



Log In / Cadastre-se

Pesquisar

HOME DESENVOLVIMENTO FRONT-END BANCO DE DADOS EM DESTAQUE TODOS

PUBLIQUE

Desenvolvimento - ASP. NET

Explorando Segurança do ASP.NET - Membership e MembershipUser

A classe estática *Membership*, encontrada dentro do Namespace *System.Web.Security*, é usada pelo ASP.NET para validar as credenciais do usuário e gerenciar as configurações do mesmo.

por Israel Aécio

1 Like 0

A classe estática *Membership*, encontrada dentro do Namespace *System.Web.Security*, é usada pelo ASP.NET para validar as credenciais do usuário e gerenciar as configurações do mesmo. Esta classe faz uso da classe *FormsAuthentication* (já existente nas versões anteriores do ASP.NET) para criar e gerenciar a autenticação do usuário dentro da aplicação Web.

Falando um pouco sobre o seu funcionamento: existe um membro interno chamado *s_Provider* do tipo *MembershipProvider* (a classe base para as classes concretas de *Membership*, como por exemplo o *SqlMembershipProvider*), o qual receberá a instância da classe concreta. Essa inicialização acontece quando um método interno chamado *Initialize* é executado. Ele se encarrega de extrair os *providers* do arquivo *Web.Config* e instanciá-los para que, quando chamarmos os métodos e propriedades, já sejam efetivamente os métodos e propriedades da classe concreta que queremos utilizar.

Como esta classe encapsula o acesso ao *provider* específico, ela fornece facilidades como: criação de novos usuários, validação de usuários, gerenciamento de usuários em seu repositório (SQL Server, ActiveDirectory, etc). Como já vimos anteriormente o funcionamento interno desta classe, podemos perceber que o ASP.NET confia no *provider* especificado no *Web.Config* para se comunicar com a fonte de dados. Como já sabemos, o .NET Framework inclui a classe *SqlMembershipProvider* para acesso e persistência dos dados no SQL Server e é este que vamos utilizar no decorrer dos exemplos deste artigo.

Mas esta arquitetura é extensível, ou seja, a qualquer momento eu posso criar o meu próprio *provider* de *Membership*, onde eu queira customizá-lo para um outro repositório qualquer. E, para que isso seja possível, é necessário herdarmos da classe abstrata chamada *MembershipProvider* e customizá-la de acordo com o nosso repositório. Para ilustrar o cenário, imaginemos que desejamos ter um *provider* de *Membership* sendo persistido em um arquivo XML:

```
public class XmlMembershipProvider : MembershipProvider
{
    // customizo aqui todos os métodos e
    // propriedades necessárias
}
```

Depois do *provider* criado, basta definí-lo nas configurações do *Web.Config* e começar a utilizá-lo.

A configuração no arquivo Web.Config

Por padrão, todas as aplicações ASP.NET 2.0 já utilizam o *Membership*. Isso porque dentro do arquivo de configuração *machine.config* já temos um *provider* chamado *AspNetSqlProvider* que utiliza o *SqlMembershipProvider*. Este *provider* aponta para um banco de dados chamado *ASPNETDB.MDF* que encontra-se dentro da pasta *App_Data* da aplicação e é habilitado quando marcamos a opção *AspNetSqlProvider* no Web Site Administration Tool. Como o intuito do artigo é mostrar como configurar isso manualmente, vamos especificar o *provider* em nossa aplicação e configurá-lo para atender as nossas necessidades e, para isso, vamos até o arquivo *Web.Config* da aplicação para efetuar tal customização.

