

# Contents

[Documentação do ASP.NET Core](#)

[Visão geral](#)

[Sobre o ASP.NET Core](#)

[Comparar o ASP.NET Core e o ASP.NET](#)

[Comparar o .NET Core e o .NET Framework](#)

[Introdução](#)

[O que há de novo](#)

[Novidades no 2.2](#)

[Novidades no 2.1](#)

[Novidades no 2.0](#)

[Novidades no 1.1](#)

[Tutoriais](#)

[Aplicativos API Web](#)

[Criar uma API Web](#)

[API Web com o MongoDB](#)

[Back-end para aplicativos móveis](#)

[Aplicativos Web](#)

[Páginas do Razor](#)

[MVC](#)

[Aplicativos Web em tempo real](#)

[SignalR com JavaScript](#)

[SignalR com TypeScript](#)

[Acesso aos dados](#)

[EF Core com Razor Pages](#)

[EF Core com MVC, BD existente](#)

[EF Core com MVC, BD novo](#)

[EF Core com MVC, tutorial longo](#)

[Princípios básicos](#)

[Visão geral](#)

[Inicialização do aplicativo](#)

[Injeção de dependência \(serviços\)](#)

[Roteamento](#)

[Ambientes \(desenvolvimento, preparação, produção\)](#)

[Configuração](#)

[Opções](#)

[Registrando em log](#)

[Tratar erros](#)

[Middleware](#)

[Host](#)

[Visão geral](#)

[Host da Web](#)

[Host Genérico](#)

[Servidores](#)

[Fazer solicitações HTTP](#)

[Aplicativos Web](#)

[Páginas do Razor](#)

[Introdução](#)

[Tutorial](#)

[Visão geral](#)

[Introdução](#)

[Adicionar um modelo](#)

[Scaffolding](#)

[Trabalhar com um BD](#)

[Atualizar as páginas](#)

[Adicionar pesquisa](#)

[Adicionar um novo campo](#)

[Adicionar validação](#)

[Filtros](#)

[Bibliotecas de classes Razor](#)

[Convenções de aplicativo e roteamento](#)

[Carregar arquivos](#)

[SDK do Razor](#)

[MVC](#)

[Visão geral](#)

[Tutorial](#)

[Introdução](#)

[Adicionar um controlador](#)

[Adicionar uma exibição](#)

[Adicionar um modelo](#)

[Trabalhar com um BD](#)

[Exibições e ações do controlador](#)

[Adicionar pesquisa](#)

[Adicionar um novo campo](#)

[Adicionar validação](#)

[Examinar os métodos Details e Delete](#)

[Exibições](#)

[Exibições parciais](#)

[Controladores](#)

[Roteamento](#)

[Uploads de arquivo](#)

[Injeção de dependência – controladores](#)

[Injeção de dependência – exibições](#)

[Teste de unidade](#)

[Componentes Razor](#)

[Estado de sessão e de aplicativo](#)

[Auxiliares de Marca](#)

[Visão geral](#)

[Criar auxiliares de marcação](#)

[Usar auxiliares de marcação em formulários](#)

[Componentes do auxiliar de marcação](#)

[Auxiliares de marcação internos](#)

[Âncora](#)

[Cache](#)

- [Cache distribuído](#)
- [Ambiente](#)
- [Formulário](#)
- [Image](#)
- [Entrada](#)
- [Rotular](#)
- [Parcial](#)
- [Selecionar](#)
- [Área de texto](#)
- [Mensagem de validação](#)
- [Resumo de validação](#)
- [Layout](#)
- [Arquivos estáticos](#)
- [Model binding](#)
- [Validação de modelo](#)
- [Sintaxe Razor](#)
- [Avançado](#)
  - [Componentes da exibição](#)
  - [Exibir compilação](#)
  - [Modelo de aplicativo](#)
  - [Filtros](#)
  - [Áreas](#)
  - [Partes do aplicativo](#)
  - [Model binding personalizado](#)
  - [Versão de compatibilidade](#)
- [Aplicativos API Web](#)
  - [Visão geral](#)
  - [Tutoriais](#)
    - [Criar uma API Web](#)
    - [API Web com o MongoDB](#)
  - [Swagger/OpenAPI](#)
  - [Visão geral](#)

[Introdução ao Swashbuckle](#)

[Introdução ao NSwag](#)

[Tipos de retorno de ação](#)

[Formatar dados de resposta](#)

[Formatadores personalizados](#)

[Analisadores](#)

[Convenções](#)

[Aplicativos em tempo real](#)

[Visão geral do SignalR](#)

[Plataformas com suporte](#)

[Tutoriais](#)

[SignalR com JavaScript](#)

[SignalR com TypeScript](#)

[Exemplos](#)

[Conceitos de servidor](#)

[Hubs](#)

[Enviar de fora de um hub](#)

[Usuários e grupos](#)

[Publicar no Azure](#)

[Considerações sobre o design da API](#)

[Clientes](#)

[Cliente .NET](#)

[Referência da API REST](#)

[Cliente Java](#)

