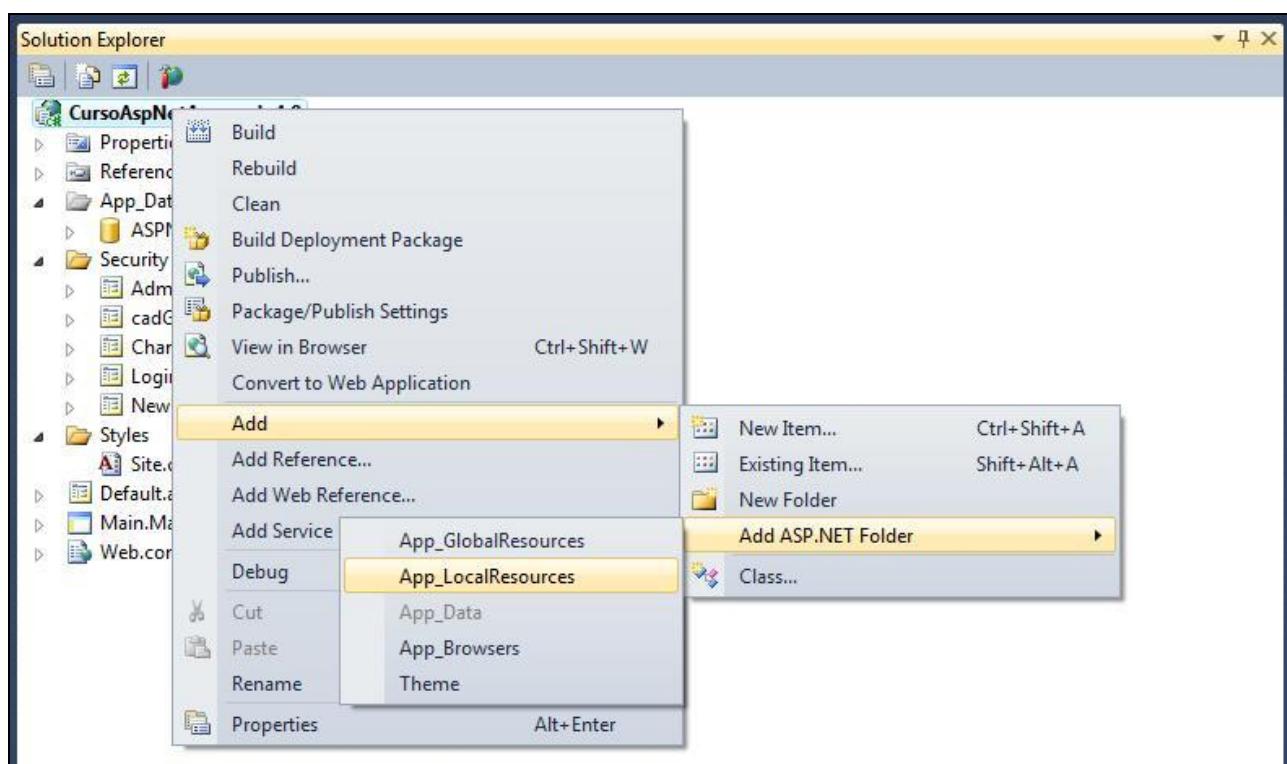
- Tradução por página;

Globalização e Localização

- Tradução global do sistema;

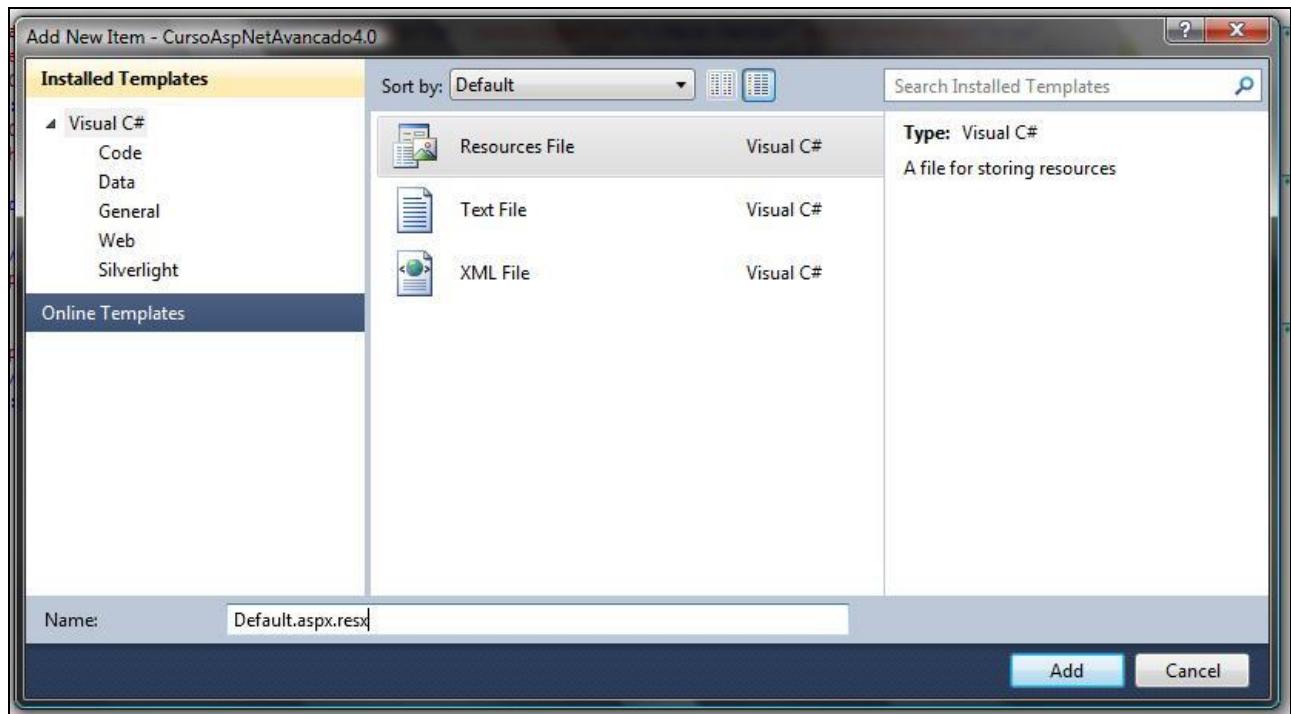
Na tradução por página devemos vincular um arquivo, que irá gerenciar o conteúdo apresentado por cada idioma, a cada webform de nossa aplicação. O arquivo responsável por gerenciar o conteúdo é do tipo resource.resx é deve ter seu nome igual ao nome da pagina acrescentado com o nome da cultura que ele gerencia. Por exemplo uma pagina chamada menu.aspx terá como um arquivo de recurso chamado menu.en.resx responsável pela tradução para o idioma inglês e um arquivo de recurso chamado menu.fr.resx responsável pela tradução para o idioma francês.

Os arquivos de recursos devem ficar contidos dentro da pasta "app_LocalResources". Essa pasta é uma pasta específica do asp.net. Para habilitarmos essa pasta devemos clicar com o botão direito em nosso web site, dentro do solution Explorer, e ir à opção add asp.net folder e selecionar a pasta app_LocalResources. Todos os arquivos de recursos para tradução por paginas deverão ser inseridos, obrigatoriamente, dentro desta pasta.



Seguindo nosso exercício agora vamos inserir arquivos de recurso para a página Default.aspx. Devemos inserir um arquivo de recurso sem fazer menção a nenhum idioma, esse arquivo de recurso será o default de nossa página, e outro arquivo de recurso referente ao idioma inglês.

O primeiro passo é inserirmos o arquivo de recurso default de nossa página (em português). Após adicionarmos a pasta app_LocalResource em nossa solução, vamos clicar com o botão direito em cima dela e adicionarmos um novo item (Add new item...). O tipo do item deve ser uma resource file e o nome do arquivo será Default.aspx.resx



Dentro do arquivo de recursos será colocado todas as chaves de tradução referente aos controles da pagina Default.aspx do nosso exemplo. O arquivo de recurso é formato por uma listagem de 3 colunas. A primeira coluna é o nome da chave de tradução, a segunda coluna será o valor da tradução e a terceira coluna é um comentário. A primeira e a segunda coluna são obrigatórias e a terceira coluna é opcional.

Obrigatoriamente o nome da chave de tradução deverá ter o seguinte formato: *nomedachave.propriedade* dessa forma é possível vincular uma propriedade a chave de tradução, informando assim qual propriedade do controle que deve ter seus valor trocado pelo valor do arquivo de recurso associado a página. Isso é importante, pois possibilita que um controle tenha diferentes propriedades suas traduzidas com diferentes valores.

Name	Value	Comment
IblBemVindo.Text	Bem-vindo à página principal de nossa aplicação! - Data de entrada:	
IblIntroCurso.Text	No curso de ASP.NET avançado iremos ver algumas das principais funcionalidades do desenvolvimento web.	
*		

No exemplo acima podemos ver o arquivo de recurso associado a pagina Default. O próximo passo é inserirmos outro arquivo de recurso, o arquivo de recurso deve ser chamado Default.aspx.en.resx, devemos inserir as mesmas chaves que inserimos no arquivo de recurso anterior, única coisa que devemos mudar são as propriedades value, que agora devem conter o texto que deve ser apresentado quando o idioma do usuário for o idioma inglês.

O arquivo de recurso em inglês deve ficar conforme a figura abaixo.

Name	Value	Comment
lblBemVindo.Text	Welcome to the home page of our application! - Login date:	
lblIntroCurso.Text	In the course of advanced asp.net we will see some key features of web development.	
*		

O próximo passo é alterar nossa pagina para que ela utilize a globalização de interface. Para que a pagina entenda que deve traduzir seus elementos de interface deve ser modificado o parâmetro UICulture da pagina para true.

```
<%@ Page Title="Página Principal" Language="C#" MasterPageFile="~/Main.Master" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="CursoAspNetAvancado4._0.Default" UICulture="auto" %>
```

O atributo UICulture também pode ser configurado dentro do web.config na tag Globalization para que ele tenha efeito em todo o sistema.

O próximo passo é vincular os controles da pagina com a sua chave do arquivo de recurso. Para isso utilizamos o atributo meta:resourceKey. Esse atributo é o elo de ligação entre o controle e o arquivo resourceKey. Quando utilizamos o meta:resourceKey não é mais necessário preencher as propriedades do controle que serão traduzidas, como por exemplo a propriedade Text.

Abaixo podemos ver como ficará a pagina Default.aspx com seus controles vinculados ao arquivo de resource através do meta:resourceKey.

```
<%@ Page Title="Página Principal" Language="C#" MasterPageFile="~/Main.Master" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="CursoAspNetAvancado4._0.Default" UICulture="auto" %>

<asp:Content ID="HeaderContent" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
<h2>
    <asp:Label ID="lblBemVindo" runat="server" meta:resourcekey="lblBemVindo" ></asp:Label>
    <asp:Label ID="lblData" runat="server" />
</h2>
<p>
    <asp:Label ID="lblIntroCurso" runat="server" meta:resourcekey="lblIntroCurso"></asp:Label>
</p>
</asp:Content>
```

Se rodarmos nossa aplicação e ficarmos alternando o idioma do browser entre português e inglês iremos ver nosso conteúdo de interface traduzido na linguagem do browser.

A outra forma de traduzir nossos elementos de interface é a tradução global do sistema. Essa forma é a mais utilizada em projetos .net devido a sua facilidade de manutenção e reaproveitamento de código. Imagine a seguinte situação: Um projeto tem aproximadamente 200 web forms diferentes e este projeto deve ser traduzido para 5 idiomas. Se utilizarmos a tradução por paginas então teríamos 5 arquivos de resources para cada pagina. Somente de arquivo de

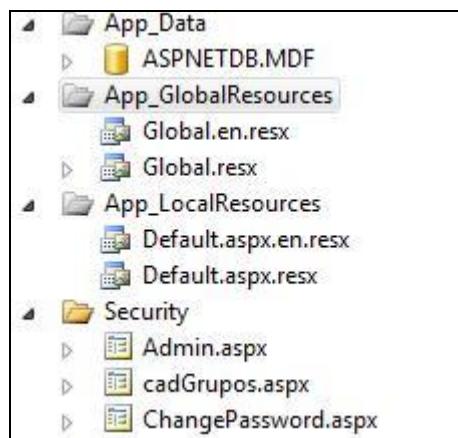
Globalização e Localização

resources teríamos 1000 arquivos em nosso projeto. A grande vantagem da tradução global do sistema é que existe um arquivo de resource por idioma e este arquivo de resource pode ser utilizado por qualquer web form de nosso projeto. Na situação hipotética que acabamos de levantar teríamos apenas 5 arquivos de resource em nosso projeto ao invés de 1000 arquivos.

Na tradução global os arquivos de resources devem ficar dentro da pasta “app_GlobalResources”. Essa pasta é uma pasta específica do asp.net. Para habilitarmos essa pasta devemos clicar com o botão direito em nosso web site, dentro do solution Explorer, e ir na opção add asp.net folder e selecionar a pasta app_GlobalResources. Todos os arquivos de recursos para tradução global deverão ser inseridos, obrigatoriamente, dentro desta pasta.

A diferença dos arquivos de resources que atuam em forma global é que a chave não pode possuir .propriedade. Elas somente irão ter um nome. Isso é necessário porque o arquivo global não fica vinculado a nenhuma página e a utilização de suas chaves deve ser configurada manualmente dentro das propriedades dos controles de interface.

Prosseguindo em nosso exercício, vamos inserir dois arquivos de resources globais dentro de nossa aplicação. Um chamado Global.resx, que será o resource default de nossa aplicação, e o arquivo Global.en.resx que será o resource global responsável pelos elementos de tradução da língua inglesa em nossa aplicação.



Em nossos arquivos de resource vamos criar uma chave chamada msgTeste. No arquivo Global.resx o valor da chave msgTeste será “Teste em português!” e no arquivo Global.en.resx o valor da chave msgTeste será “Teste em Inglês!”.

Name	Value	Comment
msgTeste	Teste em inglês	
*		

Agora vamos na pagina Default.aspx e iremos inserir uma label, logo abaixo do parágrafo de introdução do curso, com a seguinte sintaxe:

```
<p>
    <asp:Label ID="lblMsgTeste" runat="server" Text="<%$ resources:Global, msgTeste %>"></asp:Label>
</p>
```

Diferentemente da tradução por página em que o atributo meta:resourceKey se encarregava de vincular o controle com a chave do resource, na tradução global é inserido um código aspx dentro da propriedade que queremos que apresente seu valor traduzido. Na label lblMsgTeste está sendo configurado que na propriedade Text irá aparecer o valor da chave msgTeste do arquivo resource chamado Global.

Agorá vamos rodar nossa aplicação e iremos perceber que se o idioma de nosso browser for inglês será exibido na lblMsgTeste o texto “Teste em inglês!” senão, se o idioma for português ou qualquer outro, será exibido o texto “Teste em português!”.

Exercícios

1. Implementa a tradução, por página, na página de login.
2. Faça com que a nossa aplicação suporte o idioma francês.
3. Implementa tradução global na página master.

Espaço para Anotações

3. Utilizando Componentes no ASP.NET

Objetivos

Ao final deste capítulo você estará apto à:

- Trabalhar com múltiplos projetos no Visual Studio.NET
- Utilizar Dll's externas
- Instanciar classes de outros assemblies

O que são componentes?

Desde as primeiras versões das ferramentas de desenvolvimento Microsoft, uma das características mais marcantes nestas ferramentas é a capacidade de utilizarmos recursos para reutilização de código. Nas aplicações ASP.NET isto não é diferente. Quando construímos uma aplicação em ASP.NET, o ideal é tentarmos dividir a aplicação em módulos menores, a fim de tornar o gerenciamento e a manutenção da aplicação mais facilitada.

Para criarmos uma aplicação mais modularizada podemos fazer uso de componentes no ASP.NET. Os componentes são blocos de código compilados em um assembly (DLL) que podem ser utilizados em diversas aplicações. Para criarmos um componente, devemos criar um projeto do tipo Class Library. Este tipo de projeto nos permite a criação de classes que podem conter boa parte da lógica da aplicação. Desta forma, conseguimos modularizar a aplicação em diversas classes ao invés de termos toda a codificação da aplicação inserida diretamente nas páginas ASP.NET.

Utilizando uma DLL externa

Para entendermos a utilização de Dll's externas, vamos implementar a funcionalidade do Google maps em nossa aplicação.

Pege a DLL GMaps no site (<http://googlemaps.subgurim.net/descargar.aspx>) e adicione referencia em nosso projeto da Dll gmmaps.dll.

Depois disso, adiciona em sua Web.config um <AppSettings> e uma <key>

```
<appSettings>
  <add key="googlemaps.subgurim.net" value="maps"/>
</appSettings>
```

O próximo passo é criarmos um novo web form em nossa aplicação chamado "Mapa". Após criarmos o formulário vamos adicionar um item no menu principal, para que possamos navegar até esse formulário.

O próximo passo é criarmos o conteúdo do aspx de nosso formulário. Neste exercício vamos disponibilizar para o usuário uma interface onde ele possa digitar latitude, longitude e comentário de um ponto e quando ele clicar em um botão, vamos apresentar o Google Maps apontando para a localização que o usuário escolheu.

O código do aspx deve ficar conforme a imagem abaixo.

```
<%@ Page Title="" Language="C#" MasterPageFile "~/Main.Master" AutoEventWireup="true" CodeBehind="Mapa.aspx.cs"
Inherits="CursoAspNetAvancado4._0.Mapa" %>

<%@ Register Assembly="GMaps" Namespace="Subgurim.Controles" TagPrefix="Target" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">

  <p>
    <asp:Label ID="Label1" runat="server" Text="Latitude:></asp:Label>
    <asp:TextBox ID="txtLat" runat="server"></asp:TextBox>
    <br />
    <asp:Label ID="Label2" runat="server" Text="Longitude:></asp:Label>
    <asp:TextBox ID="txtLng" runat="server"></asp:TextBox>
    <br />
    <asp:Label ID="Label3" runat="server" Text="Comentário:></asp:Label>
    <asp:TextBox ID="txtComent" runat="server"></asp:TextBox>
    <br />
    <asp:Button ID="btnProcessar" runat="server" Text="Processar"
      onclick="btnProcessar_Click" />
  </p>
  <p>
    &nbsp;<Target:GMap id="mapaPrincipal" runat="server"></Target:GMap>
  </p>
</asp:Content>
```

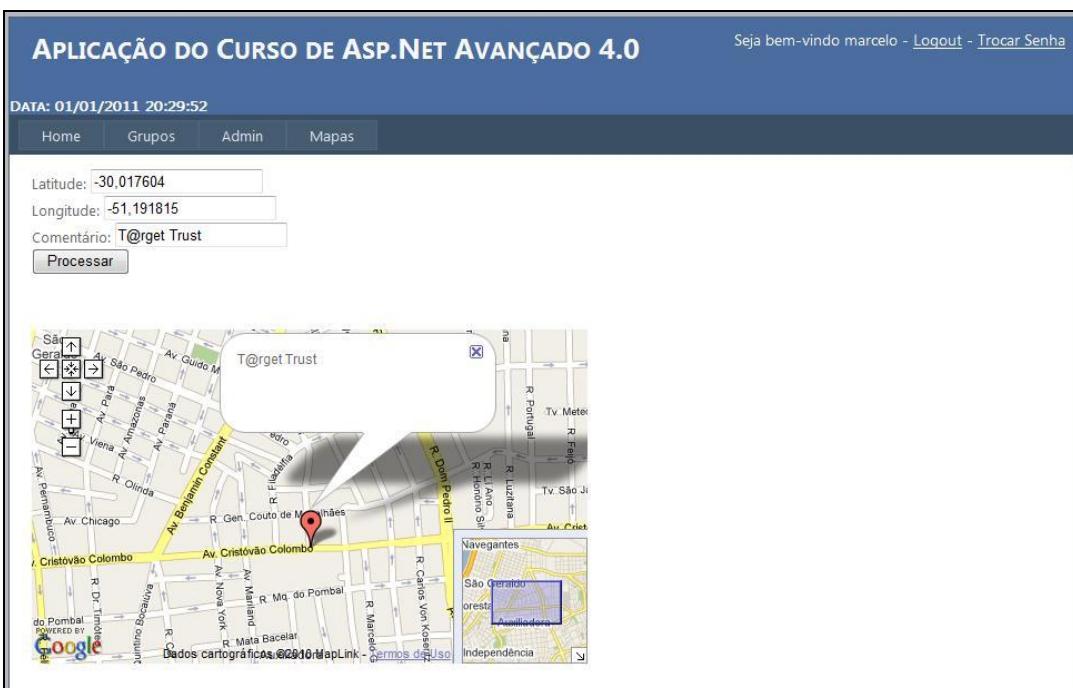
O próximo passo é inserirmos o código fonte em nosso formulário. Vamos dar dois cliques no botão processar e inserir o código a ser executado.