Já que não queremos utilizar o *provider* e o banco de dados *ASPNETDB.MDF* fornecido por padrão



Publicidade

VAMOS CRIAR APPS

QUALCOMM®

Developer Network

► CONHEÇA AS FERRAMENTAS

REVISTAS DEVMEDIA



Java mag 120



easy Java Mag 34

VER TODAS

ASSINE

TOP 10 - ARTIGOS

TOP 10 - AUTORES

- 1 Comandos básicos em SQL - insert, update, delete e select
- 2 HTML Básico
- 3 Menu em CSS - Menu dropdown horizontal com HTML5 e CSS3
- 4 Dia do Profissional de T.I - Regulamentação da profissão
- 5 Quanto vale o seu serviço? Aprenda a cobrar pelo seu trabalho.
- 6 Criando um sistema de cadastro e login com PHP e MySQL
- 7 Crop jQuery - Recortando imagens com jCrop
- 8 Manipulando dados de formulários com PHP - Parte 1
- 9 HTML Avançado
- 10 Aprenda PHP e torne-se um bom programador sem gastar quase nada

VER TODOS

pela plataforma, devemos recorrer ao arquivo Web.Config para configurar o *provider* da forma que precisarmos. Para esta configuração temos os seguintes elementos: *membership* e *providers*, que deverão estar declarados dentro do elemento *system.web*. Estes elementos, por sua vez, possuem uma série de atributos que definem as configurações e possíveis comportamentos e validações que o nosso *provider* irá desempenhar.



Para cada *provider* que é adicionado dentro do elemento *providers* especificamos os atributos de configuração deste *provider* e, através de atributos definidos no elemento *membership*, definimos configurações a nível "global de *providers*". Veremos cada um dos possíveis atributos (voltado para o uso do *SqlMembershipProvider*) e seus respectivos valores nas tabelas abaixo:

Elemento membership

Atributo	Tipo	Valor Padrão	Opcional	Descrição
defaultProvider	String	AspNetSqlProvider	Sim	O nome do <i>provider</i> que você vai utilizar. Lembrando que no elemento <i>providers</i> você pode especificar uma lista deles e é através do <i>defaultProvider</i> que você especifica qual deles utilizar.
userIsOnlineTimeWindow	Inteiro	15	Sim	Define o tempo em que usuário é considerado como online, que é utilizado para calcular o número aproximado de usuários online.
hashAlgorithmType	String	SHA1	Sim	Utilizado para especificar o algoritmo de <i>hashing</i> que será usado pelo <i>provider</i> , sempre quando o mesmo precisar efetuar <i>hashing</i> em dados sigilosos.

Elemento add - providers ["filho" do membership]

Atributo	Tipo	Valor Padrão	Opcional	Descrição
applicationName	String	<u>ApplicationVirtualPath</u>	Sim	Especifica o nome da aplicação em que o <i>provider</i> está sendo utilizado. Quando usado desta forma, várias aplicações podem usar a mesma base de dados e o mesmo nome de <i>provider</i> em outras aplicações. O <i>provider</i> que utiliza a propriedade <i>ActiveDirectory</i> não requer este atributo. Obs.: Atente-se a não definir como nome o caractere <i>/</i> . É o nome da aplicação e não o nome da aplicação correto, pois não reflete o ambiente de desenvolvimento e consequentemente pode causar problemas em registros vinculados.
commandTimeout	Inteiro	30 (ADO.NET)	Sim	Especifica o tempo em segundos para o comando executar.

			do SqlCommand que definimos entre a execução e o erro.
connectionStringName	String	Não	Especifica a <i>ConnectionString</i> utilizada pelo Connection String no elemento <code>connectionStrings</code> .
description	String	Sim	Uma breve descrição do <i>provider</i> .
enablePasswordRetrieval	Boolean False	Sim	Especifica se é permitido a recuperação de password do usuário pelo <i>provider</i> permitido.
enablePasswordReset	Boolean True	Sim	Especifica se é permitido redefinir a senha do usuário pelo <i>provider</i> de reinicialização.
maxInvalidPasswordAttempts	Inteiro 5	Sim	Define o número máximo de tentativas de acesso até o usuário seja bloqueado. O resultado da recuperação de senha ou senha.
minRequiredNonalphanumericCharacters	Inteiro 1	Sim	Define o número mínimo de caracteres especiais que devem conter, não pode ser menor que 1 e com um valor maior que 128. O valor padrão é 1 e o atributo minRequiredNonalphanumericCharacters .
minRequiredPasswordLength	Inteiro 1	Sim	Define o número mínimo de caracteres para a senha. Também não pode ser menor que 1 e com um valor maior que 128. O valor padrão é 1 e o atributo minRequiredPasswordLength .
name	String	Não	Define o nome do <i>provider</i> que deverá ser informado no elemento defaultProvider do membership para que ele seja utilizado.
passwordAttemptWindow	Inteiro 10	Sim	Define o intervalo de tempo que é efetuada a contagem de tentativas de acesso. Quando chega ao fim do intervalo especificado, o atributo maxInvalidPasswordAttempts é resetado e o número de falhas não é mais estipulado. O valor padrão é 10 e o atributo maxInvalidPasswordAttempts .