[Referência da API Java](#)

[Cliente JavaScript](#)

[Referência da API JavaScript](#)

[Hospedagem e dimensionamento](#)

[Visão geral](#)

[Serviço Azure SignalR](#)

[Backplane de Redis](#)

[SignalR com serviços em segundo plano](#)

[Configuração](#)

[Autenticação e autorização](#)

[Considerações sobre segurança](#)

[Protocolo de Hub do MessagePack](#)

[Streaming](#)

[Comparar o SignalR e o SignalR Core](#)

[WebSockets sem SignalR](#)

[Testar, depurar e solucionar problemas](#)

[Teste de unidade](#)

[Testes de unidades de páginas Razor](#)

[Controladores de teste](#)

[Depuração remota](#)

[Depuração de instantâneo](#)

[Depurando de instantâneo no Visual Studio](#)

[Testes de integração](#)

[Testes de estresse e carga](#)

[Solução de problemas](#)

[Registrando em log](#)

[Acesso aos dados](#)

[Tutoriais](#)

[EF Core com Razor Pages](#)

[Visão geral](#)

[Introdução](#)

[Criar, ler, atualizar e excluir](#)

[Classificar, filtrar, paginar e agrupar](#)

[Migrações](#)

[Criar um modelo de dados complexo](#)

[Ler dados relacionados](#)

[Atualizar dados relacionados](#)

[Tratar conflitos de simultaneidade](#)

[EF Core com MVC, BD novo](#)

[EF Core com MVC, BD existente](#)

## [EF Core com MVC, tutorial longo](#)

[Visão geral](#)

[Introdução](#)

[Criar, ler, atualizar e excluir](#)

[Classificar, filtrar, paginar e agrupar](#)

[Migrações](#)

[Criar um modelo de dados complexo](#)

[Ler dados relacionados](#)

[Atualizar dados relacionados](#)

[Tratar conflitos de simultaneidade](#)

[Herança](#)

[Tópicos avançados](#)

## [EF 6 com ASP.NET Core](#)

### [Armazenamento do Azure com o Visual Studio](#)

[Serviços Conectados](#)

[Armazenamento de Blobs](#)

[Armazenamento de filas](#)

[Armazenamento de tabelas](#)

## [Desenvolvimento do lado do cliente](#)

[Visão geral](#)

[Gulp](#)

[Grunt](#)

[LibMan](#)

[Visão geral](#)

[CLI](#)

[Visual Studio](#)

[Bower](#)

[LESS, Sass e Font Awesome](#)

[Agrupar e minificar](#)

[Link do navegador](#)

[Componentes Razor](#)

[Visão geral](#)

- [Introdução](#)
- [Criar seu primeiro aplicativo](#)
- [Componentes](#)
- [Bibliotecas de componentes](#)
- [Layouts](#)
- [Injeção de dependência](#)
- [Roteamento](#)
- [Interoperabilidade do JavaScript](#)
- [Depurar](#)
- [Modelos de hospedagem](#)
- [Hospedar e implantar](#)
  - [Visão geral](#)
  - [Configurar o vinculador](#)
- [Aplicativos de página única](#)
  - [Visão geral](#)
  - [Angular](#)
  - [React](#)
  - [React com Redux](#)
  - [JavaScriptServices](#)
- [Hospedagem e implantação](#)
  - [Visão geral](#)
  - [Hospedar no Serviço de Aplicativo do Azure](#)
    - [Visão geral](#)
    - [Publicar com o Visual Studio](#)
    - [Publicar com Visual Studio para Mac](#)
    - [Publicar com as ferramentas da CLI](#)
    - [Publicar com Visual Studio e Git](#)
    - [Implantação contínua com o Azure Pipelines](#)
    - [Módulo do ASP.NET Core](#)
    - [Solução de problemas](#)
    - [Referência de erros](#)
  - [DevOps](#)

- [Visão geral](#)
- [Ferramentas e downloads](#)
- [Implantar no Serviço de Aplicativo](#)
- [Integração contínua e implantação](#)
- [Monitorar e solucionar problemas](#)
- [Próximas etapas](#)
- [Hospedar no Windows com o IIS](#)
  - [Visão geral](#)
  - [Módulo do ASP.NET Core](#)
  - [Suporte ao IIS no Visual Studio](#)
  - [Módulos do IIS](#)
  - [Solução de problemas](#)
  - [Referência de erros](#)
- [Kestrel](#)
- [HTTP.sys](#)
- [Hospedar em um serviço Windows](#)
- [Hospedar em Linux com o Nginx](#)
- [Hospedar em Linux com o Apache](#)
- [Hospedar no Docker](#)
  - [Visão geral](#)
  - [Compilar imagens do Docker](#)
  - [Ferramentas do Visual Studio](#)
  - [Publicar em uma imagem do Docker](#)
  - [Imagens do Docker de amostra](#)
- [Configuração de平衡ador de carga e de proxy](#)
- [Hospedar em uma web farm](#)
- [Perfis de publicação do Visual Studio](#)
- [Visual Studio para Mac – Publicar em uma pasta](#)
- [Estrutura do diretório](#)
- [Verificações de integridade](#)
- [Componentes Razor](#)
  - [Visão geral](#)