O código fonte total de nosso web form deve ficar conforme a imagem abaixo.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.UI;
6  using System.Web.UI.WebControls;
7  using Subgurim.Controles;
8
9  namespace CursoAspNetAvancado4._0
10 {
11     public partial class Mapa : System.Web.UI.Page
12     {
13         protected void Page_Load(object sender, EventArgs e)
14         {
15         }
16         protected void btnProcessar_Click(object sender, EventArgs e)
17         {
18             mapaPrincipal.addControl(new GControl(GControl.preBuilt.GOvewMapControl));
19             mapaPrincipal.addControl(new GControl(GControl.preBuilt.LargeMapControl));
20             GMarker ponto = new GMarker(new GLatLng(
21                 Convert.ToDouble(txtLat.Text), Convert.ToDouble(txtLng.Text)));
22             GInfoWindow comentario = new GInfoWindow(ponto, txtComent.Text.ToString());
23             mapaPrincipal.addInfoWindow(comentario);
24             GLatLng lugar_ponto = new GLatLng(
25                 Convert.ToDouble(txtLat.Text), Convert.ToDouble(txtLng.Text));
26             mapaPrincipal.setCenter(lugar_ponto, 15);
27         }
28     }
29 }
```

Ao executar nossa aplicação, no campo latitude coloque -30,017604 e no campo longitude coloque -51,191815 e no campo comentário coloque "T@rget Trust". Vamos clicar no botão processar e iremos visualizar a t@rget trust no Google maps.



Utilizando Componentes no ASP.NET

Com esse exercício podemos ver como é simples e fácil a utilização de Dll's externas em nossas aplicações .net, onde após inserirmos a referência, da dll externa, em nosso projeto podemos começar a utilizar suas classes e seus objetos livremente, da mesma forma que utilizamos as funcionalidades do .net framework.

Private Assemblys e Shared Assemblys

Se você estiver construindo um componente para um projeto do tipo Web Site, note que quando você adiciona a referência do componente ao site e compila o projeto, o Visual Studio automaticamente copia a DLL resultante da compilação do projeto para a pasta BIN do web site.

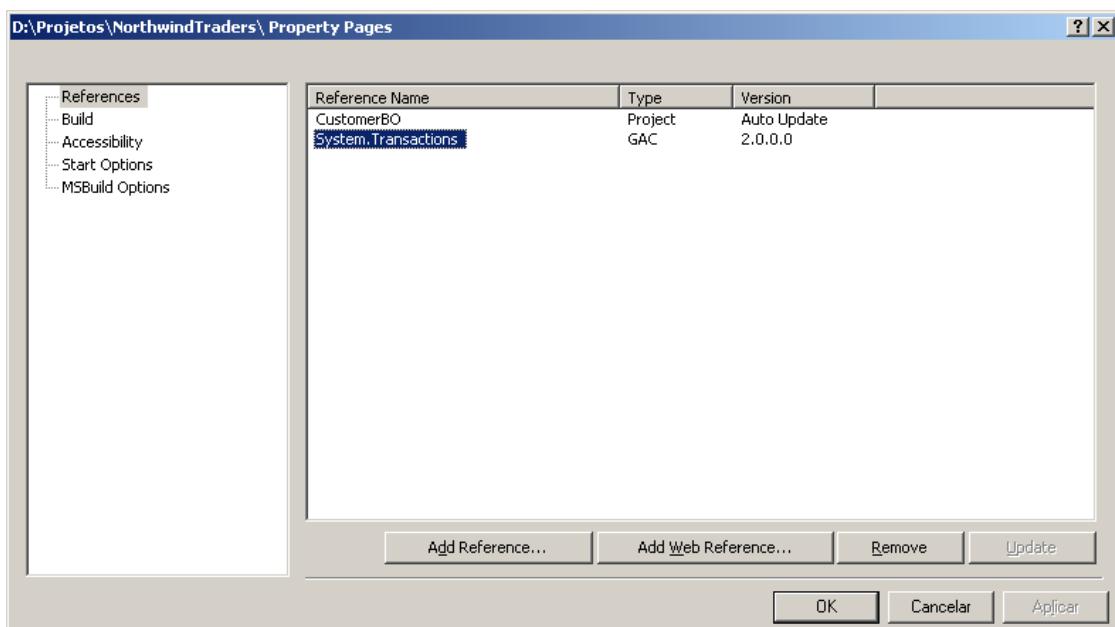
A pasta BIN é o repositório padrão para componentes de um web site. Quando o .NET Framework carrega um web site para execução ele automaticamente carrega todas as DLLs presentes na pasta BIN, considerando que estas DLLs são utilizadas pelo site para suas operações.

Devemos colocar na pasta BIN do web site todas as DLLs .NET as quais o web site faz uso, afim de que o web site funcione da maneira correta.

As dlls que se encontram na pasta BIN são chamadas private assemblys, pois são dlls utilizadas apenas para aquela aplicação, ou seja, para que a aplicação faz uso delas é preciso que elas estejam na pasta BIN da aplicação.

Analisando as referências

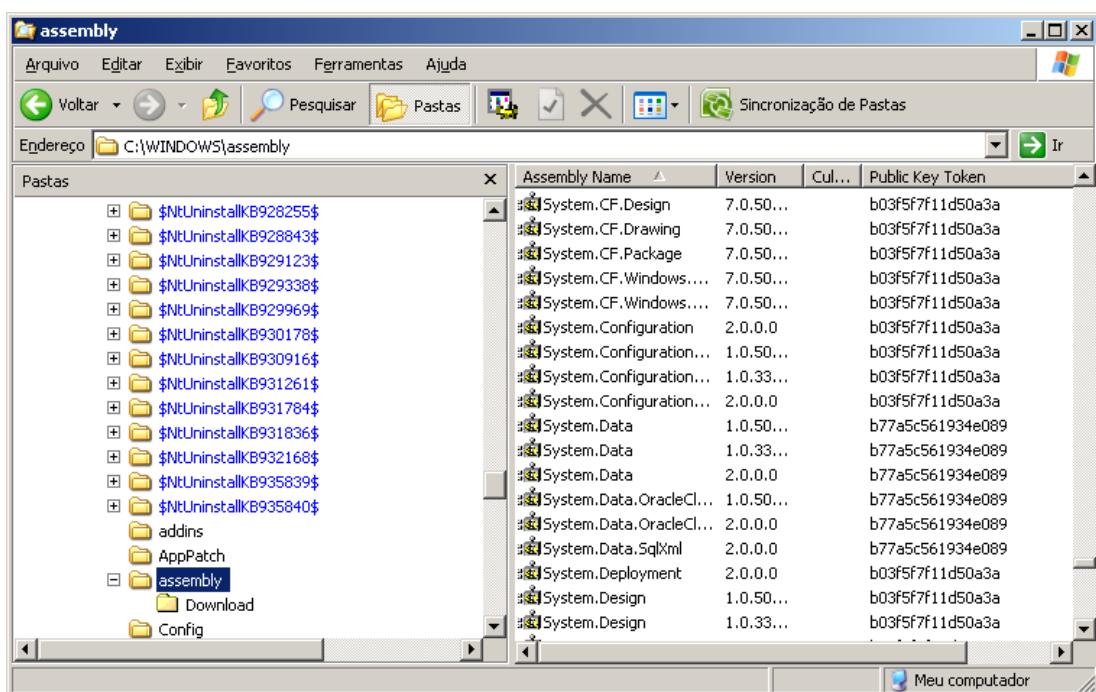
Em um projeto web, temos como verificar quais componentes nosso projeto está fazendo uso. Para isto, basta clicar com o botão direito sobre o web site no Solution Explorer e selecionar a opção “Property Pages”. A seguinte tela será apresentada.



Nesta tela podemos adicionar novas referências através do botão Add Reference.

Se tentarmos incluir um componente da aba “.NET” ao invés da aba Project ou Browse, iremos perceber um comportamento diferente para estes componentes. Se adicionarmos ao Project o componente System.Transactions por exemplo, ao compilarmos o projeto, podemos perceber que a DLL deste componente não é adicionada a pasta BIN do projeto. Isto ocorre porque estes assemblies são chamados assemblies compartilhados (shared assemblies). Tais assemblies estão disponíveis para toda a máquina na qual estão registrados. Um exemplo de shared assemblies são os próprios componentes do .NET Framework (que iniciam com prefixo System). Estes componentes não requerem uma cópia na pasta Bin projeto, pois estão disponíveis globalmente para a máquina.

Para verificar o conteúdo do Global Assembly Cache, basta abrirmos o Windows Explorer e acessarmos a pasta \WINDOWS\Assembly.



Espaço para Anotações

4. Web Parts

Objetivos

Ao final deste capítulo você estará apto à:

- Compreender as Web Parts do ASP.NET.
- Usar as Web Parts padrão em um página Web.
- Criar uma Web Part personalizada.
- Usar a Web Part personalizada em uma página da Web.

O que são web parts ?

Um dos grandes aspectos buscados por aplicações web são uma interatividade maior com o usuário, onde o usuário possa definir aspectos de customização pessoais, atingindo dessa forma uma gama maior de possíveis usuários. As customizações que os usuários podem fazer vão desde comportamentos de controles, seleção de conteúdo até o posicionamento de objetos de nossa página.

Pensando nesse aspecto a Microsoft trouxe para o .net o conceito de web parts do SharePoint. No inicio dos anos 2000 o SharePoint surgiu como uma forma altamente avançada para as organizações criarem portais e ambientes de colaboração. O sharepoint introduziu alguns componentes pré-fabricados para facilitar a criação de sites colaborativos (ao invés de criá-los desde o inicio). As paginas da Web com SharePoint são baseadas em um tipo de componente denominado Web Parts, que servem para reunir informações e funcionalidades para os usuários. Apesar das web parts do .net não serem exatamente a mesma coisa que as web parts do SharePoint, elas operam de forma muito semelhante.

As Web Parts são, de certa forma, muito parecidas com os controles personalizados, pois proporcionam uma forma de personalizar o HTML gerado pelo seu aplicativo. Diferentemente de controles personalizados que derivam da classe `System.Web.UI.Control` ou `System.Web.UI.WebControls`, as web parts derivam de `Microsoft.Sharepoint.WebPartPages.WebPart`. Embora a `WebPart` seja herdada de `System.Web.UI.Control`, ela vai além da funcionalidade de um controle convencional manipulando as interações com as classes `WebPartManager` e `WebPartZone` para dar suporte à adição, exclusão, personalização e conexão das Web Parts em uma página. As classes e membros das WebParts estão contidas dentro de `System.Web.UI.WebControls.WebParts`.

Uma grande vantagem de usá-las é que elas combinam a flexibilidade dos controles personalizados com a capacidade de gerenciamento de arrastar-soltar dos controles de usuários. Você pode arrastar WebParts concluídas das galerias e soltá-las em zonas. Também é possível modificar as propriedades compartilhadas de um grupo de WebParts e torná-las persistentes.

Dentro de nossos projetos podemos utilizar WebParts de duas formas. Através de controles ASP.NET (Label, Button, TextBox, etc) ou através da criação de classes que herdem da classe `System.Web.UI.WebControls.WebParts.WebPart`.

Os controles WebParts são úteis para desenvolver sites do tipo portal. Como os portais geralmente possuem funcionalidades muito semelhantes entre si, faz sentido criar portais a partir de uma estrutura predefinida do que criar o site do zero. A utilização de WebParts neste contexto fornece uma ótima capacidade de reutilização de código e de inserção de novas funcionalidades com um custo menor de esforço.

A arquitetura de Web Parts

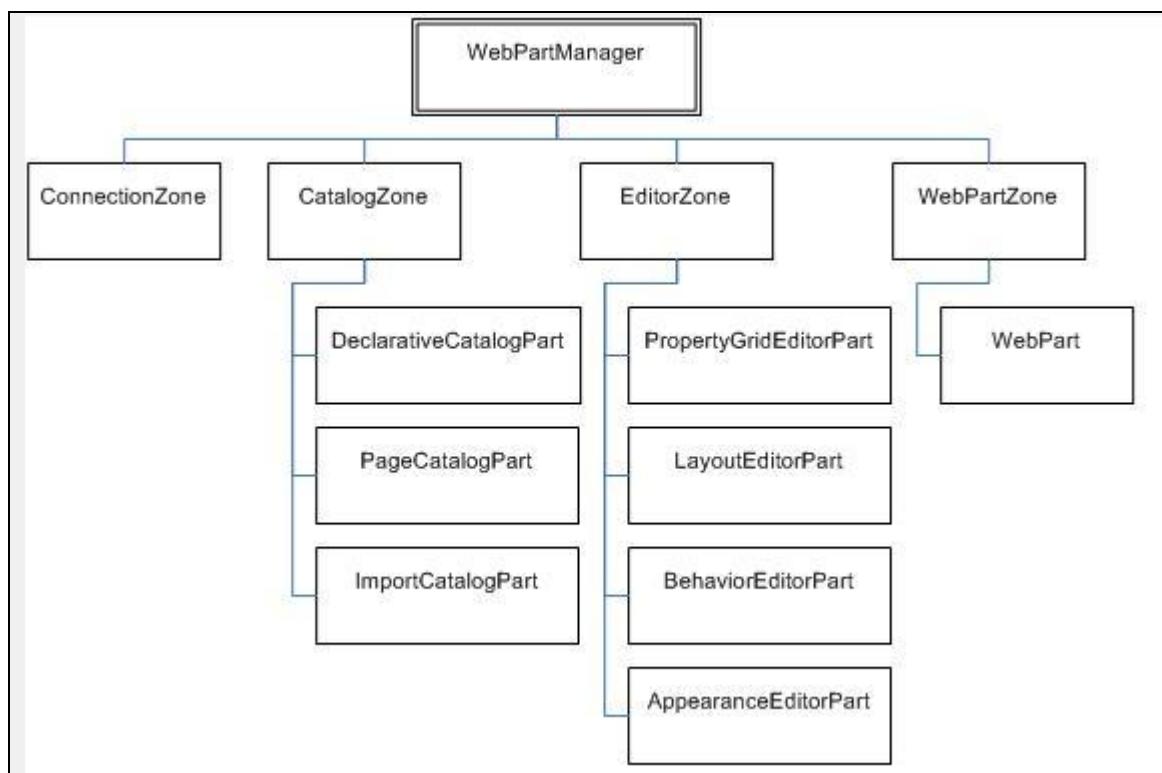
Visto que a função das Web Parts é se comportar como um meio de otimizar a interface de usuários, os componentes funcionais foram desmembrados em gerenciamento de página global e gerenciamento de zonas. Os controles WebPart precisam ser coordenados juntos. Além disso, as áreas funcionais diferentes de uma página frequentemente precisam ser manipuladas como um grupo de controles.

Em termos de classes de estruturas, as Web Parts são aninhadas em zonas, que são gerenciadas por um determinado WebPartManager que gerencia todas as zonas de WebPart de uma determinada Página.

O WebPartManager e as WebZones

Como pode ser visto na figura abaixo o WebPartManager gerencia cada WebZone, que consequentemente gerencia cada WebPart individualmente. Qualquer página que queira utilizar uma WebPart precisa de uma instância do WebPartManager. O WebPartManager é o responsável por gerenciar e coordenar a(s) zona(s) e os controles inseridos nela(s). A WebZone também gerencia qualquer elemento de interface de usuário que acompanhe o grupo de controles.

Na zona, o ZoneTemplate contém todas as Web Parts. Se um controle ASP.NET convencional estiver em um ZoneTemplate, o ASP.NET o revestirá como uma Web Part.



As zonas de Web Parts gerenciam o layout de um grupo de controles. Pronto para o uso, o ASP.NET inclui quatro zonas. São elas:

- **WebPartZone** ➔ Essa classe representa a funcionalidade básica para gerenciar controles do lado do servidor em zonas de uma página. Os controles de WebPartZone são responsáveis em hospedar tanto os controles normais do lado do servidor como também os controles WebPart. Os controles normais foram revestidos pelo controle GenericWebPart em tempo de execução para adicionar qualidades WebPart a eles.
- **CatalogZone** ➔ Essa zona hospeda os controles CatalogPart. Os catálogos geralmente gerenciam a visibilidades de partes de uma página. O controle CatalogZone mostra e oculta seu conteúdo baseado no modo de exibição selecionado pelo usuário em tempo de execução.
- **Editorzone** ➔ O controle EditorZone representa o meio através do qual os usuários finais podem modificar e personalizar as páginas da Web de acordo com suas preferências. Personalizar envolve procedimentos como configurar informações pessoais, esquemas de cores e layout. O EditorZone ajuda a gerenciar essa funcionalidade assim como a salvar e carregar essas configurações para que estejam disponíveis na próxima vez que o usuário se conectar.
- **ConnectionZone** ➔ As Web Parts frequentemente são mais úteis quando estão conectadas e se comunicam dinamicamente. O ConnectionZone gerencia essa funcionalidade.

Além de incluir várias zonas predefinidas o ASP.NET fornece alguns controles WebPart. Alguns são para gerenciar catálogos, enquanto outros são para gerenciar a edição. Eis um resumo dos controles WebPart disponibilizados pelo ASP.NET.

- **DeclarativeCatalogPart** ➔ Ao criar uma pagina WebPart, podemos adicionar as WebParts dinamicamente ou de forma declarativa. O Controle DeclarativeCatalogPart gerencia os controles do lado servidor adicionados de forma declarativa a um catálogo em uma pagina Web.
- **PageCatalogPart** ➔ Uma forma pela qual os usuários finais provavelmente desejarião personalizar um site é pela abertura e fechamento de controles. O controle PageCatalogPart representa um catalogo de página para armazenar controles que foram adicionados anteriormente a uma página que agora se encontra fechada.
- **ImportCatalogPart** ➔ Permite aos usuários importarem uma descrição Web Part de dados XML.

- **AppearanceEditorPart** → É usado para editar as propriedades de aparência de um controle WebPart ou GenericWebPart associado.
- **BehaviorEditorPart** → Oferece suporte a edição de comportamento de um controle WebPart ou GenericWebPart.
- **LayoutEditorPart** → Serve para editar as propriedades de layout de um controle WebPart ou GenericWebPart.
- **PropertyGridEditorPart** → Oferece suporte aos usuários na edição de propriedades personalizadas de controles WebPart.

Utilizando as Web Parts

Para visualizarmos o funcionamento das web parts, vamos realizar um exercício adicionando web parts em nossa aplicação. O objetivo do exercício será adicionarmos a página default duas colunas de web parts onde dentro de cada coluna irá existir hyperlinks para outros sites. Utilizando as web parts o usuário poderá configurar onde ele deseja que cada hyperlink seja apresentado e se o hyperlink deve ser apresentado.

Vamos executar os seguintes passos em nossa aplicação.

1 – Na pasta Styles vamos adicionar um arquivo css chamado WebPartsLayout.css que irá conter a configuração de css das Div que receberam as web parts. O código do arquivo css deve ficar conforme a figura abaixo.

```
.DivColunaWebPart
{
    float: left ;
    width: 49% ;
    height: 100% ;
    background-color: transparent ;
}

.DivColunaWebPart2
{
    float: left ;
    width: 49% ;
    background-color: transparent ;
    height: 100% ;
}
```

2 – Na página Default.aspx insira um controle WebPartManager, logo abaixo do parágrafo onde está a label “lblMsgTeste”. Este WebPartManager será responsável por gerenciar todas as WebParts de nossa página.

3 – Insira uma div que contenha outras duas divs dentro dela, onde cada div interna da div principal irá fazer referência as classes criadas no css. O código das divs deve ficar conforme abaixo.

```

<div>
    <div class="DivColunaWebPart">
    </div>
    <div class="DivColunaWebPart2">
    </div>
</div>

```

4 – Após termos configurado as divs, vamos arrastar para dentro de cada uma das divs internas um WebPartZone. Defina o autoFormat da primeira WebPartZone para Professional e o da segunda para Colorful.

5 – Dentro da tag ZoneTemplate de cada WebPartZone vamos inserir alguns hyperlinks. Na primeira adicione dois hyperlinks, um apontando para o site da Target Trust e o outro apontado para o site da Microsoft. Na segunda WebPartZone adicione mais dois hyperlinks, desta vez um apontando para o site da MSDN e o outro apontando para o site do Asp.Net. O código aspx resultante do passo 4 e do passo 5 pode ser visualizado na figura abaixo.

```

<div>
    <div class="DivColunaWebPart">
        <asp:WebPartZone ID="WebPartZone1" runat="server" BorderColor="#CCCCCC"
            Font-Names="Verdana" Padding="6" HeaderText="Links" HeaderStyle-ForeColor="Blue">
            <MenuLabelHoverStyle ForeColor="#E2DED6"></MenuLabelHoverStyle><MenuLabelStyle ForeColor="White"></MenuLabelStyle>
            <MenuPopupStyle BackColor="#5D7B9D" BorderColor="#CCCCCC" BorderWidth="1px" Font-Names="Verdana" Font-Size="0.6em"></MenuPopupStyle>
            <MenuVerbHoverStyle BackColor="#F7F6F3" BorderColor="#CCCCCC" BorderWidth="1px" BorderStyle="Solid" ForeColor="#333333"></MenuVerbHoverStyle>
            <MenuVerbStyle BorderColor="#5D7B9D" BorderWidth="1px" BorderStyle="Solid" ForeColor="White"></MenuVerbStyle>
            <TitleBarVerbStyle Font-Size="0.6em" Font-Underline="False" ForeColor="White"></TitleBarVerbStyle>
            <EmptyZoneTextStyle Font-Size="0.8em"></EmptyZoneTextStyle>
            <HeaderStyle HorizontalAlign="Center" Font-Size="0.7em" ForeColor="#CCCCCC"></HeaderStyle>
            <PartChromeStyle BackColor="#F7F6F3" BorderColor="#E2DED6" Font-Names="Verdana" ForeColor="White"></PartChromeStyle>
            <PartStyle Font-Size="0.8em" ForeColor="#333333"></PartStyle>
            <PartTitleStyle BackColor="#5D7B9D" Font-Bold="True" Font-Size="0.8em" ForeColor="White"></PartTitleStyle>
            <ZoneTemplate>
                <asp:HyperLink ID="HyperLink1" runat="server" Text="T@rget Trust" NavigateUrl="http://www.targettrust.com.br"></asp:HyperLink>
                <asp:HyperLink ID="HyperLink2" runat="server" Text="Microsoft" NavigateUrl="http://www.microsoft.com.br"></asp:HyperLink>
            </ZoneTemplate>
        </asp:WebPartZone>
    </div>
    <div class="DivColunaWebPart2">
        <asp:WebPartZone ID="WebPartZone2" runat="server" BorderColor="#CCCCCC" Font-Names="Verdana" Padding="6">
            <MenuLabelHoverStyle ForeColor="#FFCC66"></MenuLabelHoverStyle>
            <MenuLabelStyle ForeColor="White"></MenuLabelStyle>
            <MenuPopupStyle BackColor="#990000" BorderColor="#CCCCCC" BorderWidth="1px" Font-Names="Verdana" Font-Size="0.6em"></MenuPopupStyle>
            <MenuVerbHoverStyle BackColor="#FFFBD6" BorderColor="#FFCC66" BorderWidth="1px" BorderStyle="Solid" ForeColor="#333333"></MenuVerbHoverStyle>
            <MenuVerbStyle BorderColor="#990000" BorderWidth="1px" BorderStyle="Solid" ForeColor="White"></MenuVerbStyle>
            <TitleBarVerbStyle Font-Size="0.6em" Font-Underline="False" ForeColor="White"></TitleBarVerbStyle>
            <EmptyZoneTextStyle Font-Size="0.8em"></EmptyZoneTextStyle>
            <HeaderStyle HorizontalAlign="Center" Font-Size="0.7em" ForeColor="#CCCCCC"></HeaderStyle>
            <PartChromeStyle BackColor="#FFFBD6" BorderColor="#FFCC66" Font-Names="Verdana" ForeColor="#333333"></PartChromeStyle>
            <PartStyle Font-Size="0.8em" ForeColor="#333333"></PartStyle>
            <PartTitleStyle BackColor="#990000" Font-Bold="True" Font-Size="0.8em" ForeColor="White"></PartTitleStyle>
            <ZoneTemplate>
                <asp:HyperLink ID="HyperLink3" runat="server" Text="MSDN" NavigateUrl="http://www.msdn.com.br"></asp:HyperLink>
                <asp:HyperLink ID="HyperLink4" runat="server" Text="Asp.Net" NavigateUrl="http://www.asp.net" />
            </ZoneTemplate>
        </asp:WebPartZone>
    </div>
</div>

```

6 – Agora vamos executar nossa aplicação e poderemos ver que temos duas áreas onde os hyperlinks são apresentados e que é possível que o usuário minimize e feche os hyperlinks. Feche a aplicação e execute-a novamente e verifique como o site recorda suas ações na página. Caso seja necessário voltar para a inicialização original do site, devemos executar o comando “NomeDoWebPartManager.Personalization.ResetPersonalizationState();” dentro de um botão.

The screenshot shows a web application titled "APLICAÇÃO DO CURSO DE ASP.NET AVANÇADO 4.0". At the top, there's a header bar with the date "DATA: 05/01/2011 22:31:37" and a user "Seja bem-vindo marcelo - Logout - Trocar Senha". Below the header, a navigation bar includes "Home", "Grupos", "Admin", and "Mapas". The main content area displays a welcome message: "BEM-VINDO A PÁGINA PRINCIPAL DE NOSSA APLICAÇÃO! - DATA DE ENTRADA: 05/01/2011 22:31:37". It also contains a note: "No curso de asp.net avançado iremos ver algumas das principais funcionalidades do desenvolvimento web." and a link "Teste em português!". There are three separate sections (Web Parts) showing different display modes:

- Editado:** Shows a dropdown menu with "T@rget Trust" and a yellow box below it.
- Untitled [3]:** Shows a dropdown menu with "Asp.Net" and a yellow box below it.
- Untitled [1]:** Shows a dropdown menu with "Microsoft" and a yellow box below it.
- Untitled [2]:** Shows a dropdown menu with "MSDN" and a yellow box below it.

7 – O próximo passo é implementarmos uma forma em que possamos modificar os modos de exibição de nossas WebPartsZones. Para isso vamos adicionar um parágrafo logo abaixo de nosso WebPartManager e dentro deste parágrafo vamos inserir uma label e uma DropDownList. A drop será responsável por alterar os modos de exibição da página. O drop deve ser chamado de “DropDownListDisplayModes”. O código aspx pode ser visualizado na imagem abaixo.

```
<p>
    <asp:Label ID="lblNomDropDisplay" runat="server" Text="Troque o Display:" />
    <asp:DropDownList ID="DropDownListDisplayModes" runat="server"
        AutoPostBack="True"
        onselectedindexchanged="DropDownListDisplayModes_SelectedIndexChanged" />
</p>
```

As Web parts do ASP.NET dão suporte a cinco modos de exibição:

- **BrowseDisplayMode** → Esse é o modo normal. Não disponibiliza personalização ou edição.
- **DesignDisplayMode** → Ativa a personalização de layout arrastar e soltar.
- **EditDisplayMode** → Ativa a personalização de propriedades WebPart e permite ao usuário excluir as Web Parts.

Web Parts

- **ConnectDisplayMode** → Permite conectar as Web Parts em tempo de execução.
- **CatalogDisplayMode** → Permite adicionar as Web Parts em uma WebPartZone em tempo de execução.

8 – Agora vamos atualizar o código de nossa página Default.aspx. O código deve ficar igual ao código abaixo. No código a seguir é adicionado um membro WebPartManager denominado _wpManager à classe para armazenar a instancia do WebPartManager atual da página. É atualizado o método Page_Init para anexar um manipulador de eventos ao evento InitComplete da página. O código examina detalhadamente o WebPartManager atual em busca dos modos de exibição que recebem suporte e os coloca na DropDownList.

```
using System.Web.UI.WebControls.WebParts;
namespace CursoAspNetAvancado4._0
{
    public partial class Default : System.Web.UI.Page
    {
        //vamos adicionar um webpartmanager na pagina para pegar a instancia do webpartmanager atual
        WebPartManager _wpManager;
        protected void Page_Load(object sender, EventArgs e)
        {
            this.lblData.Text = System.DateTime.Now.ToString();
        }
        //vamos adicionar um manipular de eventos ao evento InitializationComplete
        void Page_Init(object sender, EventArgs e)
        {
            Page.InitComplete += new EventHandler(InitializationComplete);
        }
        public void InitializationComplete(object sender, System.EventArgs e)
        {
            //pega a instancia do webpartmanager da pagina
            _wpManager = WebPartManager.GetCurrentWebPartManager(Page);

            string browseModeName = WebPartManager.BrowseDisplayMode.Name;

            //vamos percorrer todos os displays modes suportados
            foreach (WebPartDisplayMode mode in _wpManager.SupportedDisplayModes)
            {
                String modeName = mode.Name;

                //certificacao para ver se o modo estah ativo antes de adicionarmos
                if (mode.IsEnabled(_wpManager))
                {
                    ListItem item = new ListItem(modeName, modeName);
                    this.DropDownListDisplayModes.Items.Add(item);
                }
            }
        }
    }
}
```

9 – Agora vamos adicionar um código no evento SelectIndexChanged do DropDownList. O código será responsável por atualizar o modo de exibição do WebPartManager.

Web Parts

```
protected void DropDownListDisplayModes_SelectedIndexChanged(object sender, EventArgs e)
{
    //pega o valor selecionado na drop
    string selectedMode = this.DropDownListDisplayModes.SelectedValue;

    //cria um display mode referente ao valor selecionado
    WebPartDisplayMode mode = _wpManager.SupportedDisplayModes[selectedMode];

    //se conseguiu criar entao atribuimos o mode
    if (mode != null)
        _wpManager.DisplayMode = mode;
}
```

10 – O próximo passo é substituir o método PreRender da página, para que sempre que ocorrer uma renderização o nosso DropDownList apresente o modo de exibição atual do WebPartManager.

```
void Page_PreRender(object sender, EventArgs e)
{
    ListItemCollection items = this.DropDownListDisplayModes.Items;
    int SelectedIndex = items.IndexOf(items.FindByText(_wpManager.DisplayMode.Name));
    this.DropDownListDisplayModes.SelectedIndex = SelectedIndex;
}
```

11 - Execute o site e note que agora podemos selecionar o modo *Design*. Posteriormente quando tivermos mais zonas na página iremos ter mais opções de seleção na DropDownList. Note que em modo *Design* agora é possível mover as Web Parts dentro de uma zona para a outra zona.

The screenshot shows the main page of the "APLICAÇÃO DO CURSO DE ASP.NET AVANÇADO 4.0". At the top, there's a header with the title and user information (Seja bem-vindo marcelo - Logout - Trocar Senha). Below the header, a timestamp shows the page was accessed on 06/01/2011 at 21:46:59. The main content area displays a welcome message: "BEM-VINDO A PÁGINA PRINCIPAL DE NOSSA APLICAÇÃO! - DATA DE ENTRADA: 06/01/2011 21:46:59". Below this, there's a note about the course and a "Troque o Display:" dropdown set to "Design". The page features three WebPartZones:

- Links**: A zone containing two web parts: "Edited" (with a "T@rget Trust" link) and "Untitled [1]" (with a "Microsoft" link).
- Untitled [1]**: A zone containing one web part: "Untitled [1]" (with a "Microsoft" link).
- WebPartZone2**: A zone containing two web parts: "Untitled [3]" (with an "Asp.Net" link) and "Untitled [2]" (with an "MSDN" link).

12 – O próximo passo é utilizarmos o EditorZone para editarmos algumas WebParts. Abaixo das divs que estão com as webPartsZone vamos adicionar duas divs, uma dentro da outra, com a div interna utilizando a classe DivColunaWebPart. Dentro da div interna vamos adicionar um EditorZone. Na

Web Parts

ZoneTemplate do EditorZone vamos adicionar um AppearanceEditorPart. Tanto o EditorZone como o AppearanceEditorPart devem ter seus estilos setados para Profissional. O resultado desse pedaço do aspx pode ser visto na figura abaixo.