				reinicializados.
passwordFormat	String	Hashed	Sim	Define o armazenamento de dados para e definidos na <i>MembershipProvider</i> são exibidos at
				<ul style="list-style-type: none"> ◦ Clear: As são criptogr ◦ Encrypte criptografad configuração especificada de configura <i>machineKey</i> ◦ Hashed: . criptografad algoritmo de
				Obs.: Como alq irreversíveis, conseguiremos senha se e diferente de <i>Has</i>
passwordStrengthRegularExpression	String		Sim	Especifica a regular para va esta expressãc com a classe R
requiresQuestionAndAnswer	Boolean	True	Sim	Indica se é o informar uma respectiva pa que isso fun espécie de "I usuário.
requiresUniqueEmail	Boolean	True	Sim	Especifica se o mail do usuário
type	String		Não	Indica o tipo implementação ou seja, a cla implementa a <i>MembershipPro</i>

Agora que já conhecemos todos os parâmetros possíveis dos elementos acima, veremos abaixo um exemplo de como estar configurando isso dentro do arquivo Web.Config da aplicação ASP.NET:

```
<?xml version="1.0"?>
<configuration>
  <connectionStrings>
    <clear/>
    <add
      name="SqlConnectionString"
      connectionString="Data Source=local;Initial Catalog=DBTest;Integrated Security=True;"/>
    </connectionStrings>
  <system.web>
    <membership defaultProvider="SqlMembershipProvider">
      <providers>
        <clear/>
        <add
```

```

    name="SqlMembershipProvider"
    type="System.Web.Security.SqlMembershipProvider"
    connectionStringName="SqlConnectionString"
    applicationName="NomeAplicacao"
    enablePasswordRetrieval="false"
    enablePasswordReset="true"
    requiresQuestionAndAnswer="false"
    requiresUniqueEmail="false"
    passwordFormat="Hashed"
    minRequiredNonalphanumericCharacters="0"
    minRequiredPasswordLength="8" />
</providers>
</membership>
</system.web>
</configuration>

```

Algumas considerações do código acima: temos uma coleção de *connectionStrings*, onde através do elemento *connectionStrings* as informamos e no atributo *connectionStringName* especificamos o nome da conexão que iremos utilizar; outro ponto importante é o elemento *add* que é "filho" do elemento *providers*: como podemos ter uma coleção de *providers*, podemos adicionar quantos desejarmos e sempre fazemos isso adicionando um novo elemento *add* e as suas respectivas configurações.

Agora que já temos conhecimento suficiente para criar a infraestrutura e configurar um determinado *provider*, vamos analisar a classe *Membership* e seus respectivos membros. Veremos agora como manipular e gerir usuários dentro de uma base de dados SQL Server e, mais tarde, quando estivermos falando sobre os controles que o ASP.NET fornece, veremos como implementar a segurança em uma aplicação utilizando *FormsAuthentication*.

Não vou me preocupar aqui em mostrar e explicar as propriedades da classe *Membership* justamente porque são propriedades de somente leitura e apenas devolvem os valores que configuramos no Web.Config para o *provider*. Nos concentraremos apenas nos métodos, pois são o mais importante e nos dão todas as funcionalidades de manutenção e criação de usuários na base de dados.

