

&lt;&lt; Impacto de Timeouts no Proxy WCF | MaxItemsInObjectGraph &gt;&gt;

## Autenticação e Autorização no ASP.NET MVC

By Israel Aece

9. fevereiro 2010 08:58

Desde a primeira versão do ASP.NET, temos três formas de autenticação: *Windows*, *Forms* e *Passport*. A primeira opção consiste em permitir o acesso desde que o usuário faça parte do domínio/máquina. Essa opção torna o gerenciamento simples, facilita o *Single Sign-On* (SSO) interno, mas é útil em um ambiente controlado, como uma *intranet*. A opção *Forms* faz com que a aplicação se encarregue de gerenciar e autenticar seus usuários, baseando-se em um *cookie* que determinará quem é o usuário logado, e esta opção é a mais ideal para a *internet*. Finalmente, temos a *Passport*, que permitia autenticar o usuário utilizando o serviço de identidade da *Microsoft*, mas por haver um custo envolvido, acabou não evoluindo.

Com a vinda do *ASP.NET MVC*, grande parte dos serviços que são disponibilizados para o *ASP.NET WebForms*, já nasceram com o MVC ou estão aos poucos sendo adicionados à plataforma. Como já era de se esperar, a questão da autenticação e autorização já fazem parte desde a versão 1.0 do MVC, onde podemos utilizar a autenticação *Windows* ou *Forms*, mudando ligeiramente como devemos proceder para configurá-los e utilizá-los.

Para definir se vamos utilizar o modelo de autenticação *Windows* ou *Forms*, devemos recorrer ao elemento *authentication* no arquivo *Web.config*, como já fazíamos anteriormente. O atributo *mode* determina qual a opção de autenticação, e um sub-elemento chamado *forms*, nos permite customizar a autenticação baseada em *Forms*. O exemplo abaixo ilustra a configuração inicial:

```
<authentication mode="Forms">
  <forms loginUrl="Authentication/LogOn" />
</authentication>
```

Aqui há uma mudança radical aqui. O *ASP.NET WebForms* é baseado em arquivos, ou seja, todas as requisições, sejam elas da aplicação ou da infraestrutura (como é o caso aqui), o alvo sempre será um arquivo físico, com extensão *\*.aspx*. Já no *ASP.NET MVC*, o alvo será sempre uma ação, que nada mais é que um método dentro de uma classe, conhecida como *Controller*. No exemplo acima, o *controller* é uma classe chamada *AuthenticationController* e dentro dele há um método chamado *LogOn*. É dentro deste método que você colocará todos os passos necessários para autenticar o usuário. Geralmente esse método deve receber o *login* e a senha, mas eventualmente poderá ter algo mais rebuscado.

Quando tentamos acessar um recurso protegido através do *FormsAuthentication*, ele automaticamente redireciona o usuário para a página de autenticação configurada acima, embutindo na URL uma *query string* chamada *ReturnUrl*, com o caminho (do arquivo ou da ação) que tentamos acessar. Esse parâmetro é útil para depois que validar o usuário, redirecioná-lo para a mesma seção que ele solicitou antes de efetuar a autenticação. Sendo assim, é necessário que o método que efetua a autenticação do usuário, também receba esse parâmetro.

Em tempo, a versão 2.0 do *ASP.NET MVC* fornece um atributo chamado *HttpPostAttribute*, que permite você decorar uma determinada ação, para que ela seja executada somente via POST, garantindo assim que ela seja executada somente através desta forma. Para ilustrar, o método *LogOn* fica da seguinte forma:

```
public class AuthenticationController : Controller
{
    [HttpPost]
    public ActionResult LogOn(string userName, string password, string returnUrl)
    {
        //...
    }
}
```

É importante dizer que se você estiver utilizando a autenticação *Windows*, nada disso é necessário, já que a autenticação acaba sendo feita automaticamente pelo IIS/ASP.NET.

Se precisar saber qual usuário está autenticado naquele momento, você pode recorrer à propriedade *User*, exposta pela classe *Controller*. Assim como já acontece com a classe *Page* do *WebForms*, essa propriedade retorna a instância de uma classe que implementa a *interface IPincipal*, fornecendo acesso ao *username* do usuário atual, e um método chamado *IsInRole*, caso você precisa refinar ainda mais a autorização. Para maiores detalhes sobre essa *interface*, consulte este o capítulo 9 deste artigo .