[Configurar o vinculador](#)

[Segurança e identidade](#)

[Visão geral](#)

[Autenticação](#)

[Introdução ao Identity](#)

[Identidade Scaffold](#)

[Adicionar dados de usuário personalizados à Identidade](#)

[Exemplos de autenticação](#)

[Personalizar Identidade](#)

[Opções de autenticação de OSS da comunidade](#)

[Configurar o Identity](#)

[Configurar a Autenticação do Windows](#)

[Provedores de armazenamento personalizados para o Identity](#)

[Google, Facebook...](#)

[Visão geral](#)

[Autenticação do Google](#)

[Autenticação do Facebook](#)

[Autenticação da Microsoft](#)

[Autenticação do Twitter](#)

[Outros provedores](#)

[Declarações adicionais](#)

[Autenticação de Web Services Federation](#)

[Confirmação de conta e recuperação de senha](#)

[Habilitar a geração de código QR no Identity](#)

[Autenticação de dois fatores com SMS](#)

[Usar a autenticação de cookie sem o Identity](#)

[Azure Active Directory](#)

[Visão geral](#)

[Integrar o Azure AD em um aplicativo Web](#)

[Integrar o Azure AD B2C em um aplicativo Web](#)

[Integrar o Azure AD B2C em uma API Web](#)

[Chamar uma API Web do WPF](#)

- [Chamar uma API Web em um aplicativo Web usando o Azure AD](#)
- [Proteger aplicativos ASP.NET Core com o IdentityServer4](#)
- [Proteger aplicativos ASP.NET Core com a Autenticação do Serviço de Aplicativo do Azure \(autenticação fácil\)](#)
- [Contas de usuários individuais](#)
- [Autorização](#)
  - [Visão geral](#)
  - [Criar um aplicativo Web com autorização](#)
  - [Convenções de autorização de Páginas Razor](#)
  - [Autorização simples](#)
    - [Autorização baseada em função](#)
    - [Autorização baseada em declarações](#)
    - [Autorização baseada em política](#)
    - [Provedores da política de autorização](#)
    - [Injeção de dependência em manipuladores de requisitos](#)
    - [Autorização baseada em recursos](#)
    - [Autorização baseada em exibição](#)
    - [Limitar a identidade por esquema](#)
  - [Proteção de dados](#)
    - [Visão geral](#)
    - [APIs de Proteção de Dados](#)
    - [APIs de consumidor](#)
      - [Visão geral](#)
        - [Cadeias de caracteres de finalidade](#)
        - [Multilocação e hierarquia de finalidade](#)
        - [Senhas hash](#)
        - [Limitar o tempo de vida de cargas protegidas](#)
        - [Desproteger cargas cujas chaves foram revogadas](#)
    - [Configuração](#)
      - [Visão geral](#)
      - [Configurar a proteção de dados](#)
      - [Configurações padrão](#)
      - [Política ampla de computador](#)

Cenários sem reconhecimento de DI

APIs de extensibilidade

Visão geral

Extensibilidade da criptografia básica

Extensibilidade de gerenciamento de chaves

APIs diversas

Implementação

Visão geral

Detalhes de criptografia autenticada

Derivação de subchaves e criptografia autenticada

Cabeçalhos de contexto

Gerenciamento de chaves

Provedores de armazenamento de chaves

Criptografia de chave em repouso

Imutabilidade de chave e configurações

Formato do armazenamento de chaves

Provedores de proteção de dados efêmeros

Compatibilidade

Visão geral

Substituir machineKey no ASP.NET

Proteger segredos no desenvolvimento

Impor o HTTPS

Compatível com o RGPD (Regulamento Geral sobre a Proteção de Dados) da UE

Provedor de configuração do Azure Key Vault

Falsificação antissolicitação

Prevenir ataques de redirecionamento abertos

Evitar scripts entre sites

Habilitar o CORS (Solicitações Entre Origens)

Compartilhar cookies entre aplicativos

Lista segura de IP

Desempenho

Visão geral

- [Cache de resposta](#)
  - [Visão geral](#)
  - [Cache na memória](#)
  - [Cache distribuído](#)
  - [Middleware de cache de resposta](#)
  - [Compactação de resposta](#)
  - [Ferramentas de diagnóstico](#)
  - [Testes de estresse e carga](#)
  - [Globalização e localização](#)
- [Visão geral](#)
- [Localização de portable object](#)
- [Avançado](#)
  - [Reescrita de URL](#)
  - [Provedores de arquivo](#)
  - [Interfaces de recurso de solicitação](#)
  - [Acessar o HttpContext](#)
  - [Alterar tokens](#)
  - [OWIN \(Open Web Interface para .NET\)](#)
  - [Tarefas em segundo plano com serviços hospedados](#)
  - [Hospedando assemblies de inicialização](#)
  - [Metapacote Microsoft.AspNetCore.App](#)
  - [Metapacote Microsoft.AspNetCore.All](#)
  - [Fazendo log com o LoggerMessage](#)
  - [Usar um inspetor de arquivo](#)
  - [Middleware de fábrica](#)
  - [Middleware baseado em fábrica com contêiner de terceiros](#)
- [Migração](#)
  - [2.2 a 3.0](#)
  - [2.1 a 2.2](#)
  - [2.0 a 2.1](#)
  - [1.x a 2.0](#)
- [Visão geral](#)