```
<div>
    <div class="DivColunaWebPart">
        <asp:EditorZone ID="EditorZone1" runat="server" BackColor="#F7F6F3"
            BorderColor="#CCCCCC" BorderWidth="1px" Font-Names="Verdana" Padding="6">
            <ZoneTemplate>
                <asp:AppearanceEditorPart ID="AppearanceEditorPart1" runat="server" />
            </ZoneTemplate>
            <EditUIStyle Font-Names="Verdana" Font-Size="0.8em" ForeColor="#333333">
            </EditUIStyle>
            <HeaderVerbStyle Font-Bold="False" Font-Size="0.8em" Font-Underline="False" ForeColor="#333333">
            </HeaderVerbStyle>
            <InstructionTextStyle Font-Size="0.8em" ForeColor="#333333">
            </InstructionTextStyle>
            <LabelStyle Font-Size="0.8em" ForeColor="#333333"></LabelStyle>
            <EmptyZoneTextStyle Font-Size="0.8em" ForeColor="#333333"></EmptyZoneTextStyle>
            <ErrorStyle Font-Size="0.8em"></ErrorStyle>
            <FooterStyle HorizontalAlign="Right" BackColor="#E2DED6"></FooterStyle>
            <HeaderStyle BackColor="#E2DED6" Font-Bold="True" Font-Size="0.8em" ForeColor="#333333">
            </HeaderStyle>
            <PartChromeStyle BorderColor="#E2DED6" BorderWidth="1px" BorderStyle="Solid">
            </PartChromeStyle>
            <PartStyle BorderColor="#F7F6F3" BorderWidth="5px"></PartStyle>
            <PartTitleStyle Font-Bold="True" Font-Size="0.8em" ForeColor="#333333">
            </PartTitleStyle>
            <VerbStyle Font-Names="Verdana" Font-Size="0.8em" ForeColor="#333333">
            </VerbStyle>
        </asp:EditorZone>
    </div>
</div>
```

Agora vamos executar nossa aplicação e você poderá ver que a opção Edit está sendo apresentada em nossa DropDownList. Ao selecioná-la será habilitado em cada web part a opção de edição. Quando utilizarmos a edição o controle EditorZone e o AppearanceEditorPart serão apresentados para interação.

13- Após termos utilizado o EditorZone, que nos permite editar aparências de webParts, vamos utilizar agora o CatalogZone que irá nos permitir incluir novas WebParts ou controles asp.net como se fossem WebParts. Logo abaixo do fechamento das utilizadas com o EditorZone, vamos adicionar outras duas divs, uma dentro da outra, com a interna utilizando a classe DivColunaWebPart. Agora adicione um CatalogZone dentro da div interna, dentro da tag ZoneTemplate, do CatalogZone, insira um controle DeclarativeCatalogPart, dentro da tag WebPartsTemplate vamos inserir o tipo de controle que desejamos adicionar como WebPart.

Neste exercício vamos colocar dentro do WebPartsTemplate uma textBox, vamos adicionar uma propriedade no textBox chamada Title (essa propriedade não é do textBox, mas a engine do CatalogZone irá utilizar o valor atribuído a essa propriedade como texto de interface a ser mostrado para os usuários. O código aspx, dessa parte, deve ficar conforme a imagem abaixo.

```
<div>
    <div class="DivColunaWebPart">
        <asp:CatalogZone ID="CatalogZone1" runat="server">
            <ZoneTemplate>
                <asp:DeclarativeCatalogPart ID="DeclarativeCatalogPart1" runat="server">
                    <WebPartsTemplate>
                        <asp:TextBox ID="TextBox1" runat="server" Title="Adicionar uma textbox:"></asp:TextBox>
                    </WebPartsTemplate>
                </asp:DeclarativeCatalogPart>
            </ZoneTemplate>
        </asp:CatalogZone>

    </div>
</div>
```

Agora vamos executar a página novamente. Alterne para Catalog Mode, marque a caixa de seleção “Adicionar uma TextBox” e adicione um controle TextBox à zona links. Dessa forma podemos ver como é possível adicionar em tempo de execução novos controles WebParts a nossa página. A Seguir iremos criar uma Web Part e iremos utilizar essa estrutura para que possamos inserir em tempo de execução a web part que criamos, que irá permitir que o usuário crie hyperlinks dinamicamente em nossa página.

Construindo Web Parts

O exercício que realizamos até agora nos mostrou como utilizar Web Parts em uma página e como alternar entre vários modos do WebPartManager em tempo de execução. Vimos como adicionar uma textBox em tempo de execução. Porém o simples fato de adicionar uma TextBox as WebPartZones não representa muita coisa. Agora na continuação do exercício, criaremos uma Web Part de hyperlinks onde o usuário irá personalizar o destino de sua navegação.

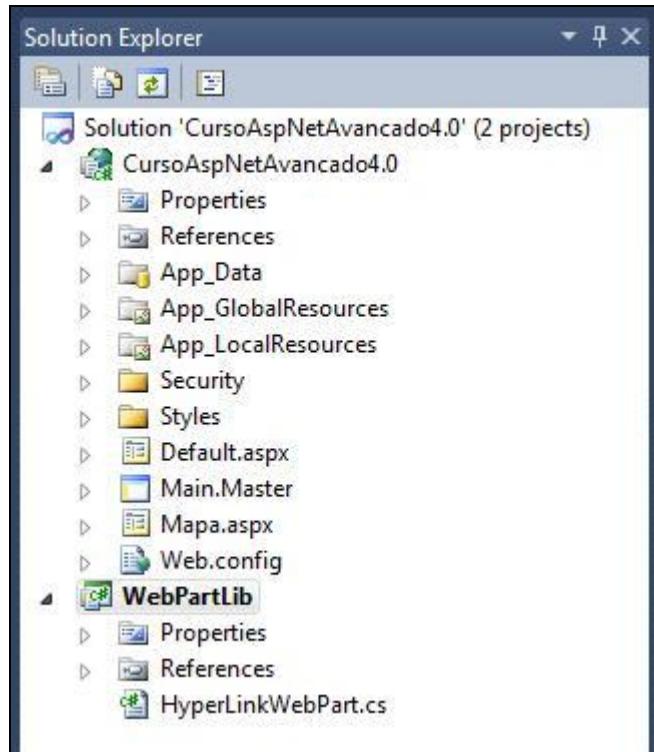
Criar uma Web Part é muito simples e bem semelhante ao desenvolvimento de controles personalizados. Para criarmos uma WebPart basta criar uma classe que derive de System.Web.UI.WebControls.WebParts.WebPart. Após isso, teremos a opção de processar HTML ou compor uma Web Part partindo de outros controles. A WebPart inclui a funcionalidade de se integrar a arquitetura da Web Part.

O próximo passo é implementarmos uma WebPart em nossa aplicação, dessa forma poderemos ver como é simples implementar uma web part com um exemplo prático. O exercício a seguir iremos criar uma Web Part de hyperlink que adicione à WebPartZone links personalizados pelo usuário. Embora possamos adicionar um controle Hyperlink convencional ao nosso catálogo, os controles normais não possuem o suporte para o usuário modificar os links, por isso o link precisa ser uma Web Part.

Vamos utilizar os seguintes passos na execução desse exercício.

Web Parts

1 – O primeiro passo é adicionar uma Dll em nossa solução e chama-lá de WebPartLib. Dentro dessa Dll iremos colocar nossa WebPart. Vamos adicionar uma classe chamada HyperLinkWebPart a nossa Dll. Essa classe será a nossa WebPart. Após termos executado essas ações nosso Solution Explorer deve estar conforme a imagem abaixo.



2 – Devemos colocar referência ao System.Web em nossa Dll. Isso é necessário porque iremos implementar funcionalidades de interfaces Web em nossa Dll.

3 – O próximo passo é adicionarmos os using na classe HyperLinkWebPart e implementarmos a herança das WebParts na classe. O código deve ficar conforme a figura abaixo.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Web;
6  using System.Web.UI;
7  using System.Web.UI.WebControls;
8  using System.Web.UI.WebControls.WebParts;
9
10
11 namespace WebPartLib
12 {
13
14     public class HyperLinkWebPart: System.Web.UI.WebControls.WebParts.WebPart
15     {
16
17     }
18
19

```

4 – A seguir vamos inserir duas variáveis privadas, do tipo string na classe HyperLinkWebPart, uma para representar o nome de exibição da Web Part e a outra para representar o URL real. Ambas serão inicializadas com valores Defaults. Após inserirmos essas duas variáveis vamos inserir outra variável privada, do tipo HyperLink, que terá como funcionalidade otimizar o controle HyperLink. Vamos sobreescrivar o método CreateChildControls para criar uma instância do controle HyperLink e adicioná-lo à coleção de controles HyperLinkWebPart. Vamos inicializar a propriedade Text do hyperlink com a variável privada que representa o nome de exibição e vamos inicializar a propriedade NavigateUrl com a variável privada que representa o URL. O código inserido deve ficar conforme a figura abaixo.

```

public class HyperLinkWebPart: System.Web.UI.WebControls.WebParts.WebPart
{
    string _strURL = "http://www.microsoft.com";
    string _strDisplayName = "Isto eh um link!";

    HyperLink _hyperlink;

    protected override void CreateChildControls()
    {
        _hyperlink = new HyperLink();
        _hyperlink.NavigateUrl = this._strURL;
        _hyperlink.Text = this._strDisplayName;
        this.Controls.Add(_hyperlink);
        base.CreateChildControls();
    }
}

```

5 – O próximo passo é expor url e nome de exibição como propriedades para que a arquitetura das web parts possam trabalhar com elas. Para permitir que as propriedades das web parts possam trabalhar com a arquitetura das web parts através do PropertyGridEditorPart, devemos acrescentar os seguintes

Web Parts

atributos as propriedades: Personalizable, WebBrowsable e WebDisplayName. O código deve ficar conforme abaixo.

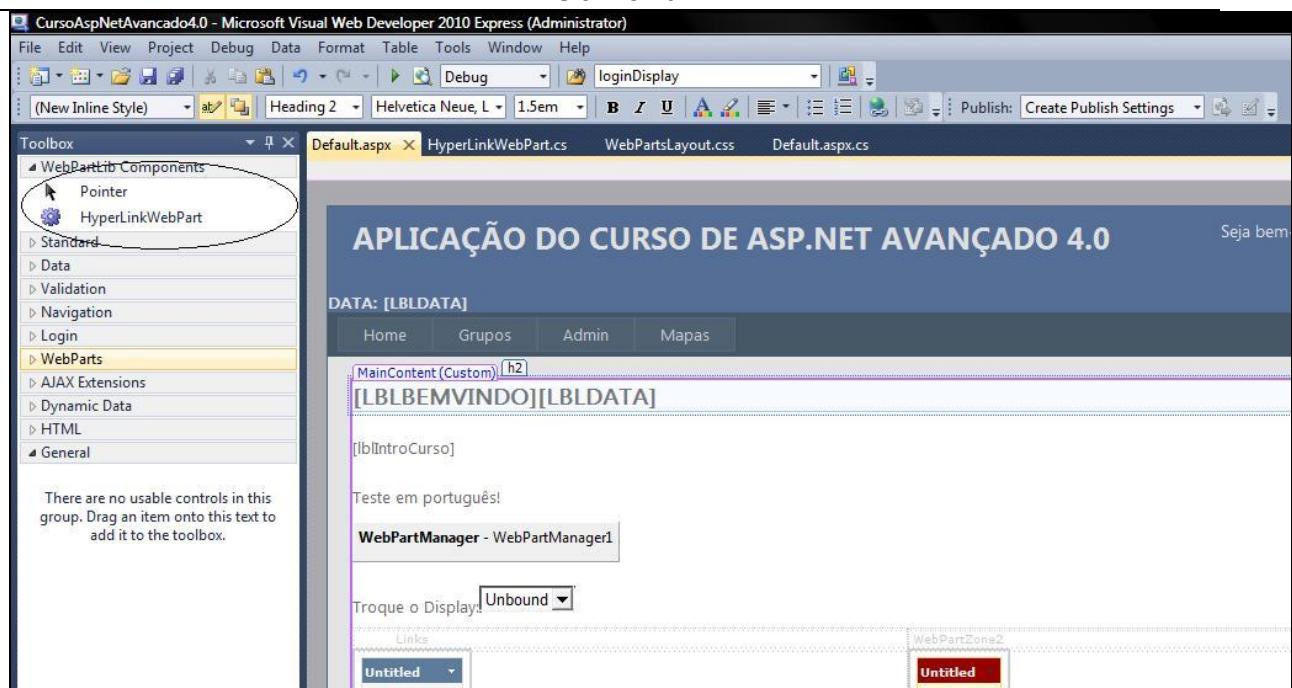
```
[Personalizable(), WebBrowsable(), WebDisplayName("Display Name")]
public string DisplayName
{
    get
    { return this._strDisplayName; }
    set
    {
        this._strDisplayName = value;
        if (_hyperlink != null)
        {
            _hyperlink.Text = this.DisplayName;
        }
    }
}

[Personalizable(), WebBrowsable(), WebDisplayName("URL")]
public string URL
{
    get
    { return this._strURL; }
    set
    {
        this._strURL = value;
        if (_hyperlink != null)
        {
            _hyperlink.NavigateUrl = this.URL;
        }
    }
}
```

Após o quinto passo nossa web part está pronta. Agora devemos compilar a DLL e adicionar a referência entre os projetos.

Após termos adicionado a referência entre os projetos note que nossa web part, HyperLinkWebPart, agora aparece no caixa de ferramentas.

Web Parts



O próximo passo agora é utilizarmos nossa web part. Para isso vamos em nossa página Default.aspx e coloque nossa CatalogZone em modo “Edit Template”. Vamos arrastar no HyperLinkWebPart para logo abaixo da textBox que inserirmos no exercício anterior. Adicione também uma propriedade Title e coloque o valor que você deseja que seja apresentado quando formos inserir nossa web part. O código aspx deve ficar conforme abaixo.

```
<div>
    <div class="DivColunaWebPart">
        <asp:CatalogZone ID="CatalogZone1" runat="server">
            <ZoneTemplate>
                <asp:DeclarativeCatalogPart ID="DeclarativeCatalogPart1" runat="server">
                    <WebPartsTemplate>
                        <asp:TextBox ID="TextBox1" runat="server" Title="Adicionar uma textbox:"></asp:TextBox>
                        <ccl:HyperLinkWebPart ID="HyperLinkWebPart1" runat="server" Title="Add a web part criada: "/>
                    </WebPartsTemplate>
                </asp:DeclarativeCatalogPart>
            </ZoneTemplate>
        </asp:CatalogZone>
    </div>
</div>
```

Adicione agora um controle PropertyGridEditorPart à EditorZone da página. Ele deve ficar logo abaixo do controle AppearanceEditorPart. O controle PropertyGridEditorPart irá prover a funcionalidade de personalização das propriedades de nossa web part.

Agora vamos rodar nossa aplicação. Selecione o modo Catalog e adicione nossa web part para a página. Após isso selecione o modo Edit, vá na nossa web part selecione Edit nela, então vá no PropertyGridEditorPart que irá aparecer no final da página e modifique os valores do display name e da URL de nossa web part, para algum site que deseja que vá a navegação. Confirme a alteração, clique em nossa web part e verifique como agora nós alteramos suas propriedades de navegação em tempo de execução.

Web Parts

Com esses exercícios pudemos ver como utilizar web parts, suas zonas e como é fácil criarmos nossas próprias web parts. Vimos também que podemos utilizar tanto controles normais como web parts junto com a arquitetura das web parts. Relembre-se as web parts são especialmente úteis para aplicativos do tipo portal por terem a habilidade de otimizar os utilitários de personalização do asp.net.

Exercícios

1. Adicione uma terceira coluna nossa aplicação e insira alguns controles para trabalharem como web part.
2. Crie uma web part onde o usuário digite uma categoria e a web part apresenta o nome do departamento que a categoria pertence (utilize nosso banco, BallonShop).

Espaço para Anotações

5. LINQ

Objetivos

Ao final deste capítulo você estará apto à:

- Entender o que é o Linq.
- Visualizar os diferentes tipos de Linq.
- Utilizar Linq to Objects;
- Entender o Linq to SQL.
- Utilizar o Linq to SQL como framework de persistência de dados.
- Implementar uma aplicação em 3 camadas com o Linq to SQL.

O que é o LINQ

LINQ (Language Integrated Query) é uma funcionalidade, incluída no framework 3.5, que tem como objetivo preencher a lacuna entre o mundo dos objetos e o mundo dos dados.

Tradicionalmente implementamos consultas a dados a partir de strings, que não possuem uma verificação do compilador em runtime e não possuem suporte ao Intellisense, que atuam em conjunto com alguma tecnologia gerenciadora da fonte de dados. Além disso, é necessário o conhecimento de cada linguagem de consulta que atua em conjunto com a fonte de dados, por exemplo, Banco de dados SQL Server, Banco de Dados Oracle, arquivos XML, Datasets, Web Services, etc. O LINQ se propõe a ser uma linguagem intermediária que forneça consultas a diferentes tipos de fonte de dados. Na utilização do LINQ são implementadas consultas em coleções de objetos fortemente tipados utilizando linguagens e operadores familiares (sintaxe muito parecida com consultas SQL de SGBD).

Resumidamente podemos definir LINQ como uma linguagem de consulta universal que atua em cima de objetos.

Todas as consultas Linq são baseadas nos tipos genéricos. Não é necessário um conhecimento profundo em genéricos, antes de começar a escrever consultas. No entanto devemos entender dois conceitos básicos.

- Quando criamos uma instância de uma classe que é uma coleção genérica, nós substituímos o “T” com o tipo de objetos que a lista conterá. Por exemplo, uma lista genérica de caracteres é expressa como `List <string>` e uma lista genérica do tipo `Customer` é expressa como `List <Customer>`. Se tentarmos adicionar um objeto `Customer` para uma lista de string iremos obter um erro de compilação. As listas genéricas não atuam com o tipo de dados `Object` e sim com o tipo de dados informados em sua criação. Uma das grandes vantagens das coleções genéricas é que não é necessário executar a conversão de tipos quando queremos utilizar seus objetos.
- `IEnumerable <T>` é a interface que permite a coleções genéricos serem enumerados, permitindo com que a coleção seja utilizada em conjunto com laços `Foreach`. Todas as coleções, mesmo as não genéricas como o `ArrayList`, possuem suporte a `IEnumerable`.

Quando implementamos uma consulta Linq sempre executamos 3 ações.

- Identificação da fonte de dados ➔ Todas as fontes de dados de consultas Linq devem implementar ou derivar de `IEnumerable` ou `IEnumerable<T>` ou uma interface derivada, como `IQueryable <T>`. Se os dados de origem não forem de um tipo passível de consulta então o provedor LINQ deve representá-lo como tal. Por exemplo, LINQ to XML carrega um documento XML em um elemento `XElement` que atua em conjunto com o LINQ. Então, caso desejemos utilizar uma fonte de dados que não seja compatível com Linq devemos implementar uma

camada intermediária que suporte uma interface genérica, para que possamos utilizar LINQ.

- Consulta → A consulta especifica quais informações devem ser recuperadas a partir de uma fonte de dados. A consulta pode também especificar como as informações são classificadas, agrupadas e filtradas. Ao decorrer deste capítulo iremos realizar um estudo detalhando sobre as consultas.
- Execução da consulta → Quando geramos a consulta apenas a armazenamos na variável de consulta. A execução real da consulta é adiada até que realizemos a interação com a consulta. Para isso temos dois tipos de execução: Execução retardada e Execução imediata.
 - Execução retardada ocorre quando executamos a consulta a partir da variável de consulta de um comando foreach.
 - Execução imediata ocorre quando utilizamos o comando `ToList<TSource>` ou `ToArrayList<TSource>` na própria criação da consulta ou quando utilizamos os comandos `Count`, `Max`, `Average` e `First` em conjunto com a variável de consulta.

Até o momento já ocorreram diversas implementações de LINQ, entre as muitas implementações podemos citar:

- Linq To Xml → Implementação do LINQ que atua diretamente sobre dados XML.
- Linq to Objects → Atua sobre coleções de objetos.
- Linq to Dataset → Realiza consulta sobre datasets.
- Linq to Flickr → Permite que realize consultas sobre fotos e armazene-as no web site Flickr.
- Linq to SQL → Implementação específica para interações do LINQ com o banco de dados SQL Server (iremos estudar esta implementação ainda neste capítulo).
- Linq to Entity (Entity Framework) → Trabalha com entidades relacionais de diversos bancos de dados.

Em função da arquitetura do Linq ser extremamente dinâmica sempre estaremos vendo novas implementações tanto por parte da Microsoft (Linq to Objects, Linq to XML, Entity Framework) como por parte de terceiros (Linq to Flickr).

O próximo passo, para que possamos entender a teoria do que foi dito até agora é realizarmos exemplos práticos. Vamos estudar Linq to Objects e Linq to SQL na sequência desse capítulo.

LINQ TO OBJECTS

A definição de Linq to Objects refere-se a utilizar consultas Linq com qualquer coleção de objetos que utilizam `IEnumerable` ou `IEnumerable<T>`, sem usar qualquer provedor de Linq como intermediário, como o Linq to SQL ou Linq to XML.

Podemos utilizar consultas Linq diretamente sobre coleções como `List<T>`, `Array`, `Dictionary of < TKey, TValue >` e qualquer outra forma de coleção que utilize `IEnumerable`.

De certa forma Linq to Objects representa uma nova forma de abordagem sobre os dados de coleções. Na forma antiga era necessário implementar algoritmos complexos para realizarmos buscas dentro da coleção. Já com o Linq to objects basta escrevemos uma consulta Linq para realizar uma busca, proporcionando um ganho tremendo em produtividade de código.

Além disso, podemos citar outras vantagens da utilização de Linq to Objects como:

- Uma maior legibilidade do código;
- Filtragem eficiente, bem como ordenação e agrupamento de dados com um mínimo de código implementado;
- Grande portabilidade para outras fontes de dados com pouca ou nenhuma modificação.

Resumidamente quanto mais complexa a operação executada nos dados, maior será a percepção de benefícios ao utilizar Linq to Object.

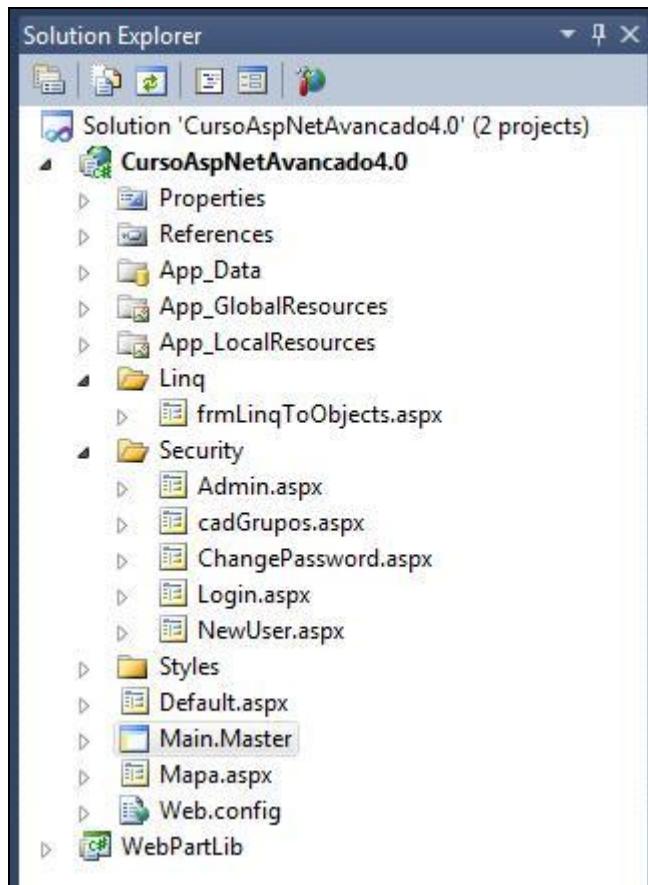
Agora vamos implementar em nossa aplicação uma página que utilize Linq to Objects como um exercício para que possamos ter uma boa compreensão de como funciona o Linq to Objects. O objetivo do exercício é implementar um array fixo dentro da página. Na página teremos dois botões. Um botão responsável por apresentar todos os itens do array e o outro botão irá apresentar os valores do array ordenados em ordem crescente e que sejam maiores que um determinado valor. Para implementarmos esse exercício vamos seguir os passos abaixo.

1 – O primeiro passo é criarmos uma pasta chamada “Linq” dentro da solution explorer de nossa aplicação, dentro desta pasta iremos inserir todas as páginas que iremos utilizar nos exercícios referentes a Linq.

Após termos inserido a pasta vamos inserir dentro da pasta uma página aspx chamada “frmLinqToObjects.aspx”. Esta página será a página que iremos implementar esse exercício.

LINQ

A solution explorer deve ficar conforme a imagem abaixo.



2 – Agora vamos inserir um item no menu de nossa aplicação, que fica na main.master, que leve o usuário a página que acabamos de criar.

3 – Vamos criar a interface de nossa página. Devemos inserir dois botões e uma textBox. O código aspx da página deve ficar conforme a figura abaixo.

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Main.Master" AutoEventWireup="true" CodeBehind="frmLinqToObjects.aspx.cs"
    Inherits="CursoAspNetAvancado4._0.Linq.frmLinqToObjects" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">

    <h2>
        Utilização de Linq to Objects
    </h2>
    <p>
        <asp:Button ID="btnSelecionar" runat="server" Text="Selecionar todos os itens" />
    </p>
    <p>
        <asp:Button ID="btnSelecionarOrdenado" runat="server"
            Text="Selecionar todos os itens ordenados" />
    </p>
    <p>
        <asp:TextBox ID="txtTexto" runat="server" TextMode="MultiLine" Height="250px"></asp:TextBox>
    </p>
</asp:Content>
```

4 – Agora o próximo passo é inserir a primeira parte do código em nossa página.

- Inserir referência a System.Text
- Declarar um array de int, chamado Números, com um tamanho de 5 posições e inicialize o array com os valores 9, 90, 1, 23 e 16. O array deve ser protected e deve ser visível em qualquer função de nossa página.

O resultado desse primeiro código deve ficar conforme a imagem abaixo.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Text;

namespace CursoAspNetAvancado4._0.Linq
{
    public partial class frmLinqToObjects : System.Web.UI.Page
    {

        protected int[] Numeros = new int [5] {9, 90, 1, 23, 16};

        protected void Page_Load(object sender, EventArgs e)
        {
        }
    }
}

```

5 – O próximo passo é adicionarmos código ao evento click do botão btnSelecionar. O código que devemos inserir pode ser visto na figura abaixo.

```

protected void btnSelecionar_Click(object sender, EventArgs e)
{
    var todosNumeros = from n in Numeros
                       select n;
    StringBuilder sb = new StringBuilder();

    foreach (int numero in todosNumeros)
    {
        sb.AppendLine(numero.ToString());
    }

    this.txtTexto.Text = sb.ToString();
}

```

Vamos analisar o código que foi inserido.