### Autorização

A autorização pode continuar sendo realizada da mesma forma que fazíamos antes, ou seja, recorrendo ao arquivo de configuração (*Web.config*) para especificar as regras de autorização. A diferença se dá também pelo fato de que o MVC é baseado em ações, e ao invés de determinar páginas e/ou diretórios nessas regras, utilizaremos as ações para refinar o acesso dos usuários. Se temos um *controller* chamado *MonitorController* e dentro dele uma ação (método) chamada *VisualizarItens*, e o acesso à ela somente pudesse ser feita por usuários autenticados, a regra ficaria da seguinte forma:

```
<location path="/Monitor/VisualizarItens">
```

### Sobre

Meu nome é **Israel Aece** e sou especialista em tecnologias de desenvolvimento Microsoft, atuando como desenvolvedor de aplicações para o mercado financeiro utilizando a plataforma .NET. [ Mais ]




☐ Incluir comentários na busca

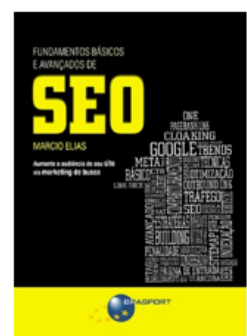
### Categorias

- .NET Framework (171)
- ASP.NET (245)
- Async (38)
- C# (31)
- CSD (113)
- Data (71)
- General (46)
- Interoperabilidade (16)
- Security (102)
- VB.NET (27)
- WCF (233)
- WIF (19)
- WPF (9)

### Tags

Arquitetura Assembly Autenticação  
 Autorização Behavior Binding  
 Caching Callbacks Cas Certificates  
 Code Collections Com+  
 Configuração Contract Debug  
 Deployment Encoding Entity Framework  
 Estensibilidade Exception  
 Globalization Hosting Http Ide  
 Identity Iis Internals Io Json Linq  
 Messagequeue Mvc Oop Partial Trust  
 Performance Provider Proxy  
 Reflection Rest Serialização Silverlight  
 Smtip Sql Server Sqlclr Talks Tcp  
 Threading Tools Transactions Ui  
 Videos Windows Services Ws-\* Wsdl Xml

### Indicações



Powered by  
 orcsweb

```
<system.web>
  <authorization>
    <deny users="?" />
  </authorization>
</system.web>
</location>
```

Caso você precise permitir o acesso à uma determinada ação para somente aqueles usuários que fazem parte de um determinado grupo (*role*), você pode configurar a regra da seguinte forma:

```
<location path="/Monitor/VisualizarItens">
  <system.web>
    <authorization>
      <allow roles="Financeiro" />
    </authorization>
  </system.web>
</location>
```

Apesar desta técnica funcionar, é difícil de manter. Para cada novo *controller* ou ação que são criados, você precisa se preocupar em olhar para o arquivo de configuração e configurar quem poderá ou não ter acesso. O problema desta técnica é que se você esquecer, usuários que talvez não deveriam ter acesso, acabarão visualizando. No *ASP.NET MVC* há uma outra forma de se trabalhar, também declarativa, mas recorrendo à atributos dentro dos *controllers*/ações, facilitando a centralização e manutenção das regras de acesso.

Para isso, o *ASP.NET MVC* fornece um atributo chamado *AuthorizeAttribute* que podemos decorar em classes (*controllers*) ou em métodos (ações). Esse atributo fornece duas propriedades do tipo *string*: *Users* e *Roles*. Na primeira propriedade, especificamos os usuários (separados por vírgula) que podem ter acesso aquele membro (classe ou método) em que o atributo está sendo aplicado. Já a segunda propriedade, *Roles*, permite especificarmos os grupos (separados por vírgula), onde somente usuários que estão dentro de um deles é que podem ter acesso ao recurso. Utilizar a propriedade *Users* não é o mais ideal, porque usuários são muito "voláteis", e em pouco tempo, essa regra provavelmente não fará mais sentido.

Como dito acima, podemos aplicar esse atributo em nível de *controller*, e assim, as suas respectivas ações irão exigir que aquela regra seja atendida antes de permitir o acesso. No primeiro exemplo, aplicamos este atributo em nível de *controller*, e todas as ações exigirão que o usuário esteja autenticado:

```
[Authorize]
public class MonitorController : Controller
{
    public ActionResult VisualizarItens()
    {
        //...
    }

    public ActionResult RecuperarIndice()
    {
        //...
    }
}
```

Já o segundo exemplo, fará com que o *controller* seja acessado somente por usuários devidamente autenticados, mas isso não é o suficiente para o método *VisualizarItens*, que além disso, exige que o usuário esteja contido no grupo *Financeiro*:

```
[Authorize]
public class MonitorController : Controller
{
    [Authorize(Roles = "Financeiro")]
    public ActionResult VisualizarItens()
    {
        //...
    }

    public ActionResult RecuperarIndice()
    {
        //...
    }
}
```

Caso você tenha uma regra de validação muito mais complexa do que isso, nada impede você de criar um filtro para efetuar essa customização, fazendo que a autorização siga os passos necessários para atender essa regra predefinida. E para finalizar, lembre-se que o MVC trata os métodos como "*opt-out*", ou seja, por padrão, todos os métodos públicos estão acessíveis. Se mesmo com a autenticação e/ou autorização ele jamais deverá ser invocado, então você deve decorá-lo com o atributo *NonActionAttribute*.