[Autenticação e identidade](#)

[ASP.NET para ASP.NET Core](#)

[Visão geral](#)

[MVC](#)

[API Web](#)

[Configuração](#)

[Autenticação e identidade](#)

[ClaimsPrincipal.Current](#)

[Associação de identidade](#)

[Módulos HTTP para middleware](#)

[Registro em log \(não o ASP.NET Core\)](#)

[Reference API](#)

[Contribuir](#)

# Introdução ao ASP.NET Core

17/01/2019 • 9 minutes to read • [Edit Online](#)

Por [Daniel Roth](#), [Rick Anderson](#) e [Shaun Luttin](#)

O ASP.NET Core é uma estrutura de [software livre](#), de multiplataforma e alto desempenho para a criação de aplicativos modernos conectados à Internet e baseados em nuvem. Com o ASP.NET Core, você pode:

- Compilar aplicativos e serviços Web, aplicativos [IoT](#) e back-ends móveis.
- Usar suas ferramentas de desenvolvimento favoritas no Windows, macOS e Linux.
- Implantar na nuvem ou local.
- Executar no [.NET Core](#) ou no [.NET Framework](#).

## Por que usar o ASP.NET Core?

Milhões de desenvolvedores usaram (e continuam usando) o [ASP.NET 4.x](#) para criar aplicativos Web. O ASP.NET Core é uma reformulação do ASP.NET 4.x, com alterações de arquitetura que resultam em uma estrutura mais enxuta e modular.

O ASP.NET Core oferece os seguintes benefícios:

- Uma história unificada para a criação da interface do usuário da Web e das APIs Web.
- Projetado para capacidade de teste.
- O [Razor Pages](#) torna a codificação de cenários focados em página mais fácil e produtiva.
- Capacidade de desenvolver e executar no Windows, macOS e Linux.
- De software livre e [voltado para a comunidade](#).
- Integração de [estruturas modernas do lado do cliente](#) e fluxos de trabalho de desenvolvimento.
- Um [sistema de configuração](#) pronto para a nuvem, baseado no ambiente.
- [Injeção de dependência](#) interna.
- Um pipeline de solicitação HTTP leve, modular e de [alto desempenho](#).
- A capacidade de hospedar no [IIS](#), [Nginx](#), [Apache](#), [Docker](#) ou hospedar em seu próprio processo.
- Controle de versão do aplicativo do lado a lado ao direcionar [.NET Core](#).
- Ferramentas que simplificam o moderno desenvolvimento para a Web.

## Compilar APIs Web e uma interface do usuário da Web usando o ASP.NET Core MVC

O ASP.NET Core MVC fornece recursos que ajudam você a compilar [APIs Web](#) e [aplicativos Web](#):

- O [padrão MVC \(Model-View-Controller\)](#) ajuda a tornar as APIs Web e os aplicativos Web testáveis.
- O [Razor Pages](#) é um modelo de programação baseado em página que torna mais fácil e produtiva a criação da interface do usuário da Web.
- A [marcação Razor](#) fornece uma sintaxe produtiva para [Páginas Razor](#) e as [Exibições do MVC](#).
- Os [Auxiliares de Marcação](#) permitem que o código do servidor participe da criação e renderização de elementos HTML em arquivos do Razor.
- O suporte interno para [vários formatos de dados e negociação de conteúdo](#) permite que as APIs Web alcancem uma ampla gama de clientes, incluindo navegadores e dispositivos móveis.
- O [model binding](#) mapeia automaticamente os dados de solicitações HTTP para os parâmetros de método de

ação.

- A [Validação de Modelos](#) executa automaticamente a validação no lado do cliente e do servidor.

## Desenvolvimento do lado do cliente

ASP.NET Core integra-se perfeitamente com estruturas conhecidas do lado do cliente e bibliotecas, incluindo [Angular](#), [React](#) e [Bootstrap](#). Para saber mais, consulte [Desenvolvimento do lado do cliente](#).

## ASP.NET Core direcionado para o .NET Framework

O ASP.NET Core 2.x pode ser direcionado para o .NET Core ou ao .NET Framework. Os aplicativos do ASP.NET Core direcionados ao .NET Framework não são multiplataforma,— são executados somente no Windows. Em geral, o ASP.NET Core 2.x é composto de bibliotecas do [.NET Standard](#). Aplicativos criados com o .NET Standard 2.0 são executados em qualquer lugar com suporte para ele.