Na primeira parte do código podemos ver a consulta Linq. Note que a estrutura da consulta Linq é parecida com uma consulta SQL, a diferença é que a seção select fica no final da consulta.

A palavra var significa que o compilador irá definir qual tipo de dados terá a variável. Em nossa consulta todosNumeros que é var o compilador irá repassar para essa variável um list<int>, visto que a fonte de dados (Números) é um array de inteiros.

Na consulta Linq atribuímos Números para n e selecionamos todos os registros de n. Dessa forma retornando todos os registros do array. A seção from identifica as fontes de dados de nossa consulta e a seção select identifica o retorno da consulta.

A consulta que realizamos indica que selecionamos todos os conteúdo do array números e o retorno é todo o conteúdo do array na ordem em que está.

Após a consulta utilizamos um foreach para percorrer todos os registros e inserimos dentro de uma string builder para ser inserida, ao final do loop, dentro da textbox de resultado.

5 – O próximo passo de nosso exercício é adicionarmos código ao evento click do botão btnSeleconarOrdenado. O código que deve ser inserido pode ser visto na figura abaixo.

```
protected void btnSeleconarOrdenado_Click(object sender, EventArgs e)
{
    var todosNumeros = from n in Numeros
                        where n > 20
                        orderby n ascending
                        select n;

    StringBuilder sb = new StringBuilder();

    foreach (int numero in todosNumeros)
    {
        sb.AppendLine(numero.ToString());
    }

    this.txtTexto.Text = sb.ToString();
}
```

Podemos ver que o código é praticamente igual ao código que está no outro botão, possuindo como diferença a consulta Linq. Podemos ver que nesta segunda consulta estamos utilizando duas seções que não foram utilizadas na outra consulta: where e orderby.

Na seção where é colocada todas as restrições e filtragens de nossas consultas. Na consulta que acabamos de implementar estamos restrigindo que o valor de n deve ser maior que 20. Dessa forma a consulta irá retornar todos os valores do array “Números” que sejam maior que 20.

A seção orderby é responsável por ordenar o resultado de nossa consulta. No exercício estamos ordenando o resultado de forma ascendente (do menor para o maior).

LINQ

Após termos inseridos esses códigos, vamos salvar nossa aplicação e rodar o sistema. Poderemos ver que quando clicamos no primeiro botão é apresentado todo o conteúdo do array Números na textbox de resultado, no formato original de seus dados e quando clicamos no segundo botão (selecionar todos os itens ordenados) são apresentados ordenadamente somente os registros do array que sejam maiores que 20.

O resultado de nosso exercício pode ser visto na figura abaixo.

The screenshot shows a web application interface. At the top, a blue header bar displays the title "APLICAÇÃO DO CURSO DE ASP.NET AVANÇADO 4.0". To the right of the title, there is a welcome message "Seja bem-vindo marcelo - [Logout](#) - [Trocá Senha](#)". Below the header, a dark navigation bar contains links for "Home", "Grupos", "Admin", "Mapas", and "LINQ". The main content area has a white background and features a section titled "UTILIZAÇÃO DE LINQ TO OBJECTS". Inside this section, there are two buttons: "Selecionar todos os itens" and "Selecionar todos os itens ordenados". Below the buttons is a text area containing the following numbers:
9
90
1
23
16

Com esse exercício podemos visualizar a utilização do Linq to Objects trabalhando com um array fixo de uma página. Mas da mesma forma que trabalhamos com o array fixo podemos trabalhar com qualquer outra forma de coleção de dados que implemente a interface `IEnumerable` e suas variações.

LINQ to SQL

LINQ to SQL é uma implementação do LINQ que atua em conjunto com o SQL Server que tem como objetivo converter consultas escritas em C# para SQL dinâmico, atuando como um framework de persistência de dados, realizando o mapeamento das tabelas do banco de dados em classes. A partir dessas classes podemos realizar operações CRUD no banco de dados.

Em virtude de os fabricantes de banco de dados não adotarem o padrão ANSI o Linq to SQL atua somente com banco de dados SQL Server. Uma implementação que tem a mesma funcionalidade do LINQ to SQL que é o Entity Framework atua com multi bancos.

A primeira vista podemos pensar que é apenas mais uma linguagem que devemos aprender e que as operações de CRUD com banco de dados já realizamos com o ADO.NET, mas ao analisarmos o cenário atual de uma forma mais criteriosa podemos entender o motivo do LINQ to SQL de existir.

Como fazemos para acessar um banco de dados relacional utilizando ADO.NET mantendo o conceito de orientação a objetos? Geralmente efetuamos a representação das tabelas em classes em forma que possamos criar uma camada responsável pelo banco de dados.

Realizamos isso basicamente de duas formas:

1. Realizando o mapeamento objeto-relacional em classes de negócio e acessando os dados utilizando DataReader.
2. Realizando o mapeamento objeto-relacional usando DataSets e o acesso de dados através de DataAdapters ou TableAdapters.

O objetivo do LINQ to SQL é reunir o melhor das duas opções de forma que você possa fazer o mapeamento objeto-relacional criando classes que representam as tabelas do banco de dados (inclusive mapeando stored procedures como métodos) e com isso possibilitando a criação de consultas e alterações no banco de dados usando a sintaxe LINQ.

Internamente o LINQ to SQL utiliza ADO.NET para realizar acessos ao banco de dados e operações CRUD.

Entendendo o DataContext

O LINQ to SQL possui a classe DataContext. O DataContext atua como um elo de ligação entre o banco de dados SQL Server e as classes das entidades LINQ to SQL mapeadas do banco. O DataContext contém as informações e os métodos para efetuar a conexão com o banco e a manipulação de dados.

O DataContext possui diversos métodos que podemos utilizar, como o método SubmitChanges que envia as atualizações das classes LINQ to SQL para o banco de dados. Além disso, é permitido criar métodos dentro do DataContext para mapear stored procedures e funções de forma que executando os métodos criados as stored procedures e as funções que foram mapeadas pela classe DataContext serão executadas.

O DataContext trabalha em conjunto com o Descritor de Objeto Relacional (Object Relational Designer – O/R Designer). O Descritor de Objeto Relacional fornece uma interface visual para criar e editar Classes LINQ to SQL.

Utilizando o Descritor de Objeto Relacional podemos inserir, alterar e excluir classes e gerenciar métodos e propriedades. Todas as ações efetuadas pelo Descritor de Objeto Relacional pode ser feita gerenciando o código do DataContext, mas os ganhos em tempo que o O/R Designer nos fornece acaba fazendo com que ele seja utilizado massivamente em projetos que utilizem LINQ to SQL.

Após termos visto estas teorias vamos implementar a utilização de LINQ to SQL em nossa aplicação para que possamos ver na prática a utilização de LINQ to SQL junto com o DataContext e o O/R Designer para acessar o banco de dados e gerar algumas consultas sobre os dados do banco.

Utilizando LINQ to SQL em nossa aplicação

Agora vamos adicionar em nossa aplicação a utilização do LINQ to SQL em uma página que terá a funcionalidade de apresentar todas as categorias de livros que o banco possui e quando selecionarmos uma categoria automaticamente a página irá apresentar para o usuário um grid com todos os livros pertencentes à categoria selecionada.

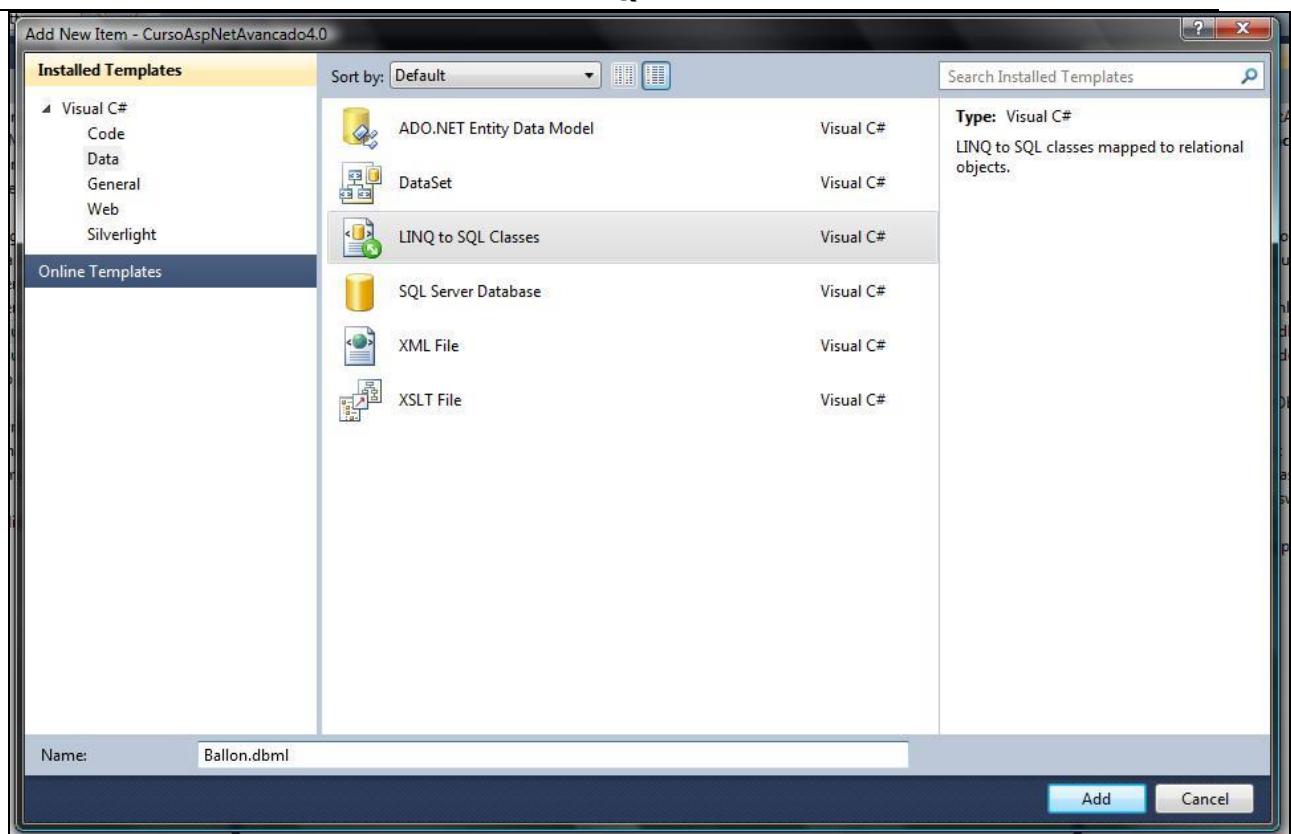
Vamos seguir os seguintes passos para executar esse exercício.

1- Adicione uma pasta chamada Data em nossa aplicação, dentro desta pasta iremos adicionar o DataContext responsável por gerenciar a ligação de nossa aplicação com o banco de dados.

2 – Configure uma conexão com o banco de dados a partir do DatabaseExplorer.

3 – Adicione dentro da pasta Data um DataContext, para adicionar um DataContext devemos selecionar um LINQ to SQL Classes que está dentro do Template Data na tela de Inserir novo item. Vamos chamar o DataContext de Ballon.dbml (todo o DataContext possui a extensão dbml). A figura abaixo apresenta como devemos inserir um DataContext.

LINQ

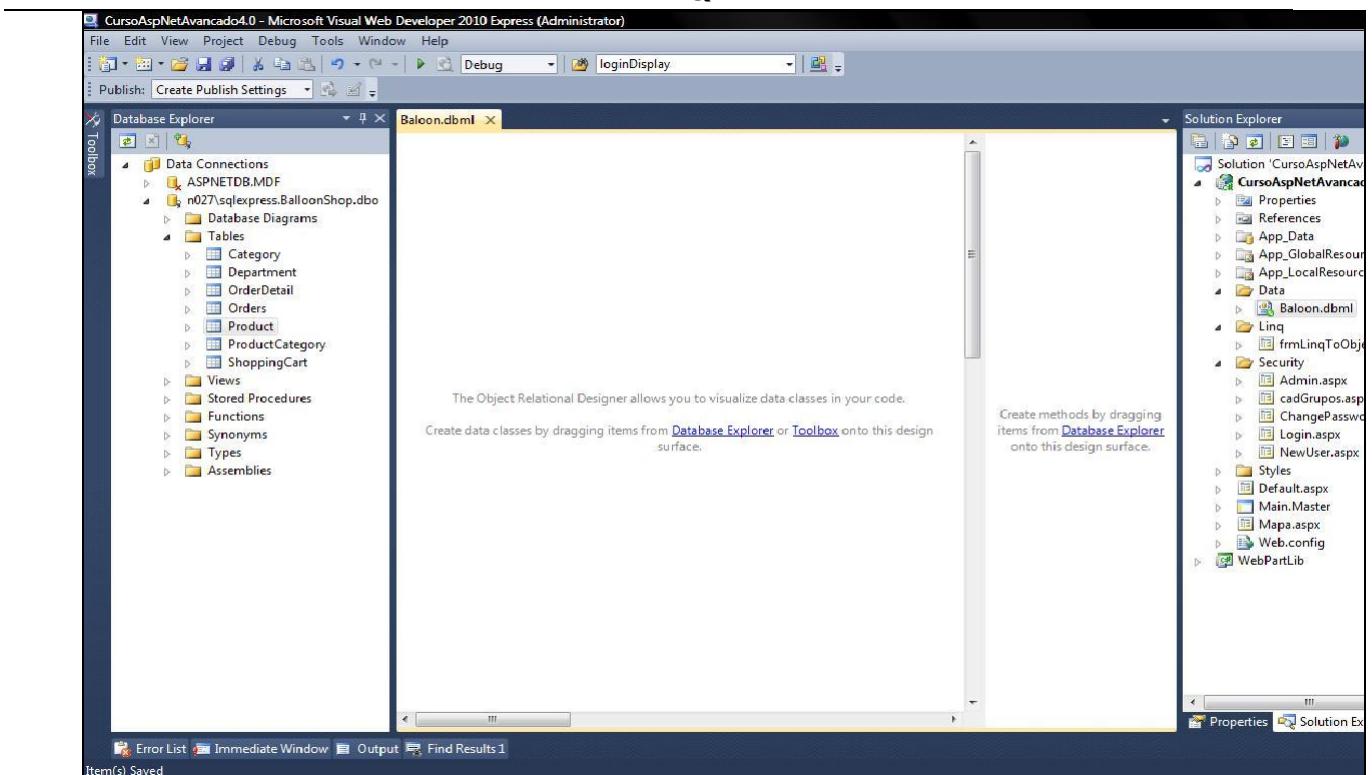


Após termos adicionado o item o Descritor de Objeto Relacional (O/R Designer) será apresentado e o arquivo LINQ to SQL Ballon.dbml vazio será inserido dentro da pasta Data.

A interface vazia corresponde ao DataContext que está pronto para ser configurado.

O O/R Designer apresenta dois painéis. O primeiro exibe as classes de entidades e o segundo exibe os métodos do DataContext. A figura abaixo apresenta o DataContext vazio.

LINQ



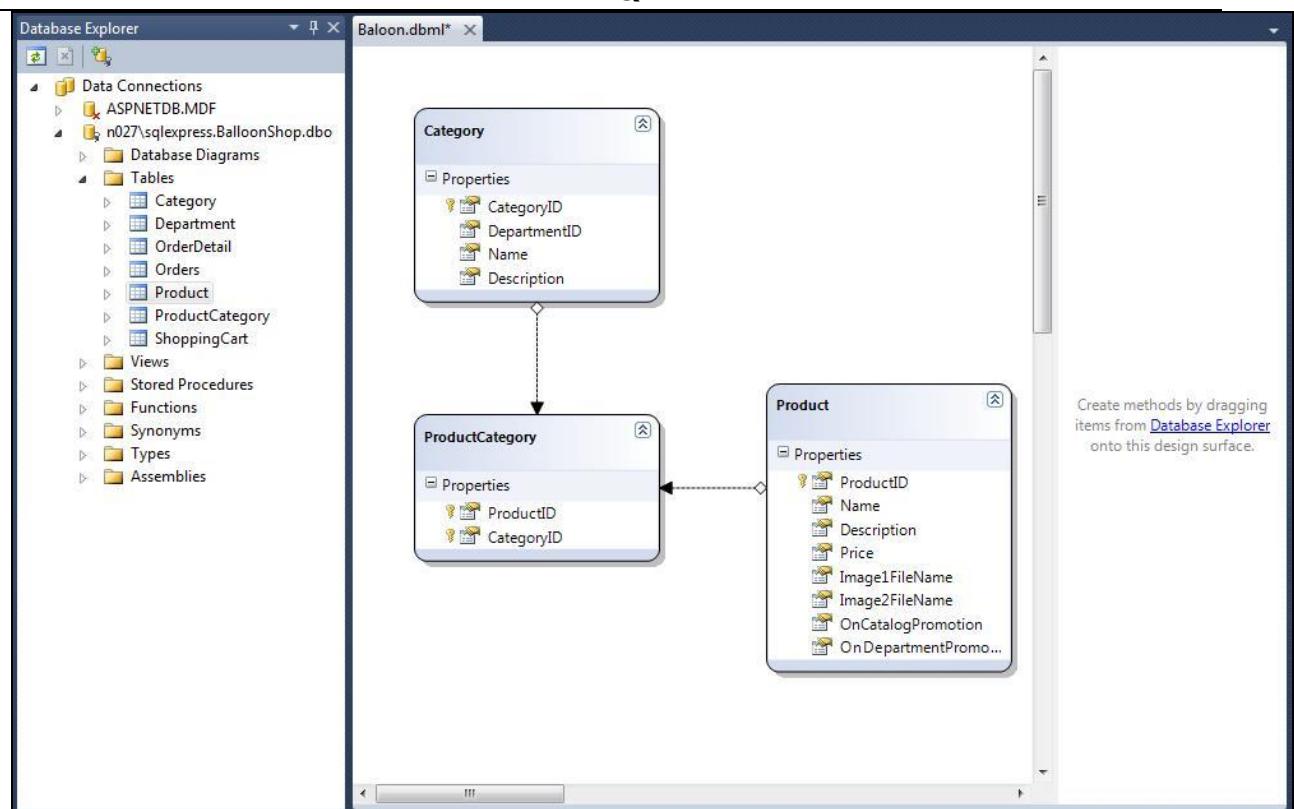
4 – O próximo passo é criarmos as classes LINQ to SQL que serão mapeadas para as tabelas do banco de dados. Para criarmos as classes basta arrastar, a partir do Database Explorer, as tabelas que queremos utilizar no DataContext. Quando arrastamos a tabela, automaticamente é criada a classe de entidade que mapeia a tabela do banco de dados.

Vamo seguir os seguintes procedimentos para criar as entidades que utilizaremos neste exercício.

- Abra o Database Explorer, localize a conexão com o banco de dados BallonShop e expanda o objeto Tables.
- Arraste as tabelas Category, ProductCategory e Product para o O/R Designer e salve o DataContext.

Após termos adicionado as tabelas o nosso contexto deve estar conforme a imagem abaixo.

LINQ



5 – Após termos preparado nosso Data Context estamos pronto para utilizar as classes geradas pelo contexto em nossa aplicação. Para isso vamos adicionar uma classe chamada Categorias dentro da pasta Data e vamos criar o método getCategories. O código do método pode ser visto abaixo.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace CursoAspNetAvancado4._0.Data
{
    public class Categorias
    {
        public IEnumerable<Category> getCategories()
        {
            BalloonDataContext db = new BalloonDataContext();

            return from t in db.Categories
                   select t;
        }
    }
}
```

Ao analisarmos o método percebemos que o retorno do método é um IEnumerable<Category>, como vimos antes todo objeto que implementa a

LINQ

interface IEnumerable é um resultado Linq, então podemos retornar a própria interface genérica.

Quando criamos a variável db do tipo BalloonDataContext estamos criando uma visão do nosso contexto de dados, então ao utilizarmos a variável db iremos ter acesso as classes e métodos pertencentes ao DataContext.

E o retorno é uma consulta LINQ que usa como fonte de dados o objeto Categories que pertence ao DataContext. O objeto Categories faz o mapeamento da tabela Category do banco de dados.

6 – O próximo passo é criarmos a interface desse exercício. Adicione uma página dentro da past Linq chamada frmCategoryProducts. Dentro da página adicione uma label com o text = “Categoria:” e uma DropDownList chamada “ddlCategorias”

O código aspx da página deve ficar conforme a imagem abaixo.

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Main.Master" AutoEventWireup="true" CodeBehind="frmCategoryProducts.aspx.cs"
Inherits="CursoAspNetAvancado4._0.Linq.frmCategoryProducts" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">

    <h2>
        Utilização de Linq to Sql
    </h2>

    <p>
        <asp:Label ID="Label1" runat="server" Text="Categoria:></asp:Label>
        <asp:DropDownList ID="ddlCategorias" runat="server"
            DataSourceID="ObjectDataSource1" DataTextField="Name"
            DataValueField="CategoryID" AutoPostBack="True">
        </asp:DropDownList>
    </p>
</asp:Content>
```

7 – Na sequência vamos criar um data source para a DropDownList, o data source deve ser do tipo object e deve selecionar na opção select o método getCategories que nós criamos na classe Categories. Para que o método apareça como uma opção de seleção devemos compilar nossa aplicação.

8 – O próximo passo é criarmos uma classe chamada Produtos, dentro da pasta Linq. Nesta classe iremos inserir o método responsável por buscar os produtos vinculados com uma determinada categoria. Vamos adicionar esse método, o nome dele deve ser “getProdutosPorCategoria” e o código do método pode ser visto na figura abaixo.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace CursoAspNetAvancado4._0.Data
{
    public class Produtos
    {

        public IEnumerable<Product> getProdutosPorCategoria(int idcategoria)
        {

            BalloonDataContext db = new BalloonDataContext();

            return from p in db.Products
                   join x in db.ProductCategories on p.ProductID equals x.ProductID
                   where x.CategoryID.Equals(idcategoria)
                   orderby p.ProductID
                   select p;
        }
    }
}

```

Ao analisarmos o código podemos visualizar a utilização do join, do Where e do orderby.

A seção join faz a união entre objetos a partir de campos chaves, no nosso método estamos juntando o objeto Products com o objeto ProductCategories quando os campos ProductID de cada objeto sejam iguais.

A seção where filtra os dados a partir de uma expressão, em nosso exercício estamos filtrando o objeto Categories onde o CategoryID seja igual ao idcategoria informado na chamada do nosso método.

A seção orderby ordena os dados da consulta, no nosso caso estamos ordenando a consulta pelo campo ProductID.

9 – O próximo passo é adicionar um gridview a nossa página, vamos chamar o gridview de grdLivros e vamos atribuir algum formato ao autoformat do grid.

10 – Vamos adicionar um Data Source ao nosso grid. O data source deve utilizar objetos como fonte de dados e no select do datasource vamos selecionar o método getProdutosPorCategoria que acabamos de criar. O DataSource irá nos perguntar qual fonte ele deve utilizar para passar como parâmetro para o método. Marque que a fonte deve ser um control, selecione nossa DropDownList e a propriedade deve ser o selectvalue (essa propriedade deve ser apresentada como default).

11 – Marque a propriedade autopostback para true da dropdownlist e vamos adicionar o seguinte código no evento SelectedIndexChanged da dropdownlist.

LINQ