Membership e Roles

A partir da versão 2.0 do ASP.NET, uma série de novos serviços foram adicionados, e entre eles temos o *Membership Provider* e *Role Provider*, algo que já falei extensivamente nestes outros artigos . Felizmente podemos continuar utilizando esses serviços, mas com um pouco mais de cautela.

Esses serviços seguem um padrão conhecido como *Provider Model*, e ao utilizar o *Membership* ou *Roles*, eu estou trabalhando de forma independente de repositório; posso trabalhar com *SQL Server*, *Oracle* ou *Xml*, apenas

Histórico

2013	Setembro (2)
	Julho (1)
	Abril (2)
2012	
2011	
2010	
2009	
2008	
2007	
2006	
2005	
2004	
2003	

configurando os *providers* através do arquivo *Web.config*. Mas apesar desses serviços seguirem um padrão de extensibilidade bem definido, isso não quer dizer que é uma boa prática você pode utilizá-los diretamente dentro dos *controllers*.

É importante dizer que ao criar aplicações MVC, a testabilidade deve estar em foco, e utilizar as classes *Membership* ou *Roles* dentro dos *controllers*, o tornará dependente destas implementações, ou melhor, destes serviços, e que por sua vez, estão fortemente acoplados ao ASP.NET. Como disse anteriormente, a arquitetura do *provider model* é bem flexível, mas para criar uma versão "*fake*" é complexo demais, pois há uma infinidade de funcionalidades que as vezes serão desnecessárias. Além disso, utilizar essas classes torna meu *controller* dependente deste tipo de serviço mas, eventualmente, eu tenho o meu próprio repositório de usuários e suas respectivas permissões, e quero fazer uso deles ao invés daqueles que o ASP.NET fornece.

#### A classe AccountController

Quando criamos um novo projeto MVC, por padrão, uma série de recursos já são adicionados, e entre eles, temos a classe (*controller*) chamada *AccountController*. Essa classe foi desenhada para servir como um *wrapper* para os serviços de *Membership* e *FormsAuthentication*. Há vários problemas com essa classe, e o primeiro deles é aquele que descrevi acima, que é a forte dependência da API do *Membership*. Apesar da *Microsoft* ter refatorado o código e criado uma *interface* chamada *IMembershipService*, há ainda alguns tipos que estão sendo utilizados, como é o caso do enumerador *MembershipCreateStatus*. Além disso, essa classe ainda viola o princípio de SRP (*Single Responsibility Principle*), que faz com que ela faça muito mais coisas do que realmente deveria.

**Conclusão:** O fato da *Microsoft* tem se preocupado em manter grande parte dos recursos do *WebForms* no MVC, é importante que você analise cuidadosamente como incorporá-lo ao MVC para não comprometer um dos maiores benefícios do MVC, que são os testes. É importante notar que grande parte do conhecimento adquirido no *ASP.NET WebForms*, acaba sendo reutilizado aqui, apenas com ligeiras mudanças devido à arquitetura do MVC.

Currently rated 5.0 by 3 people

Tags: autenticação, autorização, mvc, provider

ASP.NET | Security

Permalink | Comentários (16)

## Posts relacionados

Autenticação via Claims no ASP.NET WebForms

Nos artigos anteriores, falei sobre a proposta da Microsoft para tentar unificar o sistema de autent...

WCF - Segurança

Um dos grandes desafios de um software é a segurança do mesmo. Em qualquer software ho...

Explorando o LINQ - Background

Antes de efetivamente aprendermos sobre o LINQ e suas respectivas funcionalidades, não podemos...

## Comentários

09/02/2010 09:50:48 #



RT @israelaece: Autenticação e Autorização no ASP.NET MVC:  
http://www.israelaece.com/post/Autenticacao-e-Autorizacao-no-ASPNET-MVC.aspx"  
rel="nofollow">www.israelaece.com/.../...zacao-no-ASPNET-MVC.aspx

RT @israelaece : Autenticação e Autorização no ASP.NET MVC :  
http://www.israelaece.com/post/Autenticacao

Twitter Mirror

09/02/2010 14:41:21 #



Pingback from topsy.com

Twitter Trackbacks for

Israel Aece | AutenticaÃ§Ã£o e AutorizaÃ§Ã£o no ASP.NET MVC  
[israelaece.com]  
on Topsy.com

topsy.com

09/02/2010 15:47:22 #



Israel, tem como eu negar todo o acesso no web.config. e ir liberando as actions públicas?

eduardo spaki

10/02/2010 06:40:24 #



Boas Eduardo,

Você pode fazer um lock down na sua aplicação para permitir somente usuários autenticados acessarem os controllers:

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

E depois refinar isso controller ou via ação, com o atributo *AuthorizeAttribute*.