O ASP.NET Core 2.x dá suporte para as versões do .NET Framework compatíveis com o .NET Standard 2.0:

- O .NET Framework 4.7.1 e versões posteriores são fortemente recomendados.
- .NET Framework 4.6.1 e versões posteriores.

O ASP.NET Core 3.0 e posterior somente executará no .NET Core. Para obter mais detalhes sobre essa alteração, confira [A first look at changes coming in ASP.NET Core 3.0](#) (Uma primeira análise das alterações no ASP.NET Core 3.0).

Há várias vantagens em direcionar para o .NET Core, e essas vantagens aumentam com cada versão. Algumas vantagens do .NET Core em relação ao .NET Framework incluem:

- Multiplataforma. É executado no Windows, no Linux e no macOS.
- Desempenho aprimorado
- Controle de versão lado a lado
- Novas APIs
- Código Aberto

Estamos trabalhando duro para diminuir a diferença de API entre o .NET Framework e o .NET Core. O [Pacote de Compatibilidade do Windows](#) disponibilizou milhares de APIs exclusivas do Windows no .NET Core. Essas APIs não estavam disponíveis no .NET Core 1.x.

## Como baixar uma amostra

Muitos dos artigos e tutoriais incluem links para exemplos de código.

1. [Baixe o arquivo zip do repositório ASP.NET](#).
2. Descompacte o arquivo *Docs-master.zip*.
3. Use a URL no link de exemplo para ajudá-lo a navegar até o diretório de exemplo.

### Diretivas do pré-processador no código de exemplo

Para demonstrar vários cenários, os aplicativos de exemplo usam as instruções C# `#define` e

`#if/#else/#elif/#endif` para compilar e executar diferentes seções de código de exemplo de forma seletiva. Para esses exemplos que usam essa abordagem, defina a instrução `#define` na parte superior dos arquivos C# para o símbolo associado ao cenário que deseja executar. Alguns exemplos podem exigir que você defina o símbolo na parte superior de vários arquivos para executar um cenário.

Por exemplo, a seguinte lista de símbolo `#define` indica que quatro cenários estão disponíveis (um cenário por símbolo). A configuração da amostra atual executa o cenário `TemplateCode`:

```
#define TemplateCode // or LogFromMain or ExpandDefault or FilterInCode
```

Para alterar a amostra que executará o cenário `ExpandDefault`, defina o símbolo `ExpandDefault` e deixe os símbolos restantes comentados de fora:

```
#define ExpandDefault // TemplateCode or LogFromMain or FilterInCode
```

Para obter mais informações sobre como usar [diretivas de pré-processador C#](#) para compilar seletivamente as seções de código, consulte [#define \(Referência C#\)](#) e [#if \(Referência C#\)](#).

## Regiões no código de exemplo

Alguns aplicativos de exemplo contém seções de código cercadas pelas instruções C# `#region` e `#endregion`. O sistema de build de documentação injeta essas regiões nos tópicos renderizados da documentação.

Os nomes das regiões geralmente contêm a palavra "snippet". O exemplo a seguir mostra uma região chamada `snippet_FilterInCode`:

```
#region snippet_FilterInCode
WebHost.CreateDefaultBuilder(args)
    .UseStartup<Startup>()
    .ConfigureLogging(logging =>
        logging.AddFilter("System", LogLevel.Debug)
            .AddFilter<DebugLoggerProvider>("Microsoft", LogLevel.Trace))
        .Build();
#endregion
```

O snippet de código C# precedente é referenciado no arquivo de markdown do tópico com a seguinte linha:

```
[!code-csharp[](sample/SampleApp/Program.cs?name=snippet_FilterInCode)]
```

Você pode ignorar (ou remover) com segurança as instruções `#region` e `#endregion` que envolvem o código. Não altere o código dentro dessas instruções se você planeja executar os cenários de exemplo descritos no tópico. Fique à vontade para alterar o código ao experimentar com outros cenários.

Para obter mais informações, veja [Contribuir para a documentação do ASP.NET: snippets de código](#).

## Próximas etapas

Para obter mais informações, consulte os seguintes recursos:

- [Introdução a Páginas do Razor](#)
- [Publicar um aplicativo ASP.NET Core no Azure com o Visual Studio](#)
- [Conceitos básicos do ASP.NET Core](#)
- O [Community Standup semanal do ASP.NET](#) aborda o progresso e os planos da equipe. Ele apresenta o novo software de terceiros e blogs.

# Escolher entre o ASP.NET 4.x e o ASP.NET Core

08/01/2019 • 3 minutes to read • [Edit Online](#)

O ASP.NET Core é uma reformulação do ASP.NET 4.x. Este artigo lista as diferenças entre eles.

## ASP.NET Core

O ASP.NET Core é uma estrutura de software livre, multiplataforma, para a criação de aplicativos Web modernos e baseados em nuvem, no Windows, no macOS ou no Linux.