```
protected void ddlCategorias_SelectedIndexChanged(object sender, EventArgs e)
{
    this.grdLivros.DataBind();
}
```

12 – Crie uma opção no menu para acessar a página.

Vamos rodar nossa aplicação e note que sempre que selecionarmos uma categoria os itens do gridview irão mudar, apresentando somente os livros vinculados à categoria. Dessa forma utilizamos LINQ to SQL para acessar nosso banco de dados e apresentar para o usuário.

Podemos ver também a utilização de consultas LINQ e como podemos fazer para utilizar consultas integradas. O resultado do exercício pode ser visto abaixo.

The screenshot shows a web application interface. At the top, there's a blue header bar with the text "APLICAÇÃO DO CURSO DE ASP.NET AVANÇADO 4.0" and a user welcome message "Seja bem-vindo marcelo - Logout - Trocar Senha". Below the header is a dark navigation bar with links: Home, Grupos, Admin, Mapas, and LINQ. The main content area has a title "UTILIZAÇÃO DE LINQ TO SQL". A dropdown menu labeled "Categoria:" is set to "Cartoons". Below it is a grid view table with the following columns: ProductID, Name, Description, Price, Image1FileName, Image2FileName, OnCatalogPromotion, and OnDepartmentPromotion. The grid displays five rows of book data:

ProductID	Name	Description	Price	Image1FileName	Image2FileName	OnCatalogPromotion	OnDepartmentPromotion
21	Baby Hi Little Angel	Baby Hi Little Angel	12,9900	t115343p.jpg	115343p.jpg	<input type="checkbox"/>	<input type="checkbox"/>
25	Tweety Stars	A cute Tweety bird on a blue heart-shaped balloon with stars. Sylvester is in the background, plotting away as usual.	12,9900	t0276001.jpg	0276001.jpg	<input type="checkbox"/>	<input type="checkbox"/>
39	Toy Story	Woody and Buzz from Toy Story, on a round balloon.	12,9900	t0366101.jpg	0366101.jpg	<input checked="" type="checkbox"/>	<input type="checkbox"/>
40	Rugrats Tommy & Chucky	If you are a Rugrats fan, you'll be nuts about this purple Rugrats balloon featuring Chucky and Tommy. A definite Nickelodeon Toon favorite.	12,9900	t03944l.jpg	03944l.jpg	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Utilizando LINQ to SQL em 3 camadas

É comum termos dúvidas referentes a utilização de 3 camadas. Mas afinal o que é uma aplicação em 3 camadas? Qual o ganho que um projeto tem ao ser implementado em camadas?

O padrão de 3 camadas começou a ser utilizado a partir da necessidade de adicionar funcionalidades de negocíos, banco de dados ou interface em um sistema com o menor impacto possível nas outras funcionalidades que o sistema possui.

Quando implementamos uma arquitetura em 3 camadas nós separamos as funcionalidades da seguinte maneira:

1. Camada de Interface do Usuário (user interface) – UI – Toda e qualquer lógica de interface com o usuário (uma página web, um relatório, uma aplicação móvel, etc) é classificada dentro da camada de interface.
2. Camada de negócio (Business Logic Layer) – BLL – responsável por armazenar a lógica da aplicação.
3. Camada de acesso a dados (Data Access Layer) – DAL – camada responsável pelo acesso e persistência dos dados no banco de dados da aplicação.

Imagine que precisamos trocar o banco de dados de nossa aplicação e os comandos SQL não são compatíveis. Em um sistema que não utiliza 3 camadas poderemos ter impacto dessa alteração em qualquer parte do sistema. Já em um sistema 3 camadas o impacto ocorrerá apenas na camada de dados (DAL), facilitando e muito essa implementação.

Quando montamos nossa aplicação devemos sempre montar de uma forma que a comunicação entre as camadas seja feita de forma correta.

- A camada de dados (DAL) se comunica apenas com a camada de negócios (BLL);
- A camada de negócios (BLL) se comunica com a camada de dados (DAL) e com a camada de interface do usuário (UI);
- A camada de interface do usuário (UI) se comunica apenas com a camada de negócio (BLL).

Essas restrições são necessárias para aumentar a segurança de nossa aplicação. Imagine a seguinte situação, um hacker consegue invadir o site (que é nossa UI mesmo ele tendo acesso aos códigos da UI ele não terá nenhum

LINQ

conhecimento de nosso banco, o máximo que ele conseguirá visualizar é a chamada de métodos de nossa camada de negócio.

O LINQ to SQL entra perfeitamente na implementação da DAL, criando mais uma abstração até o banco de dados e gerenciando todo o acesso e a persistência dos dados junto com o banco.

Agora vamos passar da teoria para a prática preparando nossa aplicação para trabalhar em 3 camadas e implementando um formulário de cadastro de produtos em 3 camadas e utilizando LINQ to SQL como o responsável pela camada de dados.

Vamos seguir os seguintes passos para executar essa implementação

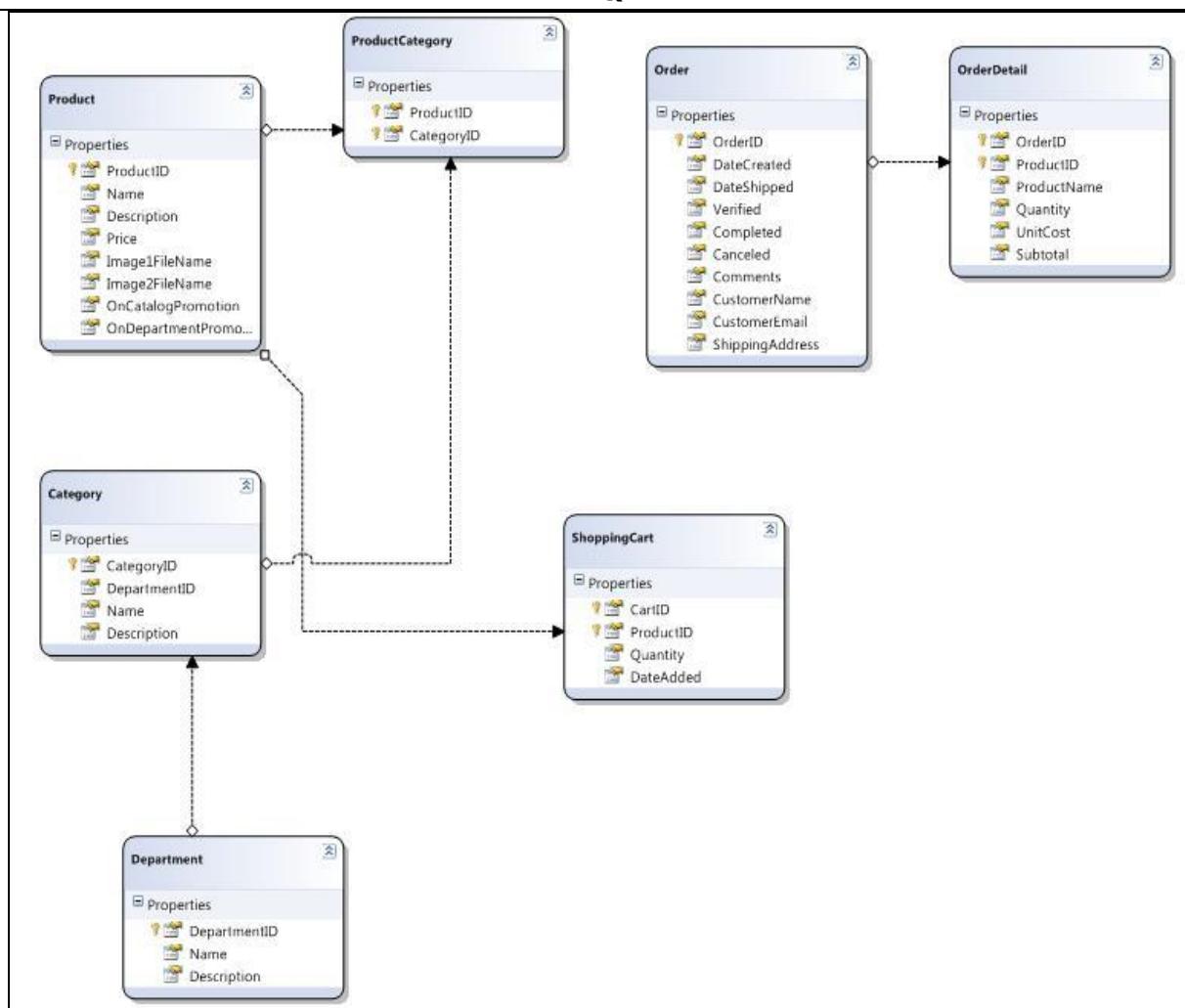
1 – Adicione uma nova Class Library em nossa aplicação chamada DAL_Balloon, dentro desta DLL iremos implementar a camada de dados.

2 – Adicione uma nova Class Library em nossa aplicação chamada BLL_Balloon, dentro desta DLL iremos implementar a camada de negócio.

3 – O próximo passo é montar as referências entre os projetos. No nosso site vamos adicionar referência a dll BLL_Balloon e na dll BLL_Balloon vamos adicionar referência a dll DAL_Balloon.

4 – Devemos inserir referência ao System.Data.Linq nos projetos BLL_Balloon e DAL_Balloon.

5 – Após termos realizados as configurações iniciais vamos montar a nossa camada de dados. Crie um DataContext na DAL_Balloon, chamado de “db_BAL.dbml” e adicione todas as tabelas do banco Balloon ao DataContext fazendo com que o contexto faça o mapeamento de todo o banco de dados. O DataContext deve ficar conforme a imagem abaixo.



Após termos atualizado o `DataContext` acabamos de terminar toda a camada de dados de nossa aplicação. Note como a utilização do LINQ to SQL facilita, e muito, a integração com banco de dados. Se fosse utilizado ADO.NET teríamos que fazer uma classe para cada tabela, deveríamos criar métodos de inclusão, alteração e deleção dos dados, propriedades, consultas, etc.

A utilização do LINQ to SQL agrega produtividade no desenvolvimento de aplicações.

6 – O próximo passo é iniciar a construção da camada de negócio, para isso vamos adicionar na DLL `BLL_Balloon` uma classe de negócio chamada “Produtos”.

7 – Insira dentro da classe `produtos` um `using` para a DLL da camada de dados, `DAL_Balloon`, e insira as seguinte propriedades.

- `ProductID` – `int`;
- `Name` – `String`;
- `Description` – `String`;
- `Price` – `decimal`;

- OnCatalogPromotion –bool;
- OnDepartmentPromotion – bool;

A classe produtos deve ficar conforme a imagem abaixo após inserirmos as propriedades.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using DAL_Balloon;

namespace BLL_Balloon
{
    public class Produtos
    {

        #region Propriedades

        public int ProductID { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public decimal Price { get; set; }
        public bool OnCatalogPromotion { get; set; }
        public bool OnDepartmentPromotion { get; set; }

        #endregion

    }
}
```

8 – Na sequência vamos criar um método que retorne todos os produtos do banco de dados com uma lista da classe produtos, da camada de negócio. Esse método irá fornecer a fonte de dados para a camada de interface utilizar no preenchimento de grids, que apresentem os produtos cadastrados. O código do método está apresentado na imagem abaixo.

LINQ

```
public List<Produtos> getProdutos()
{
    List<Produtos> listaRetorno = new List<Produtos>();

    db_DALDataContext db = new db_DALDataContext();

    var itens = from o in db.Products
                select o;

    foreach (Product item in itens)
    {
        Produtos p = new Produtos();

        p.ProductID = item.ProductID;
        p.Name = item.Name;
        p.Description = item.Description;
        p.Price = item.Price;
        p.OnCatalogPromotion = item.OnCatalogPromotion;
        p.OnDepartmentPromotion = item.OnDepartmentPromotion;

        listaRetorno.Add(p);
    }
    return listaRetorno;
}
```

9 – Após ter criado o método compile a BLL.

10 – O próximo passo é ir até o web site e criar uma página web que irá servir como um cadastro de produtos. Insira uma web page dentro da pasta Linq chamada cadProduto.aspx e crie uma opção no menu habilitando a navegação até essa página.

11 – Na sequência insira um GridView na página cadProduto, chame o grid de gridPesquisa, vincule ele a um Data Source. O DataSource deve usar objetos como fonte de dados e deve usar o método getProdutos, que nós criamos na Classe produtos da BLL como método responsável pelo Select. Após vincular o grid com o DataSource habilite paginação e seleção no GridView. Configure também a propriedade DataKeyNames com a chave “ProductID” (a chave primária da tabela de produtos).

12 – Após termos inserido o grid vamos programar os métodos para disparar o DataBind do gridview. O código deve ser inserido conforme abaixo.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        LoadGrid();
    }
}
private void LoadGrid()
{
    this.gridPesquisa.DataBind();
}
```

13 – O próximo passo é montarmos os campos da interface, para podermos inserir e alterar registros na tela de cadastro de produtos. Acima do gridView insira uma Tag <H2> e coloque um nome para a tela. Abaixo da tag <H2> insira uma tabela de 4 linhas com 4 colunas. Insira uma label indicando o campo da tela e um controle referente ao campo da tabela de produto. Utiliza Label para o productID, checkbox para os dois campos bool e textBox para os outros controles. Insira também dois botões, um botão de novo e outro botão de salvar. O código aspx deve ficar conforme a imagem abaixo.

```

<h2>
    Cadastro de Produtos
</h2>
<table class="style1">
    <tr>
        <td><asp:Label ID="Label1" runat="server" Text="ProductID:"></asp:Label></td>
        <td><asp:Label ID="lblProductID" runat="server"></asp:Label></td>
        <td><asp:Label ID="Label4" runat="server" Text="Nome:"></asp:Label></td>
        <td><asp:TextBox ID="txtNome" runat="server"></asp:TextBox></td>
    </tr>
    <tr>
        <td><asp:Label ID="Label2" runat="server" Text="Descrição:"></asp:Label></td>
        <td><asp:TextBox ID="txtDescricao" runat="server"></asp:TextBox></td>
        <td><asp:Label ID="Label5" runat="server" Text="Preço:"></asp:Label></td>
        <td><asp:TextBox ID="txtPreco" runat="server"></asp:TextBox></td>
    </tr>
    <tr>
        <td><asp:Label ID="Label3" runat="server" Text="Catalogo com promoção?"></asp:Label></td>
        <td><asp:CheckBox ID="chkCatalogo" runat="server" /></td>
        <td><asp:Label ID="Label6" runat="server" Text="Departamento com promoção?"></asp:Label></td>
        <td><asp:CheckBox ID="chkDepartamento" runat="server" /></td>
    </tr>
    <tr>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
        <td>
            <asp:Button ID="btnNovo" runat="server" Text="Novo" />
            <asp:Button ID="btnSalvar" runat="server" Text="Salvar" />
        </td>
    </tr>
</table>

```

14 – Agora vamos começar a programar nossa interface para que ela atue em conjunto com a camada de negócio. Lembre-se que a programação na camada de interface pode ter visibilidade somente em dados de interface e pode evocar funcionalidades da camada de negocio. Nunca a camada de dados pode ser usada na interface. O primeiro passo é inserirmos código no botão novo. Clique duas vezes no botão novo e adicione o código abaixo.

```

protected void btnNovo_Click(object sender, EventArgs e)
{
    newRegistro();
}

private void newRegistro()
{
    this.lblProductID.Text = "0";
    this.txtDescricao.Text = string.Empty;
    this.txtNome.Text = string.Empty;
    this.txtPreco.Text = string.Empty;
    this.chkCatalogo.Checked = false;
    this.chkDepartamento.Checked = false;
    this.txtNome.Focus();
}

```

O método do botão novo limpa todos os campos e marca o productID com “0” (zero). Utilizaremos essa informação na implementação da nossa lógica, dentro da interface.

15 – Agora vamos começar a gerar os métodos de CRUD dentro da nossa camada de negócio. O primeiro método que vamos criar será o Inserir que deve ser criado dentro da classe Produtos da nossa BLL. Crie o método conforme a imagem abaixo.

```

public void Inserir()
{
    //gera um contexto para poder trabalhar com o banco
    db_BALDataContext db = new db_BALDataContext();

    //cria um novo objeto referente a classe do Linq que faz o mapeamento da tabela product
    DAL_Balloon.Product newRegistro = new DAL_Balloon.Product();

    //preenche as propriedades do objeto com o valor das propriedades do objeto de negocio
    newRegistro.Name = this.Name;
    newRegistro.OnCatalogPromotion = this.OnCatalogPromotion;
    newRegistro.OnDepartmentPromotion = this.OnDepartmentPromotion;
    newRegistro.Price = this.Price;
    newRegistro.Description = this.Description;

    //coloca na fila para incluir o objeto no banco
    db.Products.InsertOnSubmit(newRegistro);
    //Manda atualizar o banco com todos os registros que foram colocados na fila
    db.SubmitChanges();

}

```

Ao analisarmos o código do método Inserir, podemos ver que em um primeiro momento abrimos uma conexão com o banco de dados, através de um DataContext da nossa camada de dados. A partir desse DataContext iremos mandar informações para o banco de dados. Depois geramos um objeto referente a classe que queremos inserir o registro e preenchemos as propriedades do objeto DAL com o valor das propriedades de nosso objeto de negócio.

LINQ

Na sequência inserimos nosso objeto criado em um fila de inserção a partir do método InsertOnSubmit do DataContext. E a última ação mandamos o DataContext atualizar o banco de dados com todos os registros que estavam em suas filas, a partir do método SubmitChanges.

16 – Agora vamos na nossa página, dê dois cliques no botão salvar e insira o código abaixo, que irá chamar o método de inserção da nossa camada de negócio.

```
protected void btnSalvar_Click(object sender, EventArgs e)
{
    //pega o código que está na suplier ID
    int código = int.Parse(this.lblProductID.Text);
    //se o código for igual a zero então significa que é um novo registro
    //senão significa que é um registro que está sendo alterado
    if (código.Equals(0))
    {
        InserirRegistro();
    }
    else
    {

    }
    //vamos chamar o método que atualiza o grid
    LoadGrid();
    //vamos limpar os campos
    newRegistro();
}

private void InserirRegistro()
{
    //cria uma instância da camada de negócios
    BLL_Balloon.Produtos novoRegistro = new BLL_Balloon.Produtos();
    //preenche as propriedades da camada de negócios com o valor dos controles da tela
    novoRegistro.Description = this.txtDescricao.Text;
    novoRegistro.Name = this.txtNome.Text;
    novoRegistro.OnCatalogPromotion = this.chkCatalogo.Checked;
    novoRegistro.OnDepartmentPromotion = this.chkDepartamento.Checked;
    novoRegistro.Price = Convert.ToDecimal(this.txtPreco.Text);
    //chama o método de inserção
    novoRegistro.Inserir();
}
```

No código do botão é verificado se o valor da label lblProductID é zero. Se for zero então é um registro para ser inserido, senão irá ser um registro para ser alterado (a chamada do método AlterarRegistro será criada posteriormente nesse exercício). Ao chamarmos o método inserir registro, criamos um objeto de negócio, preenchemos as propriedades dele com os valores de nossos controles de tela e chamamos o método inserir de nossa camada de negócios.

17 – Após os métodos de inclusão terem sido criados vamos criar um método na classe de negócio responsável pela alteração de registros. O código do método pode ser visto abaixo.

```

public void Alterar()
{
    //gera um contexto para poder trabalhar com o banco
    db_DALDataContext db = new db_DALDataContext();

    //busca um unico registro utilizando uma lambda expression
    Product item = db.Products.Single(x => x.ProductID.Equals(this.ProductID));

    //atualiza as propriedades do objeto com os valores do objeto de negocio
    item.Name = this.Name;
    item.OnCatalogPromotion = this.OnCatalogPromotion;
    item.OnDepartmentPromotion = this.OnDepartmentPromotion;
    item.Price = this.Price;
    item.Description = this.Description;

    //atualiza o banco
    db.SubmitChanges();
}

```

18 – Em sequência, vamos criar um método FindOne em nosso objeto de negócio, esse método será responsável por pesquisar um registro na camada de negócio e preencher as propriedades da camada de dados com o retorno da camada de negócio. O código do baixo está na imagem abaixo.

```

public void FindOne(int productid)
{
    //gera um contexto para poder trabalhar com o banco
    db_DALDataContext db = new db_DALDataContext();

    //busca um unico registro na camada de dados utilizando uma lambda expression
    Product item = db.Products.Single(x => x.ProductID.Equals(productid));

    //preenche as propriedades no objeto de negocio com o valor do objeto da camada de dados
    this.ProductID = item.ProductID;
    this.Name = item.Name;
    this.Description = item.Description;
    this.Price = item.Price;
    this.OnCatalogPromotion = item.OnCatalogPromotion;
    this.OnDepartmentPromotion = item.OnDepartmentPromotion;
}

```

19 – O próximo passo é inserirmos um tratamento no evento SelectedIndexChanged do GridView. Esse evento será responsável por chamar os métodos de seleção da BLL e preencher os controles de nossa interface com as propriedades do objeto de negócio. Visualize o código do baixo abaixo.

LINQ

```
protected void gridPesquisa_SelectedIndexChanged(object sender, EventArgs e)
{
    BLL_Balloon.Produtos pesquisa = new BLL_Balloon.Produtos();

    int codigo = int.Parse(this.gridPesquisa.SelectedDataKey.Value.ToString());

    pesquisa.FindOne(codigo);

    this.lblProductID.Text = pesquisa.ProductID.ToString();
    this.txtDescricao.Text = pesquisa.Description;
    this.txtNome.Text = pesquisa.Name;
    this.txtPreco.Text = pesquisa.Price.ToString();
    this.chkCatalogo.Checked = pesquisa.OnCatalogPromotion;
    this.chkDepartamento.Checked = pesquisa.OnDepartmentPromotion;

    this.txtNome.Focus();
}
```

20 – Agora que já consultamos registros salvos e preenchemos os controles de interface com o valor das propriedades do registro, devemos criar um método na interface que chame o método de alteração da camada de negócio. Vamos criar o método abaixo em nossa interface.

```
protected void AlterarRegistro(int ID)
{
    BLL_Balloon.Produtos item = new BLL_Balloon.Produtos();

    item.ProductID = ID;
    item.Description = this.txtDescricao.Text;
    item.Name = this.txtNome.Text;
    item.OnCatalogPromotion = this.chkCatalogo.Checked;
    item.OnDepartmentPromotion = this.chkDepartamento.Checked;
    item.Price = Convert.ToDecimal(this.txtPreco.Text);

    item.Alterar();
}
```

Após ter implementado o método alterar registro, vá ao método do botão salvar, criado anteriormente, e adicione no else a chamada para o método AlterarRegistro.

Finalmente, terminamos de implementar nosso cadastro de produtos. Rode o sistema, navegue até a página de cadastro de produtos, navegue entre os registros que são apresentados no grid, altere alguns registros e insira registros novos.

Com esse exercício implementamos todo os conceitos das 3 camadas em nossa aplicação e deixamos a carga do LINQ to SQL toda a integração e gerenciamento da base de dados, gerando assim uma abstração a mais na camada de dados.

LINQ

É importante salientar que estudamos os métodos de CRUD do LINQ to SQL, onde visualizamos que o LINQ to SQL não realiza somente consultas ao banco de dados.

No exercício não implementamos o botão excluir, mas caso queira implementar o botão o código de excluir que fica na BLL pode ser visto na imagem abaixo e o código do botão excluir você pode se basear no código de alteração para gerar a implementação.

```
public void Deletar()
{
    //abre o contexto de dados
    db_DALDataContext db = new db_DALDataContext();
    //encontra o registro que quer excluir
    Product item = db.Products.Single(x => x.ProductID.Equals(this.ProductID));
    //coloca o registro na fila de exclusão
    db.Products.DeleteOnSubmit(item);
    //deleta os registros da fila
    db.SubmitChanges();
}
```

Exercícios

1 – Gere um array de 50 numeros e implemente filtros de pesquisa com o LINQ to Object trabalhando em conjunto com controles de tela.

2 – Implemente um cadastro da tabela Department utilizando os conceitos de 3 camadas com o LINQ to SQL.

3 – Coloque no cadastro de Departamentos um gridview que apresente todas as categorias que estão vinculadas com o Departamentos.

Espaço para Anotações

6. Manutenção de Estado

Uma das características das aplicações web é que elas não mantêm seu estado. Para que elas mantenham informações utilizamos algumas técnicas entre suas requisições. O IIS por sua vez incorpora um cookie com um identificador único no cabeçalho de cada requisição http, que é enviada de volta pelo navegador. Esse cookie que torna possível identificar um mesmo usuário entre os diversos “vai e volta” de uma aplicação Web.

Desta forma a cada requisição poderíamos gravar em um banco de dados às informações importantes daquele momento, vamos supor o nome do usuário. Na próxima requisição pegamos novamente este identificador único e fariamos uma consulta em no banco de dados, já saberíamos de qual usuário se trata, poderíamos então armazenar neste momento as informações de um produto que o usuário resolver comprar e assim a vida seguiria tranquilamente.

Esta é uma forma primitiva e árdua de manter estado, existem diversas outras técnicas, todas com seus prós e contras.

Vamos estudar algumas técnicas nas próximas sessões desse capítulo. Utilizamos essas técnicas para manter valores dentro da mesma página e para passar valores entre páginas diferentes, durante as requisições que executamos em nosso sistema asp.net.

Mas antes de começarmos a estudar as técnicas vamos preparar nossa aplicação para que possamos fazer um exemplo prático e rápido de cada técnica de manutenção de estado que iremos aprender. Implemente os passos a seguir em nossa aplicação para nos preparamos para os estudos.

1 – Insira em nosso Web Site uma pasta chamada “ManEstado”.

2 – Insira dois web forms dentro da pasta criada. O primeiro chamado pagInicial e o segundo chamado pagFinal. Em todos os exercícios, que iremos fazer neste capítulo, navegaremos da pagInicial para a pagFinal utilizando técnicas diferentes para mandar informações de uma página para outra.

3 – Crie um atalho no menu para a pagInicial. Somente a pagInicial deve ter um atalho, a pagFinal não possui atalho.

Após termos preparados nossa aplicação, vamos estudar algumas técnicas de manutenção de estado.

Session

Uma das formas mais simples de manutenção de estado é através de variáveis de sessão. Cada variável está associada ao cookie de identificação do usuário. Por padrão, estas informações estão armazenadas no próprio processo do ASP.NET. Uma novidade da era .NET foi a possibilidade de armazenamento de informações de sessão em um processo separado (um servidor de estado) ou até mesmo em um SGBD. Isto trouxe novos horizontes ao desenvolvimento de aplicações Web em ASP.NET, pois simplificou a criação dos chamados Web Forms, termo que define um conjunto de servidores rodando uma aplicação específica.

Em nosso curso vamos estudar apenas o gerenciamento de sessão no próprio processo do ASP.NET (inProc), que é o comportamento padrão.

Uma variável de sessão está associada a exclusivamente a uma única sessão. Isto significa que um dado armazenado em uma variável de nome X para o usuário João não será visível na variável de sessão de mesmo nome do usuário Pedro, e vice-versa.

Sessões podem conter qualquer tipo de objeto, dessa forma podemos armazenar dentro de sessões, desde simples strings até objetos complexos de nossa aplicação, para serem utilizados por outras páginas ou pela própria página que criou a sessão.

Utilizamos o objeto Session para criar as sessões de nossa aplicação. Este objeto é o responsável por criar, alterar, excluir e manter as sessões. Utilizando ele nós conseguimos interagir com as sessões.

Para testarmos o uso de variáveis de sessão, vamos implementar seu uso em nossa aplicação. O objetivo do exercício é inserirmos um texto na pagIncial, armazernarmos esse texto em uma session, redirecionarmos a nossa página para a pagFinal e apresentarmos o valor armazenado na session em uma label. Siga os passos abaixo para executar esse exercício

1 – Insira uma label, uma textBox e um botão na página inicial. A textBox deve se chamar txtSession e o botão btnSession no campo text do botão coloque “Enviar utilizando session” e no campo Text da label insira “Digite um texto:” O código aspx deve ficar conforme a imagem abaixo.

Manutenção de Estado

```
<h2>
    Utilizacao de Session
</h2>
<p>
    <asp:Label ID="Label2" runat="server" Text="Digite um texto:"></asp:Label>
    <asp:TextBox ID="txtSession" runat="server"></asp:TextBox>
    <br />
    <asp:Button ID="btnSession" runat="server" Text="Enviar utilizando session"
        onclick="btnSession_Click" />
</p>
```

2 – Insira uma label na pagFinal.aspx, o nome da label deve ser “lblResultadoSession”. O código aspx da pagFinal pode ser visto abaixo.

```
<p>
    <asp:Label ID="lblResultadoSession" runat="server" Text=""></asp:Label>
</p>
```

3 – Dê dois cliques no botão btnSession da pagInicial e insira o código abaixo.