Método	Descrição
CreateUser	<p>Adiciona um novo usuário na base de dados. Este método é sobrecarregado.</p> <p>Como o próprio nome diz, ele adiciona um novo usuário na base de dados e retorna um objeto do tipo <i>MembershipUser</i> (veremos sobre ele mais abaixo) se o usuário for criado com sucesso. Se, por algum motivo, a criação do usuário falhar, uma <i>Exception</i> do tipo MembershipCreateUserException é atirada. Além disso, você também pode, através de um parâmetro de saída do tipo MembershipCreateStatus, recuperar o motivo pelo qual a criação falhou.</p>
DeleteUser	<p>Exclui um usuário na base de dados dado um nome de usuário. Este método é sobrecarregado.</p> <p>Este método retorna um parâmetro booleano, indicando se o usuário foi ou não excluído com sucesso. Também é possível passar como parâmetro para este método um valor booleano indicando se os dados relacionados com este usuário serão também excluídos. O padrão é True.</p>
FindUsersByEmail	<p>Dado um e-mail para ser procurado, o método retornará (paginando) uma coleção do tipo MembershipUserCollection, onde cada elemento desta coleção é do tipo MembershipUser.</p> <p>O SQL Server executa a busca utilizando o operador <i>LIKE</i> e você pode passar para o mesmo os caracteres de busca para que possa ser analisado de acordo com a sua necessidade. Lembrando também que há um <i>overload</i> para este método onde você pode especificar os parâmetros <i>pageIndex</i>, <i>pageSize</i> e <i>totalRecords</i> para retornar os dados paginados para a sua aplicação e ter uma</p>

melhor performance se houver uma quantidade considerável de registros na base de dados.

FindUsersByName

Dado um nome de usuário para ser procurado, o método retornará (paginando) uma coleção do tipo **MembershipUserCollection**, onde cada elemento desta coleção é do tipo **MembershipUser**.

O SQL Server executa a busca utilizando o operador *LIKE* e você pode passar para o mesmo os caracteres de busca para que possa ser analisado de acordo com a sua necessidade. Lembrando também que há um *overload* para este método que você pode especificar os parâmetros *pageIndex*, *pageSize* e *totalRecords* para retornar os dados paginados para a sua aplicação e, ter uma melhor performance se houver uma quantidade considerável de registros na base de dados.

GeneratePassword

Gera uma senha randômica dado um tamanho máximo de caracteres e o número de caracteres não alfanuméricos.

Este método é comumente utilizado para gerar senhas randômicas e também é chamado pelo método **ResetPassword** para reinicializar a senha do usuário, ou seja, criar uma espécie de "senha temporária". Se existir no mínimo um caractere alfanumérico, eles não poderão ser caracteres não "imprimíveis" ou caracteres ocultos. Eis aqui as possibilidades: !@#\$%^&*()_-=+[]{};<>|./?

GetAllUsers

O método retornará (paginando) uma coleção do tipo **MembershipUserCollection** contendo todos os usuários da base de dados, onde cada elemento desta coleção é do tipo **MembershipUser**.

Lembrando também que há um *overload* para este método que você pode especificar os parâmetros *pageIndex*, *pageSize* e *totalRecords* para retornar os dados paginados para a sua aplicação e, ter uma melhor performance se houver uma quantidade considerável de registros na base de dados.

GetNumberOfUsersOnline

Retorna o número de usuários online para a aplicação corrente, ou seja, a qual está definida na configuração do *provider*.

O critério para o usuário ser considerado como online é a data da última atividade ser maior que a hora corrente menos o valor especificado no atributo **UserIsOnlineTimeWindow**. A data da última atividade é atualizada para a data corrente quando as credenciais do usuário são passadas para o método **ValidateUser** ou **UpdateUser** ou quando você chama o método **GetUser** ou o mesmo método sobrecarregado, passando o parâmetro *userIsOnline* como *True*.

GetUser

Retorna um objeto do tipo **MembershipUser** que representa um usuário.

Se chamar o método *GetUser* sem nenhum parâmetro, o método retornará o usuário logado. Se optar por passar o nome do usuário para este mesmo método, mas utilizando uma sobrecarga dele, ele retornará os dados deste usuário especificado.

GetUserNameByEmail

Retorna um objeto do tipo **MembershipUser** baseando-se em um e-mail informado como parâmetro.

UpdateUser

Atualiza na base de dados as informações de um determinado usuário.