O ASP.NET Core oferece os seguintes benefícios:

- Uma história unificada para a criação da interface do usuário da Web e das APIs Web.
- Projetado para capacidade de teste.
- O [Razor Pages](#) torna a codificação de cenários focados em página mais fácil e produtiva.
- Capacidade de desenvolver e executar no Windows, macOS e Linux.
- De software livre e [voltado para a comunidade](#).
- Integração de [estruturas modernas do lado do cliente](#) e fluxos de trabalho de desenvolvimento.
- Um [sistema de configuração](#) pronto para a nuvem, baseado no ambiente.
- [Injeção de dependência](#) interna.
- Um pipeline de solicitação HTTP leve, modular e de [alto desempenho](#).
- A capacidade de hospedar no [IIS](#), [Nginx](#), [Apache](#), [Docker](#) ou hospedar em seu próprio processo.
- Controle de versão do aplicativo do lado a lado ao direcionar [.NET Core](#).
- Ferramentas que simplificam o moderno desenvolvimento para a Web.

## ASP.NET 4.x

O ASP.NET 4.x é uma estrutura consolidada que fornece os serviços necessários para criar aplicativos Web baseados em servidor, de nível empresarial, no Windows.

## Seleção de estrutura

A tabela a seguir compara o ASP.NET Core com o ASP.NET 4.x.

ASP.NET CORE	ASP.NET 4.X
Build para Windows, macOS ou Linux	Build para Windows
Páginas Razor é a abordagem recomendada para criar uma interface do usuário da Web começando com o ASP.NET Core 2.x. Confira também <a href="#">MVC</a> , <a href="#">API Web</a> e <a href="#">SignalR</a> .	Use o <a href="#">Web Forms</a> , o <a href="#">SignalR</a> , o <a href="#">MVC</a> , a <a href="#">API Web</a> , <a href="#">Webhooks</a> ou <a href="#">páginas da Web</a>
Várias versões por computador	Uma versão por computador
Desenvolva com o Visual Studio, <a href="#">Visual Studio para Mac</a> ou <a href="#">Visual Studio Code</a> usando o C# ou o F#	Desenvolva com o Visual Studio usando o C#, VB ou F#
Desempenho superior ao do ASP.NET 4.x	Bom desempenho

ASP.NET CORE	ASP.NET 4.X
Escolha o .NET Framework ou o tempo de execução do .NET Core	Use o tempo de execução do .NET Framework

Confira [ASP.NET Core targeting .NET Framework](#) (ASP.NET Core direcionado para o .NET Framework) para obter informações sobre o suporte do ASP.NET Core 2.x no .NET Framework.

## Cenários do ASP.NET Core

- [Páginas Razor](#) é a abordagem recomendada para criar uma interface do usuário da Web começando com o ASP.NET Core 2.x.
- [Sites](#)
- [APIs](#)
- [Em tempo real](#)
- [Implantar um aplicativo ASP.NET Core no Azure](#)

## Cenários do ASP.NET 4.x

- [Sites](#)
- [APIs](#)
- [Em tempo real](#)
- [Criar um aplicativo Web ASP.NET 4.x no Azure](#)

## Recursos adicionais

- [Introdução ao ASP.NET](#)
- [Introdução ao ASP.NET Core](#)
- [Implantar aplicativos ASP.NET Core no Serviço de Aplicativo do Azure](#)

# Tutorial: Introdução ao ASP.NET Core

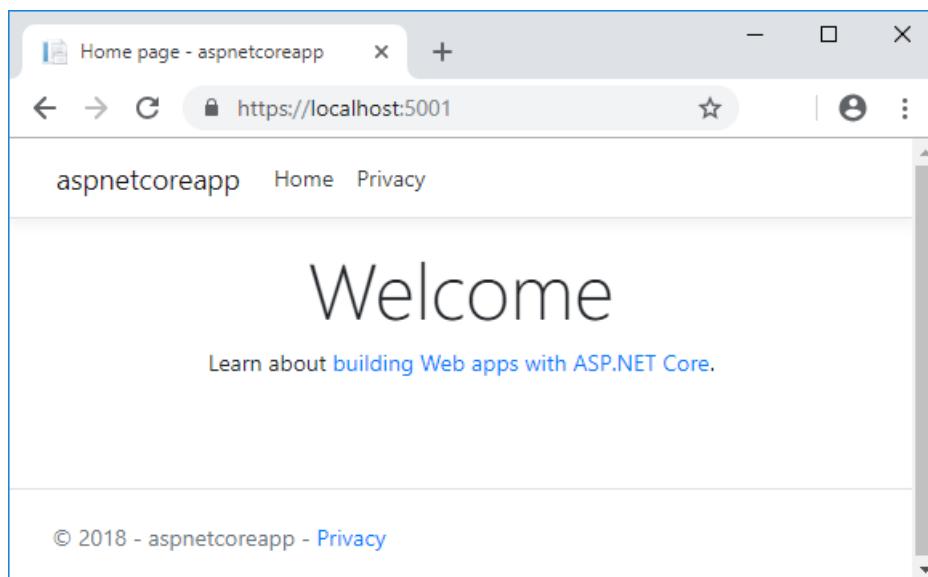
17/01/2019 • 3 minutes to read • [Edit Online](#)

Este tutorial mostra como usar a interface de linha de comando do .NET Core para criar um aplicativo Web ASP.NET Core.