```
protected void btnSession_Click(object sender, EventArgs e)
{
    if (!(Session["Session_teste"] == null))
    {
        Session.Remove("Session_teste");
    }
    Session.Add("Session_teste", this.txtSession.Text.ToString());
    Response.Redirect("pagFinal.aspx");
}
```

Note que no código abaixo o primeiro procedimento é verificar se existe uma session chamada “Session_teste”. Caso exista uma session com esse nome ela é removida com o comando Remove. O próximo passo é adicionar o valor digitado na textBox “txtSession” dentro da session chamada “Session_teste” utilizando o comando Add. A ultima linha do código redireciona nossa página para a pagFinal.

4 – Agora vamos na pagFinal.aspx e vamos inserir um código no Page_Load, que deve verificar se existe a session “Session_teste” e caso ela exista deve pegar seu conteúdo e colocar dentro da label “lblResultadoSession”. O código deve ficar conforme a imagem abaixo.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Session["Session_teste"] != null)
    {
        this.lblResultado.Text = (string)Session["Session_teste"];
        Session.Remove("Session_teste");
    }
}
```

Manutenção de Estado

Após termos executado esse exercício, rode o sistema, navegue até a página inicial e envie os dados para o outro formulário. Note que é extremamente simples, armazenar e consultar valores de objetos session e entenda porque esta técnica é uma das mais utilizadas para armazenar valores, em tempo de execução, nos sistemas web.

HiddenField

Uma das formas mais simples e antigas de manter estado dentro de uma mesma página é através de campos ocultos. O valor atribuído ao controle é mantido na propriedade value entre post backs, porém não é renderizado na página. Imagine um HiddenField como uma textBox invisível que serve para armazenar textos.

ViewState

Imagine a seguinte situação: em um determinado ponto de um cadastro o usuário, após informar diversos dados cadastrais, tem que digitar o CEP para que o sistema faça a busca do endereço. O post back é acionado e quando a requisição volta ao navegador, todos os dados digitados se perderam em algum lugar do cyberspace. Claro que ninguém vai criar um aplicativo com desta forma, porém o exemplo é para lembrar que no ASP clássico e em outras diversas linguagens de programação para Web, devemos manter o estado da página manualmente, ou seja, digitando muitas linhas de código.

No ASP.NET o recurso de ViewState mantém automaticamente os valores de controles de servidor entre um post back e outro. Na verdade o ViewState internamente nada mais é do que uma campo oculto um pouco mais sofisticado. Se você rodar uma aplicação ASP.NET sem qualquer controle verá que é criado um campo oculto para o armazenamento do ViewState:

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"  
value="/wEPDwUJNzgzNDMwNTMzZGS8mO25pQR00V4s1vgSxG3dEvK+hA= = " />
```

Note que os dados não são exibidos em texto plano, por questões de segurança. Porém este é um recurso paliativo e não devemos utilizá-lo para manter dados sensíveis.

Você ainda pode usar o ViewState para adicionar “manualmente” valores ao ViewState, lembrando que você vai conseguir recuperá-los apenas entre um post back e outro na mesma página. Abaixo um pequeno exemplo de atribuição de um valor ao ViewState:

```
ViewState["Nome"] = "T@rgetTrust";
```

QueryString

Outro modelo clássico de manutenção de estado é o através do uso de querystrings, que nada mais são do que conjuntos de pares/valores anexados a URL, provavelmente você já deve ter visto dezenas de páginas usando este recurso.

Sua utilização é simples, após a URL você adiciona o primeiro valor na forma ?Chave=Valor. Para passar mais de um conjunto, os mesmos devem ser concatenados através do caractere &. Para recuperar o valor na outra página basta usar Request.QueryString.

Vamos implementar a utilização de QueryString em nossa aplicação, siga os passos a baixo para implementar o exercício.

1 – Insira uma label, um textBox e um botão na pagInicial, logo abaixo dos controles inseridos na exercício de session, vamos utilizar o mesmo modelo de exercícios do exercício de session, a única diferença será na nomeclatura dos controles. Utilize a imagem abaixo para identificar a nomeclatura dos controles.

```
<h2>
    Utilizacao de QueryString
</h2>
<p>
    <asp:Label ID="Label3" runat="server" Text="Digite um texto para a QueryString:"></asp:Label>
    <asp:TextBox ID="txtQueryString" runat="server"></asp:TextBox>
    <br />
    <asp:Button ID="btnQueryString" runat="server"
        Text="Enviar utilizando QueryString" onclick="btnQueryString_Click" />
</p>
```

2 – Insira uma label chamada lblQueryString na pagFinal.aspx, logo abaixo do parágrafo da label inserida no exercício de session. O código deve ficar conforme abaixo.

```
<p>
    <asp:Label ID="lblQueryString" runat="server" Text=""></asp:Label>
</p>
```

3 – De dois cliques no botão btnQueryString e insira o código abaixo. Note como é utilizado o queryString ao redirecionar a página.

```
protected void btnQueryString_Click(object sender, EventArgs e)
{
    Response.Redirect("pagFinal.aspx?textoEscrito=" + this.txtQueryString.Text.ToString());
}
```

4 – Vá ao evento Page_Load da pagFinal.aspx e insira o seguinte código abaixo do código inserido no exercício anterior. Note a forma que o QueryString é recuperado e tem seu conteúdo escrito na lblQueryString.

Manutenção de Estado

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Session["Session_teste"] != null)
    {
        this.lblResultado.Text = (string)Session["Session_teste"];
        Session.Remove("Session_teste");
    }

    if (Request.QueryString["textoEscrito"] != null)
    {
        this.lblQueryString.Text = Request.QueryString["textoEscrito"].ToString();
    }
}
```

Após ter inserido os código rode a aplicação, navegue até a pagIncial, escreva algum texto na textbox “txtQueryString”, clique no botão de enviar via QueryString e visualize o texto sendo apresentado na pagFinal através do envio por QueryString.

Cookies

Outra forma de manutenção de estado é através do uso de cookies, que nada mais é do que um pequeno arquivo de texto que armazenado na maquina do usuário. Sua grande vantagem é que você pode identificar o usuário mesmo dias depois de seu acesso a página. Este recurso é muito usado, pro exemplo, em sites de comércio eletrônico, para exibir as preferências do usuário os últimos produtos que ele navegou.

O grande problema dos cookies é que o usuário simplesmente pode desabilitar este recurso em seu navegador.

Application

Semelhante em diversos aspectos com variáveis de sessão, com uma importante diferença: As variáveis de aplicação são compartilhadas entre todos os usuários da aplicação. Uma variável de aplicação esta associada a toda a aplicação. Isto significa que um dado armazenado em uma variável de nome X para o usuário João será visível na variável de aplicação de mesmo nome do usuário Pedro, e vice-versa.

Outra diferença importante é que uma variável de aplicação estará disponível durante toda a vida da aplicação, enquanto uma variável de sessão será perdida ao fim da sessão do usuário.

Vamos implementar um exercício mostrando a utilização do application em nossa aplicação, a diferença desse exercício para os outros exercícios deste

Manutenção de Estado

capítulo, é que vamos utilizar somente a pagInicial. O objetivo desse exercício será armazenar um valor em um objeto application e escrever esse valor em uma label dentro da pagInicial, para mostrar que mesmo ao fecharmos a aplicação o valor do objeto application continuará nele, sendo que o objeto application só irá perder o valor ao derrubarmos o servidor do asp.net. Siga os passos abaixo para a implementação do exercício.

1 – Insira o código aspx abaixo na pagInicial. Utilize a mesma nomenclatura.

```
<h2>
    Utilizacao de Application
</h2>
<p>
    <asp:Label ID="Label4" runat="server" Text="Digite um texto para a QueryString:"></asp:Label>
    <asp:TextBox ID="txtApplication" runat="server"></asp:TextBox>
    <br />
    <asp:Button ID="btnApplication" runat="server"
        Text="Armazenar utilizando Application" onclick="btnApplication_Click" />
    <br />
    <asp:Label ID="lblResultApplication" runat="server" />
</p>
```

2 – Dê dois cliques no botão “btnApplication” e insira o código abaixo. Repare na forma de passar valor para o objeto application.

```
protected void btnApplication_Click(object sender, EventArgs e)
{
    Application["appNome"] = this.txtApplication.Text.ToString();
}
```

3 – Insira o código para gerenciar o page.LoadComplete no código da pagInicial.aspx.cs. O objetivo do código deve ser inserir o valor do application na label “lblResultApplication”. Caso o application não exista então deve ser escrito uma mensagem na label informando que o application não possui valor. O código abaixo nos mostra como fazer essa implementação.

```
void Page_LoadComplete(object sender, System.EventArgs e)
{
    if (Application["appNome"] != null)
    {
        this.lblResultApplication.Text = Convert.ToString(Application["appNome"]);
    }
    else
    {
        this.lblResultApplication.Text = "Não existe nenhum valor no application!";
    }
}
```

Rode a aplicação, insira algum valor e clique no botão. Note que a label irá apresentar o valor digitado. Feche a aplicação e entre nela novamente, note que a label continua apresentando valor. Agora feche a aplicação, derrube o servidor do asp.net e rode novamente a aplicação. Agora note que a mensagem que não existe nenhum valor no application voltou a ser exibida.

Manutenção de Estado

Você pode se perguntar como foi possível recuperar o valor já que ninguém estava usando a aplicação em determinado momento. Na verdade, mesmo que ninguém utilize a aplicação, os valores de variáveis de aplicação só serão perdidos quando a última sessão expirar.

Exercícios

1 – Utilize sessões para armazenar o DataSource de um gridview e carregue o grid view com o data source a cada requisição feita pela página.

2 – Utilize query string para definir a cor de fundo de uma determinada página, sendo que esta página pode ser chamada por outras páginas, onde a página que chama define a cor de fundo que deve ser apresentada.

3 – Utilize application, para contar quantas vezes um usuário x se loga em sua aplicação.

Espaço para Anotações

7. Web Services

Objetivos

Ao final deste capítulo você estará apto à:

- Explicar o que é um Web Service.
- Descrever o objetivo e o processo por trás da chamada de um XML Web Service a partir de um WebForm.
- Chamar um Web Service diretamente de um browser através de HTTP (Hypertext Transfer Protocol).
- Criar um Web Service.
- Consumir Web Services de terceiros.

Web Services

O que é um Web service?

Um Web service é um componente utilizado na integração de sistemas de plataformas diferentes. Os web services fornecem uma facilidade para clientes e fornecedores e podem ser vistos como um componente de alcance global.

Os web services usam o protocolo padrão HTTP (Hypertext Transfer Protocol) para transmitir dados e um formato de dados portável que é baseado no XML. Os dados são transferidos no formato XML e encapsulados pelo protocolo SOAP.

O HTTP e o XML são tecnologias padronizadas que podem ser usadas por outros ambientes de programação fora do .Net Framework, ou seja, você pode criar web services usando o VS 2010, e os aplicativos cliente que estão executando em um ambiente totalmente diferente, como Java, podem usá-los. O inverso também é possível, você pode criar web services usando Java e escrever aplicativos cliente usando C#.

A visão do cliente de um web service é a de uma interface que expõe vários métodos bem definidos. O cliente precisa chamar esses métodos usando os protocolos padrão da internet, passando os parâmetros em um formato XML, e recebendo respostas em um formato XML.

Muitos consideram um web service como uma DLL web, visto que ambos tem funcionalidades muito parecidas, tendo como diferença que as Dll's rodam dentro do computador e o web service roda em um servidor web.

O SOAP nos Web services

O SOAP (Simple Object Access Protocol) é um protocolo construído sobre o HTTP, ele permite a comunicação entre aplicações rodando em diferentes sistemas operacionais, com diferentes tecnologias e linguagens de programação.

O SOAP define uma gramática XML para especificar os nomes dos métodos que um cliente deseja invocar em um web service, definindo os parâmetros e valores de retorno. Quando um cliente invoca um web service, ele deve especificar o método e os parâmetros usando essa gramática XML.

Estrutura de um Web service

Os web services do .NET framework são similares aos aplicativos Web do ASP.NET pois consistem em duas partes: o arquivo .asmx que define o web service

é um arquivo de código C# com o sufixo .cs. O arquivo.cs contem o código c# para os métodos adicionados ao Web service.

O arquivo .asmx contém somente uma linha como a diretiva mostrada abaixo:

```
<%@ WebService Language="C#" CodeBehind="Target.asmx.cs"
Class="Service.Target" %>
```

Manipulando dados complexos

O SOAP permite que você passe estruturas de dados complexas entre um cliente e um web service como parâmetros de entrada, parâmetros de saída ou valores de retorno. Neste caso as estruturas são convertidas em um formato que pode ser transmitido pela rede e remontadas na outra extremidade, conhecidas como processo de serialização. O processo de serialização SOAP converte a classe em XML, envia a versão serializada pela rede usando SOAP e reconstrói a classe em XML na outra extremidade. Podemos também personalizar o mecanismo de serialização usando as várias classes de atributos SOAP do namespace System.Xml.Serialization.

Criando e utilizando um Web Service

Vamos implementar em nossa aplicação um web service local. Para isso vamos seguir os passos abaixo.

1 – Insira duas pastas em nosso web site. Uma pasta chamada PagesWebService, onde ficarão as páginas que implementaremos nesse capítulo, e uma pasta chamada WebServices, dentro desta pasta iremos inserir o Web Service local de nossa aplicação.

2 – Clique com o botão direito na pasta WebServices, selecione new item e insira um item do tipo WebService, o nome do item deve ser Calculo. Dessa forma estamos inserindo um web service em nossa aplicação, basta inserir o item dentro da solution, que será criado o Web Service.

3 – Apague o método default (Hello World) e insira um método de cálculo chamado soma. O código do web service pode ser visto abaixo.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;

namespace CursoAspNetAvancado4.WebServices
{
    /// <summary>
    /// Summary description for Calculo
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    public class Calculo : System.Web.Services.WebService
    {
        [WebMethod]
        public int Soma(int a, int b)
        {
            return (a + b);
        }
    }
}

```

Note que o método Soma possui um atributo WebMethod. Esse atributo torna o método disponível para acesso através da web. Note também que o namespace foi alterado para ficar mais intuitivo.

Dentro de um web service, qualquer tipo de função pode ser escrita como um método, desde um cálculo simples, como o método Soma que acabamos de implementar, até uma consulta complexa em um banco de dados.

4 – Agora vamos incluir um web form, na pasta PagesWebService, o nome do web form deve ser “UtilizandoWS.aspx”. Insira três labels, dois textbox e um Button. Utilize o código abaixo para montar seu aspx. O objetivo do web form é apresentar duas textbox para que o usuário insira o primeiro e o segundo valor a serem somados e quando for clicado no botão, será chamado o método do web form e o resultado será exibido na label lblResultado.

```
<h2>
Utilizando Web Service Local
</h2>

<p>
    <asp:Label ID="Label1" runat="server" Text="Digite o valor 1:"></asp:Label>
    <asp:TextBox ID="txtValor1" runat="server"></asp:TextBox>
    <br />
    <asp:Label ID="Label2" runat="server" Text="Digite o valor 2:"></asp:Label>
    <asp:TextBox ID="txtValor2" runat="server"></asp:TextBox>
    <br />
    <asp:Button ID="btnProcessar" runat="server" Text="Processar"
        onclick="btnProcessar_Click" />
    <br />
    <asp:Label ID="lblResultado" runat="server" Text=""></asp:Label>
</p>
```

5 – Dê dois cliques no botão btnProcessar e implemente um código que crie um objeto do tipo do web service e depois utilize esse objeto para chamar o método de soma, passando os valores das textbox para os parâmetros dos métodos. O resultado do método deve ser passado para a label lblResultado. O código abaixo apresenta como deve ser implementado o método do botão.

```
protected void btnProcessar_Click(object sender, EventArgs e)
{
    CursoAspNetAvancado4.WebServices.Calculo item = new WebServices.Calculo();

    this.lblResultado.Text = item.Soma(Convert.ToInt32(this.txtValor1.Text),
        Convert.ToInt32(this.txtValor2.Text)).ToString();

}
```

6 – Insira uma opção no menu para chamar a página, rode a aplicação e visualize o nosso web service local funcionando perfeitamente. Podemos ver como é simples implementar um web service com o .net framework 4.0. Até o framework versão 3.5 era necessário criar um projeto do tipo asp.net web service, agora com o framework 4.0 basta criar um web site normal e adicionar um item do tipo web service para dentro do web site.

É importante lembrar que os tipos utilizados como parâmetro e também os tipos utilizados como retorno do método do WebService são transformados em XML antes de serem transportados do cliente para o servidor, por tanto, devemos utilizar classes e tipos que sejam serializáveis.

Chamando um XML Web Service por HTTP

O processo de chamar um Web Service por HTTP, também chamado de acesso direto, é tipicamente usado por desenvolvedores em tempo de desenvolvimento para identificar e testar Web Services. O acesso direto permite que se veja os métodos, propriedades e o resultado de um Web Service em um ambiente amigável para o desenvolvedor.

O primeiro passo para acessar um Web Service é encontrar um. Para encontrar Web Services se utiliza serviços de busca. Esses serviços de busca estão evoluindo e mudando rapidamente conforme a tecnologia ganha aceitação na comunidade da Internet.

O processo para encontrar e utilizar um Web Service é como segue:

1. Desenvolvedores de Web Services publicam descrições e localizações para seus Web Services em um serviço universal chamado UDDI (Universal Description, Discovery and Integration).
2. Se consulta o site do UDDI para encontrar Web Services disponíveis que satisfaçam as necessidades requeridas. O site do UDDI oferece uma lista de Web Services que incluem o arquivo .disco (Discovery File - DISCO) com URLs para os Web Services.
3. Escolhe-se um Web Service e se acessa o arquivo DISCO para localizar a URL do Web Service e o documento WSDL (Web Services Description Language).
4. Constrói-se um objeto proxy a partir do documento WSDL.

Uma classe proxy é um código que se parece exatamente com a classe que ela representa; entretanto, a classe proxy não contém nenhuma lógica da aplicação. Ao invés disso, a classe proxy contém código de conversão e transporte lógico. Um objeto proxy permite que um cliente accesse um Web Service como se ele fosse um objeto COM local.

5. Usa-se o objeto proxy para chamar o Web Service
6. Usa-se o Web Service a partir de uma WebForm por meio do proxy.

A especificação do UDDI define uma maneira de publicar e procurar informações a respeito de um Web Service e a respectiva empresa que fornece o mesmo. Empresas registram individualmente informações sobre seus Web Services para que sejam utilizados por outras empresas. Após o registro, o Web Service fica disponível de forma gratuita para qualquer um que queira utilizar o serviço.

Após ter encontrado um Web Service no UDDI, usa-se a URL .asmx para navegar até a página de descrição, na qual fornece informações sobre o que o Web Service faz, os métodos que ele contém, seus parâmetros e respostas. Também é possível utilizar a página de descrição para testar as funcionalidades do Web Service.

Ao acessar a página de descrição do Web Service, o browser mostra todos os métodos disponíveis. Clicando em um método mostra os parâmetros necessário para executar tal método.

Também pode-se clicar no link “Service Description”, no topo da página de descrição, para ver o contrato WSDL, na qual contém uma descrição em XML do Web Service.

Para chamar um método do Web Service, deve-se preencher os parâmetros e clicar em “Invoke”. O WebForm passa o nome do método, os parâmetros necessários e seus respectivos valores para a URL do Web Service. O Web Service sempre retorna o resultado no formato XML.

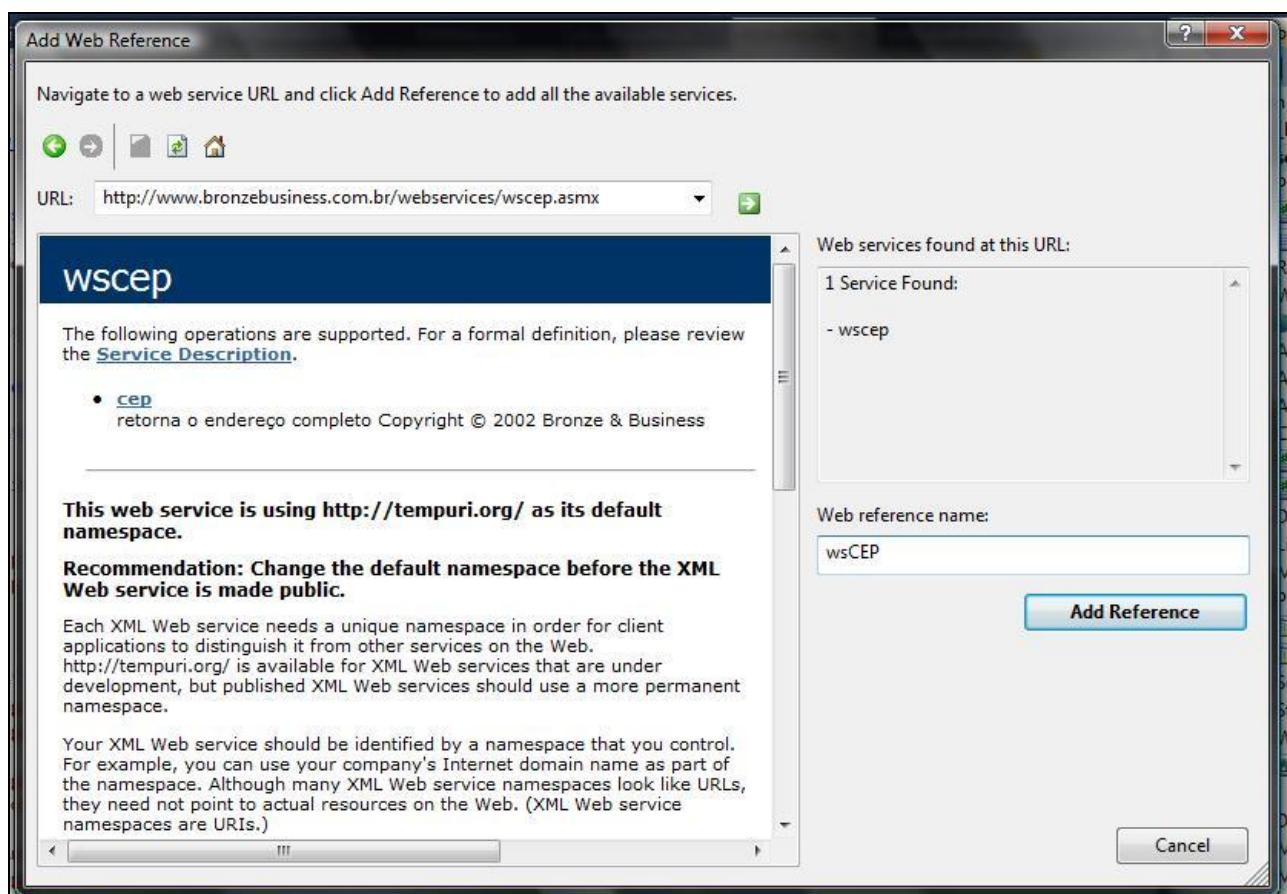
Utilizando um WebService externo

Vamos implementar em nossa aplicação a utilização de um WebService externo. Siga os passos abaixo para executar este exercício.

1 – Insira um web form, na pasta PagesWebService, chamado UtilizandoWSExterno.aspx.

2 – Clique com o botão direito no web site e selecione a opção “Add Web Reference . . .”, sempre que quisermos utilizar um web service externo em nossa aplicação devemos adicionar uma web reference.

3 – No campo url, digite o caminho do web service que deseja utilizar, neste exercício digite: <HTTP://www.bronzebusiness.com.br/webservices/wscep.asmx> e clique em Go. Após o visual Studio buscar e apresentar as informações do web service digite wsCep (nome amigável que utilizaremos em nosso código para chamar os objetos e métodos do web service) e clique em Add Reference. Utilize a figura abaixo como referência.



Após termos adicionado a web reference podemos utilizar o web service como qualquer outro objeto em nossa aplicação.

4 – Dando sequência ao nosso exercício, adicione uma pasta chamada ClassesApoio no web site, dentro desta pasta vamos inserir classes auxiliares para nossos exercícios.

5 – Insira uma classe chamada Endereco.cs dentro da pasta ClassesApoio, utilizaremos essa classe para armazenar os valores do web service e mandar as informações para a página de interface. Crie as propriedades dessa classe conforme imagem abaixo.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace CursoAspNetAvancado4._0.ClassesApoio
{
    public class Endereco
    {
        public string TipoLogradouro { get; set; }
        public string Logradouro { get; set; }
        public string Bairro { get; set; }
        public string UF { get; set; }
        public string Cidade { get; set; }
    }
}
```

6 – Insira uma classe chamada ConsultaCEP.cs. Essa classe irá possuir um método estático que irá retornar um objeto Endereco. O método irá consultar o web service enviando um numero de CEP, o web service irá retornar as informações relacionadas com o CEP e o método preencherá o objeto de retorno Endereco com as informações recebidas do web service. A imagem abaixo mostra a implementação desta classe.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace CursoAspNetAvancado4._0.ClassesApoyo
{
    public class ConsultaCEP
    {
        public static Endereco GetEndereco(string CEP)
        {

            wsCEP.wscep servicoCEP = new wsCEP.wscep();
            System.Data.DataSet ds = servicoCEP.cep(CEP);
            Endereco newEndereco = new Endereco();

            if (ds != null)
            {
                if (ds.Tables[0].Rows.Count > 0)
                {
                    newEndereco.Bairro = ds.Tables[0].Rows[0]["bairro"].ToString();
                    newEndereco.Cidade = ds.Tables[0].Rows[0]["cidade"].ToString();
                    newEndereco.Logradouro = ds.Tables[0].Rows[0]["nome"].ToString();
                    newEndereco.TipoLogradouro = ds.Tables[0].Rows[0]["logradouro"].ToString();
                    newEndereco.UF = ds.Tables[0].Rows[0]["UF"].ToString();
                }
            }

            return newEndereco;
        }
    }
}

```

Note que o serviço é chamado da mesma maneira que criar uma nova instância de um objeto qualquer. Pode ser visto também que o método “cep” do web service retorna um System.Data.DataSet com uma DataTable que possui somente um DataRow com as informações referente ao CEP.

7 – Agora vamos implementar os controles da página UtilizandoWSExterno.aspx. O objetivo é implementar uma textBox onde o usuário digite o CEP que deseja buscar informações e quando ele clicar em um botão de processamento, o sistema busque as informações do CEP digitado e escreva elas em outras textBox. Insira o código aspx abaixo na página UtilizandoWSExterno.aspx.