Esse método recebe um objeto do tipo **MembershipUser**. Este

objeto pode ser um que recuperamos anteriormente da base de dados ou até mesmo um novo que acabamos de instanciar e definir seus valores.

ValidateUser

Dado um nome de usuário e senha, este método retorna um valor booleano indicando se é ou não um usuário válido.

Em alguns dos métodos acima comentamos sobre o objeto chamado *MembershipUser*. Veremos abaixo a lista de métodos e propriedades do mesmo com uma breve descrição.

Propriedade	Descrição
Comment	Comentários adicionais sobre o usuário.
CreationDate	Data/hora em que o usuário foi criado dentro da base de dados.
Email	E-mail do usuário.
IsApproved	Indica se o usuário pode ou não se autenticar. Mesmo informando o nome de usuário e senha válidos e esta propriedade estiver definida como False , o método ValidateUser retornará False .
IsLockedOut	Indica se o usuário está barrado devido a tentativas incorretas de acessar a aplicação.
IsOnline	Indica se o usuário está ou não online.
LastActivityDate	Recupera a data/hora do último acesso/atividade dentro da aplicação.
LastLockoutDate	Recupera a data/hora do último vez que esse usuário foi barrado.
LastLoginDate	Recupera a data/hora da última vez em que o usuário fez o login com sucesso na aplicação.
LastPasswordChangedDate	Recupera a data/hora da última vez em que o usuário alterou a sua senha.
PasswordQuestion	Recupera uma determinada questão para servir como "lembrete" da senha do usuário.
ProviderName	Recupera o nome do <i>provider</i> que o mesmo está utilizando.
ProviderUserKey	Recupera o identificador do usuário dentro da base de dados. Na base de dados, o identificador é criado e armazenado como um tipo de dado chamado <i>uniqueidentifier</i> . Como a propriedade <i>ProviderUserKey</i> retorna um Object se quiser ter algo mais tipado, terá que convertê-lo para o objeto Guid , que corresponde ao tipo de dado <i>uniqueidentifier</i> da base de dados.
UserName	Recupera o nome do usuário.

Método**Descrição****ChangePassword**

Altera a senha do usuário na base de dados.

Este método recebe dois parâmetros: senha atual e a nova senha e, retorna um valor booleano indicando se a alteração aconteceu com sucesso. Vale lembrar que as senhas informadas neste método devem estar de acordo com as especificações informadas na configuração do *provider* no arquivo Web.Config.

ChangePasswordQuestionAndAnswer Altera a pergunta e resposta que ajudam o usuário a lembrar a senha.

GetPassword Recupera a senha de um determinado usuário.

Se a propriedade **EnablePasswordRetrieval** estiver definida como **False**, uma *Exception* será atirada. Se o atributo **passwordFormat** do *provider* estiver definido como **Hashed**, a senha antiga não será recuperada, mas uma nova senha será gerada e retornada através do método **ResetPassword**.

ResetPassword Reinicializa a senha do usuário e a retorna ao chamador.

Se o atributo **enablePasswordReset** estiver definido como **False** e o método **ResetPassword** for chamado, uma *Exception* será atirada. Agora, se o atributo **requiresQuestionAndAnswer** estiver definido como **True**, você pode utilizar o método **ResetPassword**, desde que utilize a sobrecarga do mesmo, onde você deve fornecer a palavra-chave para o usuário.

UnlockUser Desbloqueia um usuário para o mesmo poder voltar a acessar a aplicação. Um valor booleano é retornado indicando se o desbloqueio foi ou não realizado com sucesso.

Depois da teoria veremos abaixo um trecho curto que mostra como chamar esses métodos e propriedades via código:

```
// Criando Usuário
MembershipCreateStatus status = MembershipCreateStatus.UserRejected;
MembershipUser user =
    Membership.CreateUser(
        "IsraelAece",
        "P@$$w0rd",
        "israel@projetando.net",
        "Questão Password",
        "Resposta Password",
        true,
        out status);
Response.Write(string.Format("Status Criação: {0}", status.ToString()));
```

```
// Recuperando Usuário
MembershipUser user = Membership.GetUser("IsraelAece");
if(user != null)
{
    Response.Write(string.Format("E-mail: {0}", user.Email));
}
```

```
// Excluindo Usuário
if(Membership.Delete("IsraelAece"))
{
    Response.Write("Usuário excluído com sucesso.");
}
```

```
// Populando um GridView
this.GridView1.AutoGenerateColumns = true;
this.GridView1.DataSource = Membership.GetAllUsers();
this.GridView1.DataBind();
```