Você aprenderá como:

- Criar um projeto de aplicativo Web.
- Habilitar o HTTPS local.
- Executar o aplicativo.
- Editar uma página do Razor.

No final, você terá um aplicativo Web de trabalho em execução no seu computador local.



## Pré-requisitos

- [SDK do .NET Core 2.2](#)

## Criar um projeto de aplicativo Web

Abra um shell de comando e insira o seguinte comando:

```
dotnet new webapp -o aspnetcoreapp
```

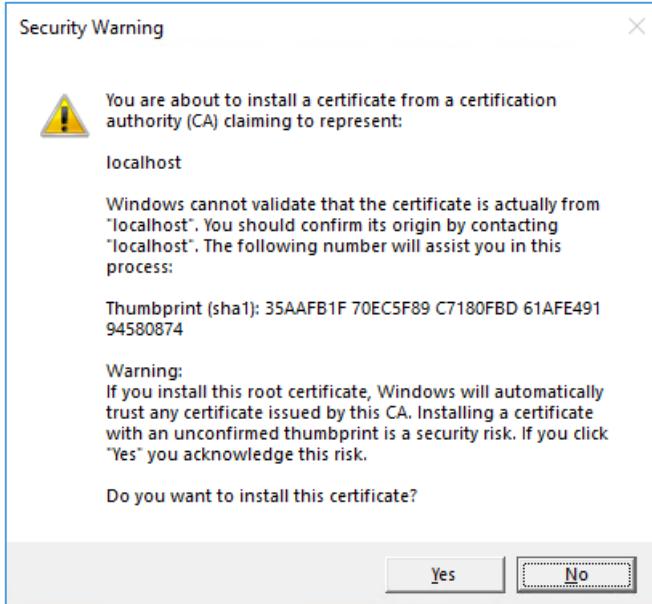
## Habilitar o HTTPS local

Confie no certificado de desenvolvimento HTTPS:

- [Windows](#)
- [macOS](#)
- [Linux](#)

```
dotnet dev-certs https --trust
```

O comando anterior exibe a caixa de diálogo a seguir:



Selecione **Sim** se você concordar com confiar no certificado de desenvolvimento.

## Executar o aplicativo

Execute os seguintes comandos:

```
cd aspnetcoreapp  
dotnet run
```

Depois que o shell de comando indicar que o aplicativo foi iniciado, navegue até <https://localhost:5001>. Clique em **Aceitar** para aceitar a política de privacidade e cookies. Este aplicativo não armazena informações pessoais.

## Editar uma página do Razor

Abra *Pages/Index.cshtml* e modifique a página com a seguinte marcação realçada:

```
@page  
@model IndexModel  
{  
    ViewData["Title"] = "Home page";  
  
    <div class="text-center">  
        <h1 class="display-4">Welcome</h1>  
        <p>Hello, world! The time on the server is @DateTime.Now</p>  
    </div>
```

Navegue até <https://localhost:5001> e verifique se as alterações são exibidas.

## Próximas etapas

Neste tutorial, você aprendeu como:

- Criar um projeto de aplicativo Web.

- Habilite o HTTPS local.
- Execute o projeto.
- Faça uma alteração.

Para saber mais sobre o ASP.NET Core, confira a introdução:

[Introdução ao ASP.NET Core](#)

# Novidades do ASP.NET Core 2.2

30/01/2019 • 11 minutes to read • [Edit Online](#)

Este artigo destaca as alterações mais significativas no ASP.NET Core 2.2, com links para a documentação relevante.

## Analisadores e convenções do Open API

O Open API (também conhecido como Swagger) é uma especificação independente de linguagem para descrever APIs REST. O ecossistema do Open API tem ferramentas que permitem descobrir, testar e produzir o código do cliente usando a especificação. O suporte para gerar e visualizar documentos do Open API no ASP.NET Core MVC é fornecido por meio de projetos controlados pela comunidade, como [NSwag](#) e [Swashbuckle.AspNetCore](#). O ASP.NET Core 2.2 fornece ferramentas e experiências de tempo de execução aprimoradas para a criação de documentos do Open API.

Para obter mais informações, consulte os seguintes recursos:

- [Usar os analisadores da API Web](#)
- [Usar convenções de API Web](#)
- [ASP.NET Core 2.2.0-preview1: analisadores e convenções do Open API](#)

## Suporte de detalhes do problema

O ASP.NET Core 2.1 introduziu o `ProblemDetails`, com base na especificação RFC 7807 para transmitir os detalhes de um erro com uma Resposta HTTP. No 2.2, `ProblemDetails` é a resposta padrão para códigos de erro do cliente em controladores atribuídos com `ApiControllerAttribute`. Um `IActionResult` que retorna um código de status de erro do cliente (4xx) retorna um corpo `ProblemDetails`. O resultado também inclui uma ID de correlação que pode ser usada para correlacionar o erro usando logs de solicitação. Para erros do cliente, `ProducesResponseType` usa como padrão `ProblemDetails` como o tipo de resposta. Isso é documentado na saída do Open API/Swagger gerada com o NSwag ou o Swashbuckle.AspNetCore.