```
<h2>
    Utilizando Web Service Externo</h2>
<p>
    <asp:Label ID="Label1" runat="server" Text="CEP:></asp:Label>
    <asp:TextBox ID="txtCEP" runat="server" MaxLength="9"></asp:TextBox>
    <asp:Button ID="Button1" runat="server" Text="Consultar CEP"
        onclick="Button1_Click" />
    <br />
    <br />
    <br />
    <asp:Label ID="Label2" runat="server" Text="Tipo:></asp:Label>
    <asp:TextBox ID="txtTipo" runat="server"></asp:TextBox>
    <br />
    <asp:Label ID="Label3" runat="server" Text="Logradouro:></asp:Label>
    <asp:TextBox ID="txtLogradouro" runat="server"></asp:TextBox>
    <br />
    <asp:Label ID="Label4" runat="server" Text="Estado:></asp:Label>
    <asp:TextBox ID="txtEstado" runat="server"></asp:TextBox>
    <br />
    <asp:Label ID="Label5" runat="server" Text="Cidade:></asp:Label>
    <asp:TextBox ID="txtCidade" runat="server"></asp:TextBox>
    <br />
    <asp:Label ID="Label6" runat="server" Text="Bairro:></asp:Label>
    <asp:TextBox ID="txtBairro" runat="server"></asp:TextBox>
    <br />
</p>
```

8 – Dê dois cliques no botão Consultar CEP e insira o código abaixo.

```
protected void Button1_Click(object sender, EventArgs e)
{
    ClassesApoio.Endereco end = new ClassesApoio.Endereco();

    if (this.txtCEP.Text != string.Empty)
    {
        end = ClassesApoio.ConsultaCEP.GetEndereco(this.txtCEP.Text);

        txtCidade.Text = end.Cidade;
        txtEstado.Text = end.UF;
        txtLogradouro.Text = end.Logradouro;
        txtTipo.Text = end.TipoLogradouro;
        txtBairro.Text = end.Bairro;
    }
}
```

9 – Insira a chamada da página no menu principal do sistema e rode a aplicação, navegue até a página, insira algum CEP válido, clique em Consultar CEP e visualize as informações apresentadas.

Note como é simples e funcional utilizar web services de terceiros, após termos criado uma web reference, temos acesso ao web service como se ele fosse um objeto qualquer dentro de nossa aplicação.

Exercícios

Estes exercícios visão revisar o conteúdo visto ao longo deste capítulo sobre XML Web Services.

1. Onde deve-se procurar informações sobre Web Services disponíveis?
2. Como poderia se testar um Web Service rapidamente para ver os métodos e parâmetros que ele disponibiliza?
3. Desenvolva um web service que realize a soma entre 2 números e teste no browser.

Espaço para Anotações

8. Utilizando AJAX no ASP.NET

Objetivos

Ao final deste capítulo você estará apto à:

- Entender o que é o Ajax;
- Utilizar o controle ScriptManager;
- Utilizar o controle UpdatePanel;
- Registrar scripts a partir do ScriptManager;
- Utilizar AjaxControlToolkit;

O que é AJAX ?

Por AJAX entendemos Assynchronous JavaScript And XML, que significa execução assíncrona de javascript com uso de XML. A tecnologia AJAX do Microsoft ASP.NET permite a criação de páginas web pages que incluem uma experiência rica do usuário com elementos mais responsivos e interativos com a interface do usuário (UI). O AJAX do ASP.NET fornece as bibliotecas de script de cliente que incorporam ECMAScript (Javascript) cross-browser e tecnologias dinâmicas do HTML (DHTML), além de integrar-se com a plataforma de desenvolvimento voltada ao servidor do ASP.NET. Usando AJAX, você pode melhorar a experiência do usuário e a eficiência de suas aplicações web.

Por que usar AJAX no ASP.NET?

AJAX permite construir aplicações web ricas com muitas vantagens sobre as aplicações web baseadas totalmente no servidor. Alguns dos benefícios oferecidos pela tecnologia AJAX são:

- Eficiência melhorada executando partes significativas de página no browser, ao invés do servidor
- Elementos familiares de UI tais como indicadores do progresso, tooltips, e janelas pop-up.
- Updates parcial da página, onde somente as partes da web page que mudaram são atualizadas
- Compatibilidade com os browsers os mais populares e geralmente os mais usados, que inclui o Internet Explorer, Firefox, Chrome, etc.
- Adição de scripts via código .Net.
- Comportamento semelhante a aplicações windows forms.

Quanto a performance, a utilização do Ajax não tem um impacto significativo.

Utilizando Ajax em uma aplicação Web

Para entendermos a utilização do Ajax, vamos implementar seu uso em nossa aplicação e a cada tópico que iremos estudando vamos ir aumentando nossa implementação.

O primeiro passo é preparar o básico em nossa aplicação para que possamos realizar exercícios. Siga os passos.

1 – Adicione uma pasta chamada Ajax em nosso web site.

2 – Adicione um web form chamado pagAjaxGeral.aspx dentro da pasta criada!

3 – Adicione um item no menu da aplicação que faça a chamada de página.

4 – Insira os seguintes controles na pagAjaxGeral.aspx: Duas labels, uma textBox e um Button. O objetivo é digitar um texto, dar uma pausa no processamento e apresentar o texto digitado na label de resultado, dessa forma poderemos ver a página sendo carregada de forma normal e a página sendo carregada pelo Ajax. Utilize o código abaixo como base para montar o aspx.

```
<h2>
    Visão geral do Ajax
</h2>
<p>
    <asp:Label ID="Label1" runat="server" Text="Digite um texto:"></asp:Label>
    <br />
    <asp:TextBox ID="txtTexto" runat="server"></asp:TextBox>
    <br />
    <asp:Button ID="btnOk" runat="server" Text="Ok" />
    <br />
    <asp:Label ID="lblResultado" runat="server" Text=""></asp:Label>
</p>
```

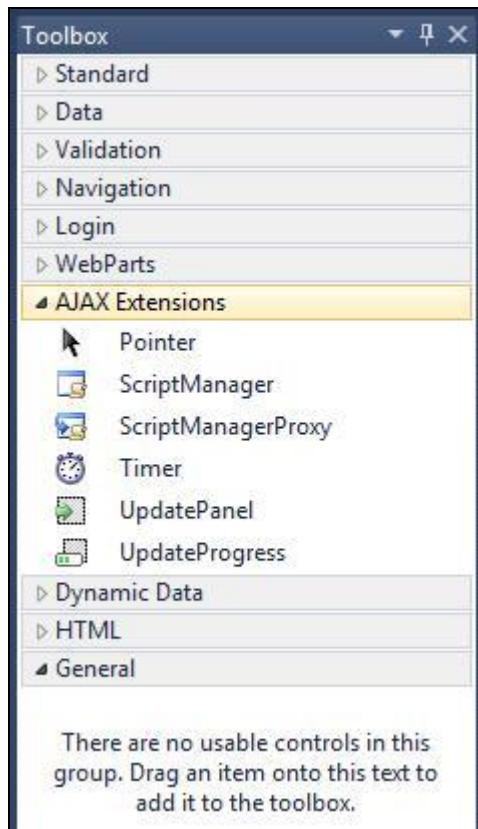
5 – Dê dois cliques no botão btnOk e insira o código abaixo. Leia os comentários do código para entender o objetivo de cada linha.

```
protected void btnOk_Click(object sender, EventArgs e)
{
    //add uma parada de 3 segundos no processamento
    //para exibir a barra de carregamento do navegador
    System.Threading.Thread.Sleep(3000);
    //escreve na label de resultado o texto digitado
    this.lblResultado.Text = "O texto digitado foi: " + this.txtTexto.Text;
}
```

6 – Rode o sistema, navegue até a página, digite algum texto e clique no botão. Repare que o sistema irá dar uma pausa de 3 segundos no processamento (em função da chamada do método Sleep), enquanto ocorre o processamento note que a barra de progresso do navegador é apresentada, isso mostra que uma requisição foi enviada para o servidor e a página está aguardando a resposta.

Após esse passo já temos o básico para iniciarmos o estudo do Ajax em nossa aplicação.

Quando nos referirmos aos controles Ajax, lembre-se que eles estão na caixa de ferramentas, na aba AjaxExtension.



O controle ScriptManager

Toda a página que utiliza os recursos do AJAX necessita de um controle em especial, o controle chamado Script Manager. O ScriptManager é responsável por gerenciar toda a comunicação da página com o servidor, atuando como um Proxy de comunicação. Todos os scripts da página são gerenciados pelo ScriptManager.

Se quisermos utilizar Ajax em uma página, é obrigatória a utilização de um ScriptManager ou de um controle que derive dele.

Se executarmos uma página que contém um controle scriptManager, podemos perceber que no código HTML será renderizado uma série de scripts que permitem o uso das funções do AJAX.

Toda página aspx pode ter somente um controle ScriptManager.

Insira um controle ScriptManager na página pagAjaxGeral.aspx, insira o controle acima da tag H2.

Rode o sistema e perceba que nada ocorre, porque o ScriptManager trabalha em conjunto com o controle UpdatePanel.

O controle UpdatePanel

Uma das grandes vantagens dos controles AJAX do ASP.NET é que eles não requerem tanta codificação como outras implementações de AJAX disponíveis. Um dos exemplos disso é o controle UpdatePanel. O controle updatepanel nos permite indicar na página quais os conteúdos que poderão ser alterados durante um postback.

Quando inserimos um UpdatePanel ele automaticamente é gerenciado pelo ScriptManager da página.

O controle UpdatePanel atua como um container, sendo que ele possui uma tag interna chamada ContentTemplate, onde todo o conteúdo inserido dentro do ContentTemplate irá ter seus métodos e comportamento gerenciados pelo Ajax.

O UpdatePanel possui também a tag Triggers, onde são associados controles e respectivos eventos que o UpdatePanel deve ficar associado. Veremos a utilização de Triggers mais adiante em nosso capítulo.

Para entendermos o funcionamento do UpdatePanel, insira esse controle, logo abaixo do ScriptManager, na página pagAjaxGeral, abra a tag ContentTemplate do UpdatePanel e arraste todo o conteúdo do aspx que tínhamos inserido antes, para dentro da tag ContentTemplate. Se baseie na figura abaixo para arrumar o código aspx.

```

<%@ Page Title="" Language="C#" MasterPageFile="~/Main.Master" AutoEventWireup="true"
CodeBehind="pagAjaxGeral.aspx.cs" Inherits="CursoAspNetAvancado4._0.Ajax.pagAjaxGeral" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">

    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>

    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>

            <h2>
                Visão geral do Ajax
            </h2>
            <p>
                <asp:Label ID="Label1" runat="server" Text="Digite um texto:"/></asp:Label>
                <br />
                <asp:TextBox ID="txtTexto" runat="server"></asp:TextBox>
                <br />
                <asp:Button ID="btnOk" runat="server" Text="Ok" OnClick="btnOk_Click" />
                <br />
                <asp:Label ID="lblResultado" runat="server" Text=""></asp:Label>
            </p>

        </ContentTemplate>
    </asp:UpdatePanel>

</asp:Content>

```

Após ter executado essa ação rode o sistema, digite algo e clique no botão btnOk. Conseguiu notar a diferença? Agora parece que a página não está fazendo nada e simplesmente a lblResultado apresenta o texto, sem que a página seja recarregada.

Esse é o comportamento de páginas que utilizam o Ajax. O processo de requisição e espera ao servidor é mascarado e a página não é toda renderizada, apenas o conteúdo de dentro do UpdatePanel é atualizado. E mesmo assim esse processo não é apresentado para o usuário.

O Controle UpdateProgress

Quando utilizamos Ajax o processamento passa despercebido para o usuário da aplicação, mas muitas vezes é interessante avisar o usuário que está sendo efetuado um processamento, para isso temos o controle UpdateProgress.

O controle UpdateProgress apresenta uma estrutura HTML, definida pelo programador, para o usuário quando um processamento está acontecendo.

O conteúdo que deve ser apresentado deve ser colocado dentro da tag interna do UpdateProgress chamada ProgressTemplate.

Vamos inserir um UpdateProgress em nossa aplicação para entendermos o comportamento do controle.

Primeiramente adicione na solution um gif que represente a ação de processamento (um bom site para baixar gif's é o www.ajaxload.info).

Adicione um controle UpdateProgress logo abaixo do fechamento do controle UpdatePanel e insira o código que pode ser visto na figura abaixo.

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server">
    <ProgressTemplate>
        <div id="Layer1" style="position:absolute; z-index:1; left: 45%; top: 45%; ">
            Processando . . .
            <br />
            <asp:Image ID="Image1" runat="server" ImageUrl="~/loading.gif" />
        </div>
    </ProgressTemplate>
</asp:UpdateProgress>
```

Agora rode o sistema e veja que quando você disparar um processamento o UpdateProgress será apresentado.

O Controle Timer

Outro controle disponível na plataforma AJAX do ASP.NET é o controle de Timer. O controle de timer permite que a página execute postbacks em um intervalo regular de tempo, conforme as configurações do controle.

O controle de Timer pode ser utilizado em aplicações de três cenários.

- Atualizar periodicamente o conteúdo de um UpdatePanel sem atualizar a página inteira
- Executar uma determinada função de postback no servidor a cada intervalo definido no Timer
- Postar a página para o servidor de maneira síncrona, fazendo com que toda a página seja atualizada

O controle de Timer possui uma propriedade Timer, que indica o número de milissegundos em que o evento da página deve acontecer.

Vamos inserir um controle Timer em nossa aplicação para que possamos entender bem o seu funcionamento.

1- Adicione o código abaixo, logo abaixo da abertura do parágrafo de dentro do ContentTemplate do UpdatePanel.

```

<h2>
    Visão geral do Ajax
</h2>
<p>

    <asp:Timer ID="Timer1" runat="server" Interval="1000" >
    </asp:Timer>
    <asp:Label ID="lblData" runat="server" />
    <br />
    <br />

```

2 – Dê dois cliques no controle Timer e adicione um código ao evento Tick que atualize a lblData com a data atual do sistema. O código abaixo apresenta como fazer isso.

```

protected void Timer1_Tick(object sender, EventArgs e)
{
    this.lblData.Text = System.DateTime.Now.ToString();
}

```

Rode o sistema e visualize que a cada segundo a lblData apresenta a data e a hora atual do sistema.

Triggers

O UpdatePanel possui a tag Triggers. Essa tag tem a funcionalidade de acoplar controles fora do ContentTemplate com o UpdateProgress, permitindo que o UpdatePanel gerencie qualquer controle da página.

Dentro da tag Triggers utilizamos a tag AsyncPostBackTrigger, dentro desta tag preencheremos as propriedades ControlID com o nome do controle que queremos associar ao Trigger e a propriedade EventName com o nome do evento que será acoplado com o UpdatePanel.

Vamos inserir a utilização de Triggers em nossa aplicação para que possamos ter um pleno entendimento. Siga os passos abaixo para implementar o exercício.

1 – Arraste o controle Timer1 para fora do UpdatePanel.

2 – adicione uma tag Triggers no UpdatePanel, acima da abertura da tag ContentTemplate, dentro da tag Triggers adicione um AsyncPostBackTrigger e preencha a propriedade ControlID com o nome do Timer e a propriedade EventName adicione o texto Tick que irá fazer referência ao evento Tick do timer. O resultado do aspx pode ser visto na figura abaixo.

```

<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
<asp:Timer ID="Timer1" runat="server" Interval="1000" ontick="Timer1_Tick" >
</asp:Timer>
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <Triggers>
        <asp:AsyncPostBackTrigger ControlID="Timer1" EventName="Tick" />
    </Triggers>
    <ContentTemplate>
        <h2>
            Visão geral do Ajax
        </h2>
        <p>
            <asp:Label ID="lblData" runat="server" />
            <br />
            <br />
            <asp:Label ID="Label1" runat="server" Text="Digite um texto:"></asp:Label>
            <br />
            <asp:TextBox ID="txtTexto" runat="server"></asp:TextBox>
            <br />
            <asp:Button ID="btnOk" runat="server" Text="Ok" OnClick="btnOk_Click" />
            <br />
            <asp:Label ID="lblResultado" runat="server" Text=""></asp:Label>
        </p>
    </ContentTemplate>
</asp:UpdatePanel>

```

Rode a aplicação e visualize que o updatePanel continua gerenciando o evento tick do timer, mesmo com o timer estando fora do ContentTemplate.

Inclusão de Scripts via ScriptManager

Uma funcionalidade muito importante que o ScriptManager possui é gerar, via código .net, código script na página.

A criação de scripts é realizada utilizando os métodos da classe responsável pelo ScriptManager e não pelo controle.

O método utilizado é o RegisterClientScriptBlock, que recebe como parâmetro o local onde ele deve inserir o script (no caso a própria página), o tipo do objeto, uma nomenclatura do script para efetuar o registro, o script e uma boolen que indica se deve registrar a tag do script.

Vamos implementar essa funcionalidade em nossa aplicação. Adicione um botão chamado btnScripts, na página pagAjaxGeral.aspx, e o texto do botão deve ser “Disparar alert javascript”.

Dê dois cliques no botão e insira o código abaixo. Depois rode a aplicação, clique no botão e visualize um alert javascript sendo criado a partir de código .net.

Utilizando AJAX no ASP.NET

```
protected void btnScripts_Click(object sender, EventArgs e)
{
    //vamos criar a função javascript
    System.Web.UI.ScriptManager.RegisterClientScriptBlock(this,
        this.GetType(),
        "Inclusão de alert",
        "function AlertHello() { alert('Mensagem javascript criada via código .net!'); }", true);

    //vamos chamar a função javascript
    System.Web.UI.ScriptManager.RegisterClientScriptBlock(this,
        this.GetType(),
        "chamada do alert",
        "javascript:AlertHello();", true);
}
```

AJAX Control Toolkit

O AJAX Control Toolkit é um pacote adicional que contém uma série de controles desenvolvidos para a plataforma AJAX ASP.NET. Este pacote não é instalado junto com as extensões de AJAX para o ASP.NET, deve ser instalado separadamente (o link para o pacote de instalação encontra-se no site do AJAX ASP.NET (<HTTP://ajax.asp.net>)).

Tenha muito cuidado na hora do download para baixar a versão do framework 4.0 do Ajax Control Toolkit. No próprio site do AjaxControlToolkit existe bastante material escrito e vídeos que explicam como utilizar os controles.

Após a instalação é aconselhável que você adicione os controles em sua toolbox do Visual Studio.NET. Para isto, basta seguir os seguintes passos.

1. Clique com o botão direito sobre a toolbox
2. Selecione a opção Add Tab
3. Dê a tab o nome de AJAX Control Toolkit
4. Clique com o botão direito sobre a área vazia na nova seção da toolbox e selecione “Choose Items...”
5. Clique no botão browse e selecione a DLL AjaxControlToolkit.DLL que se encontra na pasta de instalação do Toolkit.

Após seguir estes passos os controles do toolkit estarão disponíveis para uso na sua toolbox, conforme a figura a seguir.

Utilizando AJAX no ASP.NET

The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "Utilizando AJAX no ASP.NET". The menu bar includes File, Edit, View, Project, Debug, Format, Tools, Window, Help, and Full Screen. The toolbar has icons for Save, Undo, Redo, Cut, Copy, Paste, Find, Replace, and others.

The left side features a "Toolbox" pane containing categories like WebParts, AJAX Extensions, HTML, and AjaxControlToolkit. Under AjaxControlToolkit, numerous controls are listed: Pointer, Accordion, AccordionPane, AlwaysVisibleControlExtender, AnimationExtender, AsyncFileUpload, AutoCompleteExtender, CalendarExtender, CascadingDropDown, CollapsiblePanelExtender, ColorPickerExtender, ComboBox, ConfirmButtonExtender, DragPanelExtender, DropDownExtender, DropShadowExtender, DynamicPopulateExtender, FilteredTextBoxExtender, HoverMenuExtender, Editor, ListSearchExtender, MaskedEditExtender, MaskedEditValidator, ModalPopupExtender, MultiHandleSliderExtender, MutuallyExclusiveCheckBox..., NoBot, NumericUpDownExtender, PagingBulletedListExtender, and PasswordStrength.

The main workspace displays the "Main.Master" page. The code view shows the master page structure:

```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Main.master.cs" Inherits="CursoAspNetAvancado4._0.Main" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <link href="/Styles/Site.css" rel="stylesheet" type="text/css" />
    <asp:ContentPlaceHolder ID="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form runat="server">
        <div class="page">
            <div class="header">
                <div class="title">
                    <h1>Aplicação do Curso de Asp.Net Avançado 4.0</h1>
                    <h2></h2>
                </div>
            </div>
        </div>
    </form>
</body>

```

The design view shows a header section with the title "APLICAÇÃO DO CURSO DE ASP.NET AVANÇADO 4.0". Below the header is a navigation menu with items: Home, Grupos, Admin, Mapas, LINQ, Man. Estado, Web Service, and Ajax. To the right of the menu, there is a login status message: "Seja bem-vindo [UserName] - Login" and "lwwVisao". Below the status message, it says "Usuário não está logado".

The bottom status bar indicates the current state: Design, Split, Source, and the current position: Ln 15, Col 9, Ch 9, INS.

Utilizando controles do Ajax Control Toolkit

Agora vamos implementar em nossa aplicação a utilização de alguns controles pertencentes ao Ajax Control Toolkit.

Dentro do Toolkit temos diversos controles, mas para apresentarmos a facilidades de utilização desses controles, vamos aprender a utilizar 3 controles, o ValidatorCalloutExtender, o CollapsiblePanelExtender e o PopupControlExtender.

Para utilizar os outros controles pertencentes ao Ajax Control Toolkit, consulte a documentação que pode ser encontrada no site do toolkit.

ValidatorCalloutExtender

O controle ValidatorCalloutExtender tem como objetivo ser um extensor dos validadores do asp.net, adicionando a eles um comportamento mais amigável e a possibilidade de personalizar a aparência do validador.

Vamos implementar a utilização de um ValidatorCalloutExtender em nossa aplicação. Siga os passos abaixo para executar essa implementação.

1 – Insira um web form chamado pagValidatour na pasta Ajax de nossa aplicação.

2 – Insira o código abaixo no aspx de pagValidatour. No momento que for inserir o ValidatorCalloutExtender, arraste o controle da toolbox do AjaxControlToolKit.

```

<%@ Page Title="" Language="C#" MasterPageFile="~/Main.Master" AutoEventWireup="true"
CodeBehind="pagValidatour.aspx.cs" Inherits="CursoAspNetAvancado4._0.Ajax.pagValidatour" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
    <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
    </asp:ToolkitScriptManager>
    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
            <h2>
                Utilizando o ValidatorCalloutExtender</h2>
            <p>
                <asp:Label ID="Label1" runat="server" Text="Digite o texto aqui:"/><asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
                <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server" Display="None"
                    ErrorMessage="É obrigatório que o texto seja digitado!" ToolTip="É obrigatório que o texto seja digitado!" ControlToValidate="TextBox1"/><asp:RequiredFieldValidator>
                <asp:ValidatorCalloutExtender ID="ValidatorCalloutExtender1" TargetControlID="RequiredFieldValidator1" runat="server">
                </asp:ValidatorCalloutExtender>
                <br />
                <asp:Button ID="Button1" runat="server" Text="Processar" OnClick="Button1_Click" />
                <br />
                <br />
                <asp:Label ID="lblResultado" runat="server" Text=""></asp:Label>
            </p>
        </ContentTemplate>
    </asp:UpdatePanel>
</asp:Content>

```

Note que não estamos utilizando o ScriptManager e sim o ToolkitScriptManager. Para utilizarmos os controles do Toolkit devemos sempre utilizar o ToolkitScriptManager. Este controle herda as implementações do ScriptManager e possui algumas alterações de otimização com os controles do toolkit.

3 – Dê dois cliques no botão Processar e insira um código que escreva na lblResultado a data e a hora atual.

```

protected void Button1_Click(object sender, EventArgs e)
{
    this.lblResultado.Text = System.DateTime.Now.ToString();
}

```

4 – Crie uma opção no menu principal para chamar a página e rode o sistema. Não digite nenhum valor da textbox e clique no botão. Note que vai aparecer um “balão” informando ao usuário que ele necessita digitar algum texto. Esse “balão” é o controle ValidatorCalloutExtender. A aparência do controle pode ser customizada, a aparência de “balão” é a default. Visualize na figura abaixo como o controle é apresentado.

APLICAÇÃO DO CURSO DE ASP.NET AVANÇADO 4.0Seja bem-vindo marcelo - [Logout](#) - [Trocar Senha](#)

DATA: 23/01/2011 19:40:15

Home Grupos Admin Mapas LINQ Man. Estado Web Service Ajax

UTILIZANDO O VALIDATORCALLOUTEXTENDER

Digite o texto aqui:

Processar

É obrigatório que o texto seja digitado!

CollapsiblePanelExtender

O controle CollapsiblePanelExtender tem a funcionalidade de ficar vinculado com painéis (ou com qualquer controle do tipo container visual) e apresentar/esconder as informações do painel conforme receba cliques.

Para que ocorra o entendimento das funcionalidades deste controle vamos implementar sua utilização em nossa aplicação. Siga os passos abaixo.

- 1 - Insira um web form chamado pagCollapsible na pasta Ajax de nossa aplicação.

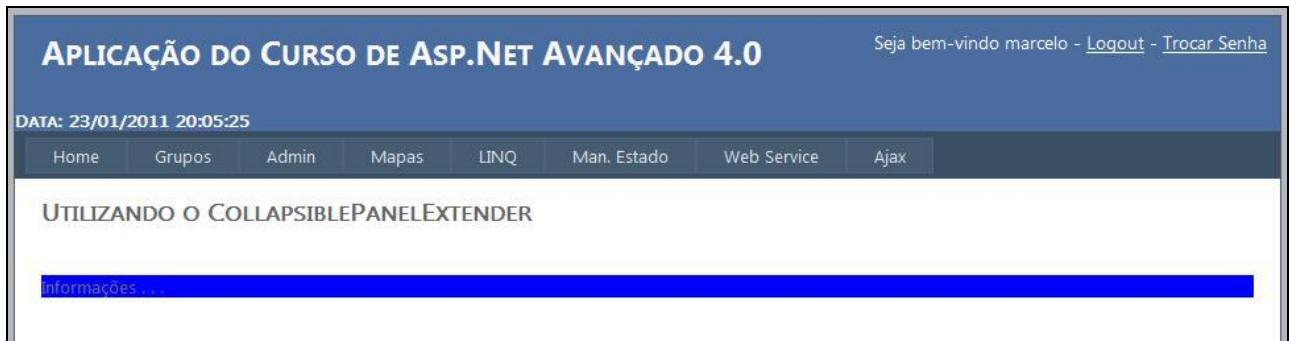
- 2 – Insira o código abaixo no aspx da pagCollapsible.