Integração com outras tabelas

Um das principais dúvidas é como integrar a tabela de *aspnet_Membership* com outras tabelas do banco de dados, como por exemplo, uma tabela com o endereço, R.G., C.P.F., entre outros dados. Neste caso cria-se uma coluna chamada *UserId* do tipo *uniqueidentifier* nesta tabela "filha" que será

uma **ForeignKey** da tabela *aspnet_Membership*. Claro que o objeto *MembershipUser* ainda continuará com as mesmas propriedades e, apesar de um pouco complicado, você teria que sobrescrever o *provider* para poder contemplar esse novo design do seu objeto.

Para ilustrar, analise a imagem abaixo. Foi criada a relação entre as tabelas através do *aspnet_Membership.UserId* x *Colaboradores.ColaboradorID* para firmar o relacionamento entre as tabelas:

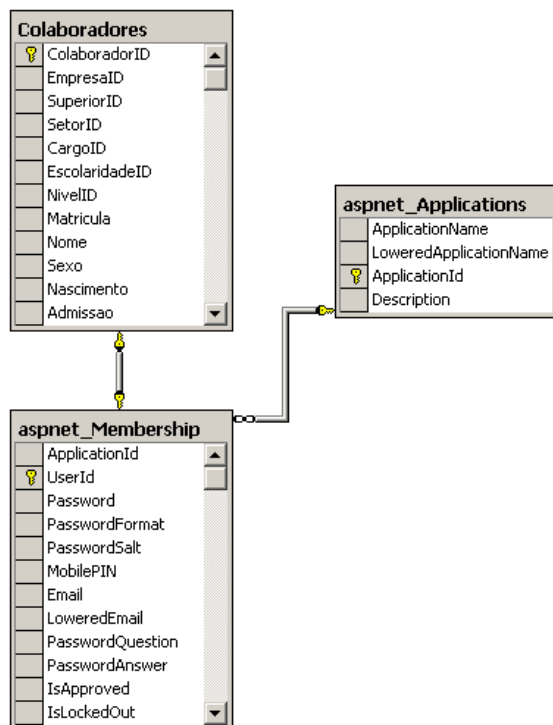


Figura 5 - Relacionamento com a tabela *aspnet_Membership*.



Israel Aécio - Especialista em tecnologias de desenvolvimento Microsoft, atua como desenvolvedor de aplicações para o mercado financeiro utilizando a plataforma .NET. Como instrutor Microsoft, leciona sobre o desenvolvimento de aplicações .NET. É palestrante em diversos eventos Microsoft no Brasil e autor de diversos artigos que podem ser lidos a partir de seu site <http://www.israelaeco.com/>. Possui as seguintes credenciais: MVP (Connected System Developer), MCP, MCAD, MCTS (Web, Windows, Distributed, ASP.NET 3.5, ADO.NET 3.5, Windows Forms 3.5 e WCF), MCPD (Web, Windows, Enterprise, ASP.NET 3.5 e Windows 3.5) e MCT.

1 Like 0

Leia também

Qualidade de Software: dicas para escrever um código de qualidade
ASP.NET

Consultar conteúdo de listas do SharePoint por Web Service

ASP. NET

Utilizando ListView para fazer um CRUD

ASP. NET

Criando RSS XML para o Seu Próprio Site

C#


Popup em destaque no site em C# e ASP.Net

C#

Estamos aqui:    

Linha de Código faz parte do grupo Web-03

[Política de privacidade e de uso](#) | [Anuncie](#) | [Cadastre-se](#) | [Fale conosco](#)

**Linha de Código**

Curtir

Você curtiu isso.

Você e outras 7.750 pessoas curtiram [Linha de Código](#).



Plug-in social do Facebook

© 2013 Linha de Código. Todos os direitos reservados