IsraelAece

17/02/2010 13:38:41 #



Mas daí teria como eu liberar só o controller de login por exemplo?

eduardo spaki

17/02/2010 13:56:36 #



Boas Eduardo,

Por padrão, o Controller + Action que representa o login (que está configurado no elemento <forms />), não é bloqueado.

IsraelAece

22/02/2010 16:11:30 #



Israel,

ótimo post. gostaria muito de ver alguém implementando a autenticação propria no mvc. é possível? ao invés de usar a autenticação padrão do asp.net mvc (MembershipProvider)

Rodrigo Nunes

22/02/2010 22:09:49 #



Boas Rodrigo,

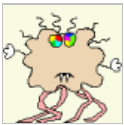
O Membership é um serviço interessante que já está pronto e testado.

Mas se quiser criar algo próprio, você está livre, basta criar toda a estrutura necessária para os membros do teu site, e também as classes necessárias para manipulá-la.

Ou você se refere ao FormsAuthentication?

IsraelAece

25/02/2010 09:41:57 #



Parabens bom artigo ainda mais nesse tempo onde o .net mvc esta evoluindo.

Israel será que você pode mostrar um AccountController de exemplo ou me enviar por email?

Vinicius

25/02/2010 10:32:11 #

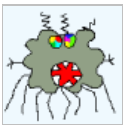


Boas Vinicius,

Para acessá-la, basta criar uma aplicação ASP.NET MVC, que ela já virá juntamente com a template.

IsraelAece

06/05/2010 12:31:10 #



Israel, boa tarde!

Estou criando em asp um sistema de formulario, onde possui consulta, inclusao, alteração e exclusao de um cadastro. Até ai tudo bem, só que preciso definir usuarios e nivel de acesso, por exemplo: Administrador, Usuario Vip e Usuario Simples.

O Administrador tem acesso a consulta, inclusao, alteração e exclusao, a do cadastro principal e da inclusao de um novo usuario.

O Usuario Vip tem acesso a consulta, inclusao, alteração e exclusao do cadastro.

O Usuario Simples tem acesso a inclusao e consulta.

Como faço para definir os niveis, e quando a pessoa fizer o login ir direto a pagina destinada ao nivel dessa pessoa?

Jenifer

06/05/2010 15:47:04 #



Boas Jenifer,

Para proteger cada action de um usuário, você pode utilizar o atributo AuthorizeAttribute, separando com vírgula os papéis que podem acessar aquele método.

Já para direcionar o usuário após o login para um local dependendo do seu perfil, então terá que fazer isso manualmente. Depois do login, você pode direcioná-la para uma action que, baseando-se no perfil do usuário, o redirecionará para a página/action correspondente.

IsraelAece

22/06/2010 09:48:27 #

Israel, boa tarde!

No meu caso os acessos as telas do sistemas são definidos por grupos(perfis)...

Só que a minha regra para cada action(consulta, inclusão, alteração, emitir nota, cancelar nota, etc...) está cadastrada no banco de dados.

Qual seria a melhor maneira de efetuar as restrições neste caso?



Grato

Renato 

22/06/2010 21:56:57 #

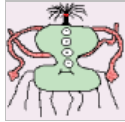
Boas Renato,




Se entendi a sua dúvida, você tem que decorar as suas actions com atributos AuthorizeAttribute, definindo que somente usuários que estão dentro desta role é que podem acessar a action, ou seja, exatamente como foi feito neste artigo.

IsraelAece 

25/06/2010 18:08:03 #



Israel, minha dúvida é se o Membership é seguro contra ataques de SQL Injection pretendo fazer um projeto e publica-lo na web, dai a pergunta se alguém saber q estou usando p Banco de dados do Aspnet alguém pode tentar deletar alguma tabela ou remover registros pelo fato deste cara saber os nomes das tabelas e seus campos.

mateus 

25/06/2010 18:12:40 #

Boas Mateus,



O Membership é seguro contra isso. A Microsoft se preocupa bastante com segurança, e a forma como ele foi construída está imune a esse tipo de problema.

IsraelAece 

Os comentários estão fechados

Powered by BlogEngine.NET 1.5.0.0  
Theme by Mads Kristensen