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Main.Master" AutoEventWireup="true" CodeBehind="pagCollapsible.aspx.cs" Inherits="CursoAspNetAvancado4._0.Ajax.pagCollapsible" %>
<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit" TagPrefix="asp" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server"></asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
    <h2>Utilizando o CollapsiblePanelExtender</h2>
    <p>
        <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server"></asp:ToolkitScriptManager>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
            <ContentTemplate>
                <asp:CollapsiblePanelExtender ID="cpeInformacoes" runat="server"
                    CollapsedText="Informações . . . "
                    ExpandedText="Esconder informações . . . "
                    Collapsed="true"
                    TextLabelID="lblTituloInf"
                    CollapseControlID="panelTituloInf"
                    ExpandControlID="panelTituloInf"
                    TargetControlID="panelCorpoInf" >
                </asp:CollapsiblePanelExtender>
                <asp:Panel ID="panelTituloInf" runat="server" BackColor="Blue" >
                    <asp:Label ID="lblTituloInf" runat="server" Text="Indisponivel" />
                </asp:Panel>
                <asp:Panel ID="panelCorpoInf" runat="server" BackColor="red">
                    <br />
                    Corpo da mensagem
                    <br />
                    Corpo da mensagem
                    <br />
                </asp:Panel>
            </ContentTemplate>
        </asp:UpdatePanel>
    </p>
</asp:Content>
```

Note que foi inserido dentro do panel panelCorpoInf algumas tags br e um texto qualquer. É importante salientar que a cor do panelCorpoInf é vermelha.

Estamos utilizando isso em nosso exercício para destacar as informações quando elas serão exibidas.

3 – Crie um item no menu para chamar a página e rode nossa aplicação. Clique no painel azul que é apresentado e note que o painel vermelho é apresentado, clique novamente no painel de cabeçalho e perceba que o painel vermelho é escondido.



PopupControlExtender

O controle PopupControlExtender tem a finalidade de adicionar um popup a algum controle. Isso é muito utilizado em Web Mails, como o gmail ou o hotmail, quando vamos escrever algum nome do destinatário do email e logo abaixo da caixa de digitação nos é apresentada um popup com os emails que temos cadastrado.

Vamos implementar sua utilização em nossa aplicação para que ocorra o entendimento de suas funcionalidades. Siga os passos abaixo.

1 - Insira um web form chamado pagPopup na pasta Ajax de nossa aplicação.

2 – Insira o código abaixo no aspx da página pagPopup.

Utilizando AJAX no ASP.NET

```
<%@ Page Title="" Language="C#" MasterPageFile("~/Main.Master" AutoEventWireup="true"
    CodeBehind="pagPopup.aspx.cs" Inherits="CursoAspNetAvancado4._0.Ajax.pagPopup" %>
<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit" TagPrefix="asp" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
    <style type="text/css">
        .PopupControl
        {
            background-color: #AAD4FF; position: absolute; visibility: hidden;
            border-style: solid; border-color: Black; border-width: 1px;
        }
    </style>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
    <asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
    </asp:ToolkitScriptManager>
    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
            <asp:Label ID="Label1" runat="server" Text="Enviar email para: "></asp:Label>
            <asp:TextBox ID="txtNome" runat="server" Width="500px"></asp:TextBox>
            <br />
            <asp:Panel ID="panel1" runat="server" CssClass="PopupControl">
                <asp:RadioButtonList ID="RadionButtonList1" runat="server" AutoPostBack="true" Width="146px"
                    OnSelectedIndexChanged="RadionButtonList1_SelectedIndexChanged">
                    <asp:ListItem Text="Marcelo Rosenthal" />
                    <asp:ListItem Text="Juliana Luzzi" />
                    <asp:ListItem Text="Pedro Tavarez" />
                    <asp:ListItem Text="Isabel Celina" />
                </asp:RadioButtonList>
            </asp:Panel>
            <br />
            <asp:PopupControlExtender ID="PopupControlExtender1" runat="server" TargetControlID="txtNome"
                PopupControlID="panel1" Position="Bottom" CommitProperty="value">
            </asp:PopupControlExtender>
        </ContentTemplate>
    </asp:UpdatePanel>
</asp:Content>
```

Note que foi inserido um código css dentro da página este código é utilizado para dar uma aparência ao panel que será utilizado como popup. No PopupControlExtender é configurado o controle que deve receber o popup (TargetControlID), o controle que deve ser apresentado com o popupControl (PopupControlID) e posição que deve ser apresentado em relação ao controle (Position).

3 – Selecione o controle RadionButtonList1, crie um evento SelectedIndexChanged e insira o código abaixo.

```
protected void RadionButtonList1_SelectedIndexChanged(object sender, EventArgs e)
{
    if (!string.IsNullOrEmpty(this.RadionButtonList1.SelectedValue))
    {
        PopupControlExtender.GetProxyForCurrentPopup(this.Page).Commit(
            this.txtNome.Text + " " + this.RadionButtonList1.SelectedValue);
    }
    this.RadionButtonList1.ClearSelection();
}
```

4 – Crie um item no menu para a página e rode a aplicação. Clique com o mouse sobre a textBox e veja que será apresentado um popup com o panel. Selecione um item do panel e veja o texto da listItem ser escrito na textBox.

APLICAÇÃO DO CURSO DE ASP.NET AVANÇADO 4.0

Seja bem-vindo marcelo - [Logout](#) - [Trocara Senha](#)

DATA: 23/01/2011 21:11:21

Home Grupos Admin Mapas LINQ Man. Estado Web Service Ajax

Enviar email para:

- Marcelo Rosenthal
- Juliana Luzzi
- Pedro Tavarez
- Isabel Celina

Espaço para anotações

9. Reflection

Objetivos

Ao final deste capítulo você estará apto à:

- Entender o que é o Reflection;
- Utilizar Assembly, AssemblyName e Módulos;
- Compreender o Type;
- Implementar Reflexão em sua aplicação;

O que é Reflection?

Reflection é um conjunto de classes que permitem acessar e manipular assemblies, módulos, tipos e os metadados que eles contêm. Por exemplo, você pode utilizar reflection para enumerar assemblies, módulos e classes carregadas e os métodos, propriedades, campos e eventos que cada tipo expõe. O reflection desempenha um papel fundamental no .net framework 4.0 e funciona como um bloco de construção para outras partes importantes do runtime. O runtime utiliza a reflexão em muitas circunstâncias, por exemplo, para enumerar os campos quando um tipo está sendo serializado ou quando sofre marshaling para um outro processo ou para uma máquina. O visual Studio utiliza a reflexão transparentemente sempre que você acessa um método de um objeto por vinculação tardia.

O código relacionado à reflexão em geral utiliza os tipos no namespace System.Reflection. A única classe utilizada pela reflexão fora desse namespace é System.Type, que representa um tipo em módulo gerenciado.

Funcionalidades e ganhos de utilizar reflexão em nossa aplicação

A utilização de reflexão em nossas aplicações tem como principais funcionalidades a automatização de métodos e a criação de arquiteturas genéricas.

Utilizando reflexão podemos criar comparadores universais, atribuidores de valores genéricos, personalização de métodos a partir de comparadores de tipos ou atributos, etc. Essa utilização proporciona ganhos tremendos em questões como segurança de código, otimização de processamento, reutilização de código e ganhos de produtividade.

Em função da grande gama de implementações que a reflexão disponibiliza ao desenvolvedor ela é amplamente utilizada em sistemas complexos.

Uma das desvantagens de utilizar reflection é a complexidade do código gerado, mas depois de programar dominar as técnicas de reflexão, essa desvantagem não é tão impactante.

O tipo Assembly

O tipo Assembly representa um assembly .Net. Esse tipo não oferece um método construtor porque você nunca realmente cria um assembly, mas simplesmente obtém uma referência a um assembly existente.

Após carregarmos o assembly o objeto irá fornecer uma forma de debugar e inspecionar as propriedades do assembly.

Verifique na imagem abaixo algumas formas de carregar o Assembly.

```
// pega uma referência ao assembly em que o código está sendo executado
System.Reflection.Assembly ass = System.Reflection.Assembly.GetExecutingAssembly();

//outra forma de alcançar o mesmo resultado que a linha acima
System.Reflection.Assembly assLoad = System.Reflection.Assembly.GetAssembly(typeof(System.Data.DataSet));

//Obtem uma referencia a um assembly de acordo com seu nome de exibição
System.Reflection.Assembly asm = System.Reflection.Assembly.Load("mscorlib");

//Obtém uma referência a um assembly de acordo com o nome do seu arquivo ou seu nome completo
System.Reflection.Assembly assLf = System.Reflection.Assembly .LoadFrom ("@c:\myapp\mylib.dll");
```

O tipo AssemblyName

A classe AssemblyName representa o objeto que o .net utiliza para armazenar a identidade e recuperar as informações sobre um assembly. Um objeto AssemblyName completamente especificado tem um nome, uma cultura e um número de versão, mas o runtime também pode utilizar objetos AssemblyName parcialmente preenchidos ao procurar um assembly a ser vinculado ao código chamador. Mais comumente, você obtém uma referência a um objeto AssemblyName existente utilizando a propriedade GetName do objeto Assembly.

```
//Obtem uma referencia a um assembly de acordo com seu nome de exibição
System.Reflection.Assembly asm = System.Reflection.Assembly.Load("mscorlib");
//recebe o assemblyName a partir do Assembly
System.Reflection.AssemblyName an = asm.GetName();

System.Text.StringBuilder sb = new System.Text.StringBuilder();

//insere na stringbuild o nome completo do assemblyname
sb.AppendLine(an.FullName.ToString());
//insere a arquitetura de processamento do assembly
sb.AppendLine(an.ProcessorArchitecture.ToString());
//pega a versao do assembly
sb.AppendLine(an.Version.Major.ToString());
sb.AppendLine(an.Version.Minor.ToString());
sb.AppendLine(an.Version.Build.ToString());
sb.AppendLine(an.Version.Revision.ToString());
```

O tipo Module

A classe Module representa um dos módulos em um assembly. Você pode enumerar todos os membros em um Assembly utilizando o método Assembly.GetModules.

Esta classe serve para desbrinchar as funcionalidades de um assembly.

```
//Obtem uma referencia a um assembly de acordo com seu nome de exibição
System.Reflection.Assembly asm = System.Reflection.Assembly.Load("mscorlib");
System.Text.StringBuilder sb = new System.Text.StringBuilder();

//percorre todo os modules do assembly
foreach (System.Reflection.Module item in asm.GetModules())
{
    //adiciona na stringbuilder o nome do module e o scopo de utilizacao
    sb.AppendLine(string.Format("{0} - {1}", item.Name, item.ScopeName));
}
```

Trabalhando com Tipos

A classe System.Type é central para todas as ações de reflexão. Ela representa um tipo gerenciado, um conceito que inclui classes, estruturas, módulos, interfaces e enums. A classe type fornece todos os meios de enumerar campos, propriedades, métodos e eventos de um tipo, bem como configurar propriedades e campos e invocar métodos dinamicamente.

Recuperando um objeto Type

A classe Type não possui nenhum construtor porque nunca é criado um objeto Type, ao invés disso, sempre é obtido uma referência a um tipo existente. Quando utilizamos um objeto Type, estamos pegando um tipo existente e utilizando esse tipo para sofrer o máximo de interações possíveis com nosso código, como verificar se o tipo é uma classe, um enum, uma estrutura ou uma interface.

Utilizamos a classe Type para destrinchar completamente um objeto ao ponto de descer, em seus maiores detalhes, para conseguirmos criar um código mais sofisticado, reutilizável e genérico possível.

Utilizando Reflection em nossa aplicação

Até agora vimos bastante teorias sobre a utilização de reflexão, seus objetos e seus ganhos, agora vamos implementar o uso de reflection em nossa aplicação.

A implementação será a utilização de reflection para preencher os controles da tela, genericamente, a partir de um array que irá servir como fonte de dados.

Siga os passos abaixo para efetuar essa implementação.

1 – Adicionar uma classe chamada Item na pasta ClassesApoio e crie as propriedades da classe conforme a figura abaixo. A classe item será a classe base para o array que será a fonte de dados.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace CursoAspNetAvancado4._0.ClassesApoio
{
    public class Item
    {
        public int ID { get; set; }

        public string Descricao { get; set; }

        public string codigoEAN { get; set; }

        public string Tipo { get; set; }

        public double Peso { get; set; }

        public double Preco { get; set; }
    }
}
```

2 – Adicione uma classe BasePaginas na Pasta ClassesApoio, essa classe servirá como base para todas as páginas que queiram utilizar reflection. Como é uma classe base ela deve herdar a System.Web.UI.Page, para que as páginas que herdem dela recebam as funcionalidades de uma página. Utilize a figura abaixo como base para montar sua classe

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
namespace CursoAspNetAvancado4._0.ClassesApoyo
{
    public class BasePaginas: System.Web.UI.Page
    {
    }
}
```

3 – Insira os métodos de reflexão na classe BasePaginas, o objetivo desses métodos é receber um objeto e preencher genéricamente, utilizando reflexão, os controles textBox, da tela, que possuam o mesmo nome que as propriedades do objeto. Visualiza as duas imagens abaixo para criar os métodos.

```
public void PreencheControlesGenericos(Item origem)
{
    //percorre todos os controles bases
    foreach (System.Web.UI.Control c in this.Controls)
    {
        //se tem filhos
        if (c.Controls.Count > 0)
        {
            //chama a sobrecarga passando os filhos
            this.PreencheControlesGenericos(origem, c.Controls);
        }
    }
}
```

Utilizando AJAX no ASP.NET

```
public void PreencheControlesGenericos(Item origem, System.Web.UI.ControlCollection controles)
{
    //executa um laço para percorrer todos os controles
    foreach (System.Web.UI.Control c in controles)
    {
        //se tem controles internos entao chama a função recursivamente
        if (c.Controls.Count > 0)
        {
            //chama a si mesmo passando os controles genericos
            PreencheControlesGenericos(origem, c.Controls);
        }
        else
        {
            //declarar uma variavel que seja do tipo de informacao de propriedades
            System.Reflection.PropertyInfo aProp;
            //verifica se o controle eh do tipo textbox
            if (typeof(System.Web.UI.WebControls.TextBox).IsInstanceOfType(c))
            {
                //busca dentro do objeto de origem uma propriedade que seja igual ao nome do controle
                aProp = origem.GetType().GetProperty(c.ID.ToString());

                //se tem alguma propriedade com o mesmo nome da textbox
                if (aProp != null)
                {
                    //verifica se tem algum valor dentro do objeto de origem
                    if (aProp.GetValue(origem, null) != null)
                    {
                        //preenche o textbox com o valor de dentro do objeto de negocio
                        ((System.Web.UI.WebControls.TextBox)c).Text = aProp.GetValue(origem, null).ToString();
                    }
                }
            }
        }
    }
}
```

4 – Crie uma pasta na aplicação chamada Reflection e dentro desta pasta insira um web form chamado pagReflection.aspx.

5 – Adicione o segundo código no evento Page_Load da página. Este código irá criar uma fonte de dados fake para o nosso exercício.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {

        List<ClassesApoio.Item> lista = new List<ClassesApoio.Item>();

        for (int i = 0; i < 10; i++)
        {
            ClassesApoio.Item newItem = new ClassesApoio.Item();

            newItem.ID = i;
            newItem.Descricao = "Item " + i.ToString();
            newItem.codigoEAN = "EAN " + i.ToString();
            newItem.Peso = 33.2 * i;
            newItem.Preco = 13.56 * i;
            newItem.Tipo = "Tipo " + i.ToString();

            lista.Add(newItem);
        }

        Session["lista"] = lista;
    }
}
```

6 – Arrume a herança do web form, agora ao invés dele herdar de System.Web.UI.Page ele irá herdar de BasePaginas.

```
public partial class pagReflection : ClassesApoio.BasePaginas
{
```

7 – Insira os controles abaixo no aspx da página.

```

<h2>
    Utilizando Reflection</h2>
<p>
    <asp:Label ID="Label1" runat="server" Text="Digite uma posicao entre 0 e 9:"></asp:Label>
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    <asp:Button ID="Button1" runat="server" OnClick="Button1_Click" Text="Buscar" />
    <br />
    <br />
    <br />
    <asp:Label ID="Label2" runat="server" Text="ID:"></asp:Label>
    <asp:TextBox ID="ID" runat="server"></asp:TextBox>
    <br />
    <asp:Label ID="Label3" runat="server" Text="Descricao:"></asp:Label>
    <asp:TextBox ID="Descricao" runat="server"></asp:TextBox>
    <br />
    <asp:Label ID="Label4" runat="server" Text="EAN:"></asp:Label>
    <asp:TextBox ID="codigoEAN" runat="server"></asp:TextBox>
    <br />
    <asp:Label ID="Label5" runat="server" Text="Peso:"></asp:Label>
    <asp:TextBox ID="Peso" runat="server"></asp:TextBox>
    <br />
    <asp:Label ID="Label6" runat="server" Text="Preco:"></asp:Label>
    <asp:TextBox ID="Preco" runat="server"></asp:TextBox>
    <br />
    <asp:Label ID="Label7" runat="server" Text="Tipo:"></asp:Label>
    <asp:TextBox ID="Tipo" runat="server"></asp:TextBox>
</p>

```

8 – Dê dois cliques no Button1 e insira o código abaixo.

```

protected void Button1_Click(object sender, EventArgs e)
{
    List<ClassesApoio.Item> lista = (List<ClassesApoio.Item>)Session["lista"];
    int Index = Convert.ToInt32(this.TextBox1.Text);

    if (Index >= 0 && Index < 10)
    {
        this.PreencheControlesGenericos(lista[Index]);
    }
}

```

9 – Crie uma opção no menu que chame a página, rode o sistema, acesse a página, escreva um numero de 0 à 9 e veja como os controles serão preenchidos via reflection, sem que seja especificado diretamente o preenchimento de cada controle e sim utilizando a estrutura de preenchimento genérica criada a partir do Reflection.

Espaço para anotações

10. Distribuindo sua Aplicação

Objetivos

Ao final deste capítulo você estará apto a:

- Entender as diretivas de compilação;
- Publicar sua aplicação asp.net;

Distribuindo sua Aplicação ASP.NET

Normalmente todo o software, depois de finalizado, deve ser distribuído. Aplicações convencionais na sua maioria são distribuídas através de instaladores específicos, o que permite que qualquer usuário possa instalar a aplicação em seu computador de maneira fácil, simplesmente seguindo alguns passos básicos.

Aplicações Web na sua maioria são instaladas em um único ou em alguns poucos servidores. Você pode simplesmente copia-la para o destino ou até mesmo gerar um instalador, como numa aplicação convencional.

Mas além de distribuir sua aplicação existem outros cuidados que você deve ter antes de colocar sua aplicação em produção.

Primeiramente quando você cria sua aplicação, por padrão ele vai estar com a depuração habilitada. A depuração é habilitada no arquivo web.config.

A depuração, como você deve imaginar, permite que você depure seu aplicativo. Isto tem um custo: Alguns arquivos temporários têm que ser mantidos e o desempenho da aplicação é afetado. Quando você roda seu aplicativo pela primeira vez, dentro da IDE do VS, uma caixa de diálogo informa que a depuração está desabilitada. Você pode habilitá-la ou rodar a aplicação sem depuração, que seria o equivalente a executá-la fora da ambiente do VS:

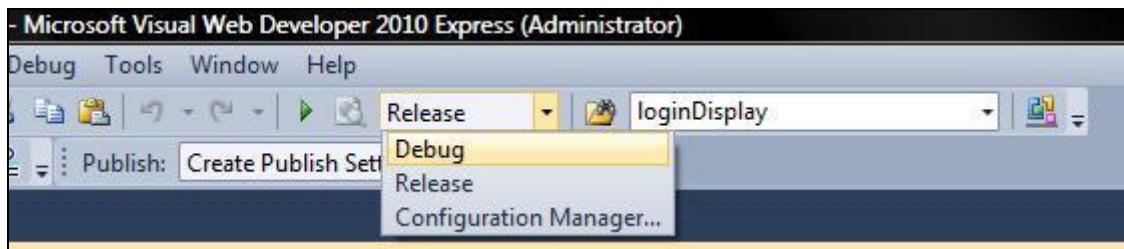


Ao autorizar a habilitação de depuração,

```
<compilation debug="true" targetFramework="4.0">
```

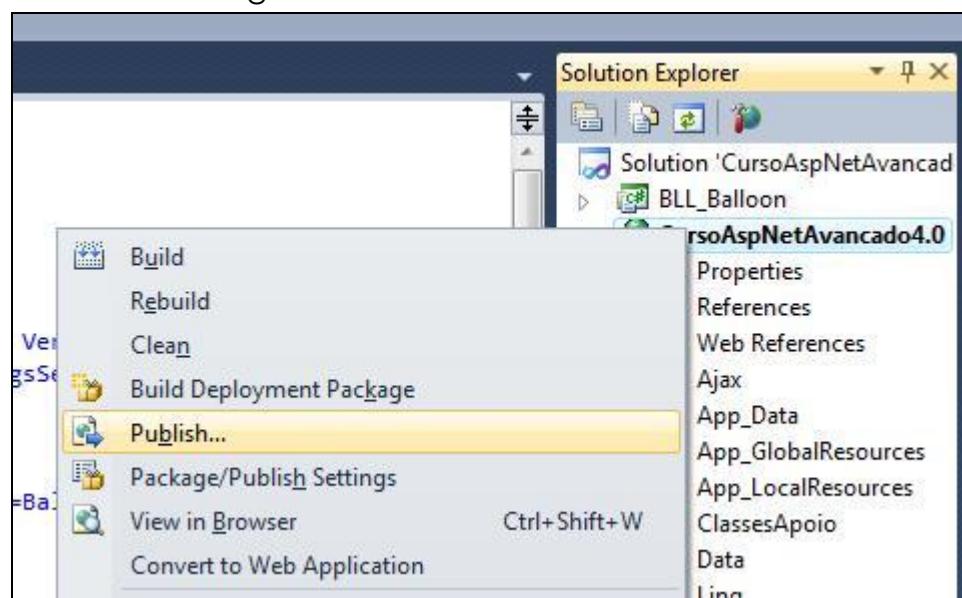
O que isto tem a ver com distribuição de aplicação? É importante lembrar de desabilitar a depuração quando compilar sua aplicação para distribuição, pois será gerado um código otimizado e mais compacto.

Outra dica importante é colocar a aplicação em modo de Realise, no próprio VS, como na imagem abaixo:



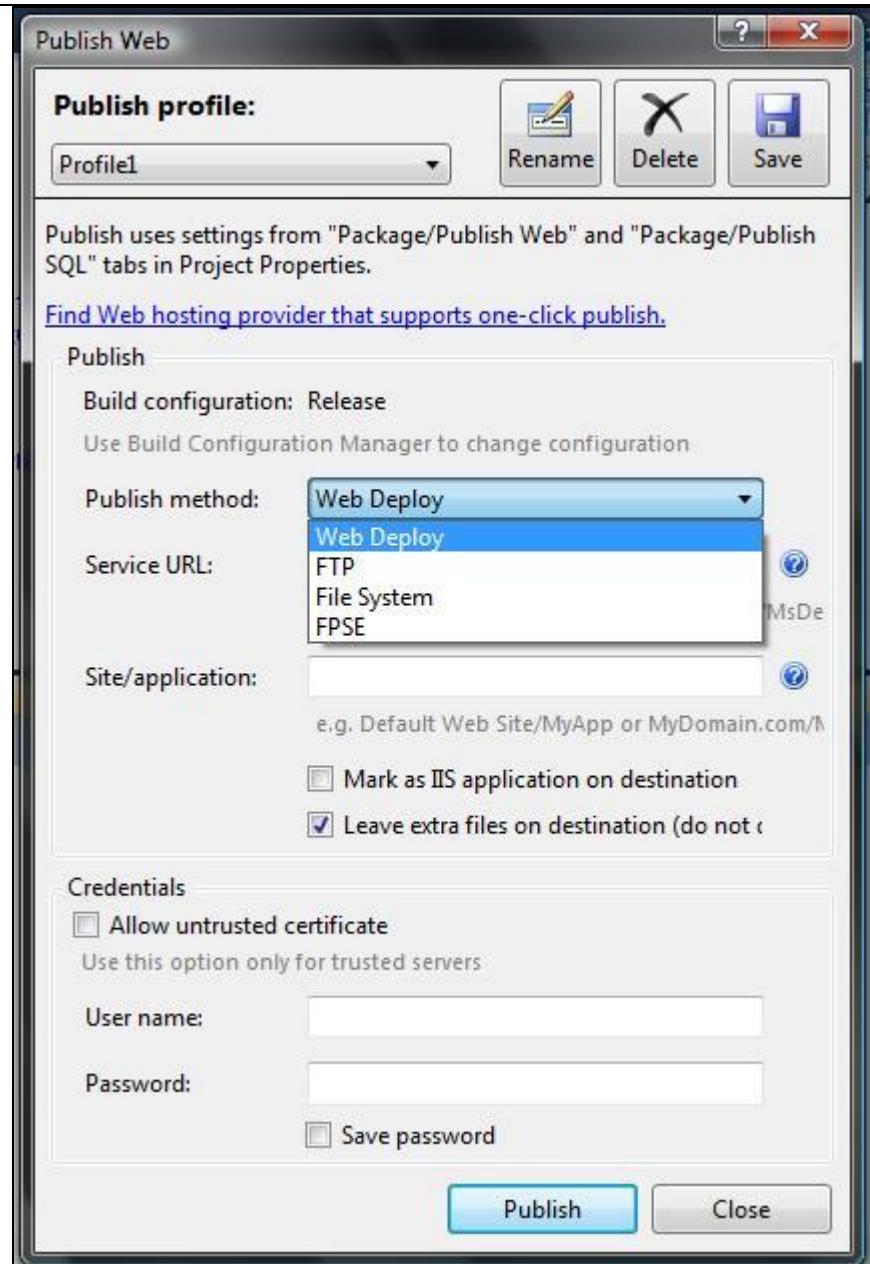
Publicando sua aplicação

A ferramenta de publicação cria uma cópia pré-compilada de seu web site. Para acioná-lo basta clicar com o botão direito no web site e selecionar a opção “Publish...” conforme a imagem abaixo.



Será apresentada uma janela onde iremos selecionar a opção de publicação que queremos executar. Podemos publicar direto para um servidor web, um servidor FTP, um diretório de sistema ou servidor na rede interna e via FPSE.

A figura abaixo apresenta as opções.



Cada opção terá suas respectivas configurações e após devemos clicar em Publish e o Visual Studio ficará encarregado de realizar o resto do processo.

Espaço para Anotações