## Roteamento de ponto de extremidade

O ASP.NET Core 2.2 usa um novo sistema de *roteamento de ponto de extremidade* para expedição aprimorada de solicitações. As alterações incluem novos membros da API de geração de link e transformadores de parâmetro de rota.

Para obter mais informações, consulte os seguintes recursos:

- [Roteamento de ponto de extremidade no 2.2](#)
- [Transformadores de parâmetro de rota](#) (confira a seção **Roteamento**)
- [Diferenças entre o roteamento baseado em IRouter e em ponto de extremidade](#)

## Verificações de integridade

Um novo serviço de verificações de integridade facilita o uso do ASP.NET Core em ambientes que exigem verificações de integridade, como o Kubernetes. As verificações de integridade incluem middleware e um conjunto de bibliotecas que definem um serviço e uma abstração `IHealthCheck`.

As verificações de integridade são usadas por um orquestrador de contêineres ou um平衡ador de carga para determinar rapidamente se um sistema está respondendo às solicitações normalmente. Um orquestrador de

contêineres pode responder a uma verificação de integridade com falha interrompendo uma implantação sem interrupção ou reiniciando um contêiner. Um balanceador de carga pode responder a uma verificação de integridade encaminhando o tráfego para fora da instância com falha do serviço.

As verificações de integridade são expostas por um aplicativo como um ponto de extremidade HTTP usado por sistemas de monitoramento. As verificações de integridade podem ser configuradas para uma variedade de cenários de monitoramento em tempo real e sistemas de monitoramento. O serviço de verificações de integridade é integrado ao [projeto BeatPulse](#), que facilita a adição de verificações para dezenas de sistemas e dependências populares.

Para obter mais informações, confira [Verificações de integridade no ASP.NET Core](#).

## HTTP/2 no Kestrel

O ASP.NET Core 2.2 adiciona suporte ao HTTP/2.

O HTTP/2 é uma revisão principal do protocolo HTTP. Alguns dos recursos importantes do HTTP/2 são o suporte à compactação de cabeçalho e fluxos totalmente multiplexados em uma única conexão. Embora o HTTP/2 preserve a semântica do HTTP (cabeçalhos HTTP, métodos etc.), ele é uma alteração da falha do HTTP/1.x com relação a como esses dados são estruturados e enviados pela conexão.

Como consequência dessa alteração no enquadramento, os servidores e os clientes precisam negociar a versão de protocolo usada. O recurso ALPN (Negociação de Protocolo da Camada de Aplicativo) é uma extensão TLS com a qual o servidor e o cliente podem negociar a versão de protocolo usada como parte do handshake TLS. Embora seja possível ter um conhecimento prévio entre o servidor e o cliente sobre o protocolo, todos os principais navegadores dão suporte ALPN como a única maneira de estabelecer uma conexão HTTP/2.

Para obter mais informações, confira [Suporte ao HTTP/2](#).

## Configuração do Kestrel

Em versões anteriores do ASP.NET Core, as opções do Kestrel são configuradas por meio da chamada a `UseKestrel`. No 2.2, as opções do Kestrel são configuradas por meio da chamada a `ConfigureKestrel` no construtor do host. Essa alteração resolve um problema com a ordem dos registros `IIServer` para a hospedagem em processo. Para obter mais informações, consulte os seguintes recursos:

- [Atenuar conflitos do UseIIS](#)
- [Configurar opções do servidor Kestrel com ConfigureKestrel](#)

## Hospedagem em processo do IIS

Em versões anteriores do ASP.NET Core, o IIS funciona como um proxy reverso. No 2.2, o Módulo do ASP.NET Core pode inicializar o CoreCLR e hospedar um aplicativo dentro do processo de trabalho do IIS (`w3wp.exe`). A hospedagem em processo fornece ganhos de desempenho e diagnóstico durante a execução com o IIS.

Para obter mais informações, confira [Hospedagem em processo para IIS](#).

## Cliente Java do SignalR

O ASP.NET Core 2.2 introduz um Cliente Java para o SignalR. Esse cliente dá suporte à conexão a um Servidor do ASP.NET Core SignalR por meio do código Java, incluindo aplicativos Android.

Para obter mais informações, confira [Cliente Java do ASP.NET Core SignalR](#).

## Melhorias do CORS

Em versões anteriores do ASP.NET Core, o Middleware do CORS permite o envio dos cabeçalhos `Accept`, `Accept-Language`, `Content-Language` e `Origin`, independentemente dos valores configurados em `CorsPolicy.Headers`. No 2.2, uma correspondência de política do Middleware do CORS só é possível quando os cabeçalhos enviados em `Access-Control-Request-Headers` corresponder exatamente aos cabeçalhos indicados em `WithHeaders`.

Para obter mais informações, confira [Middleware do CORS](#).

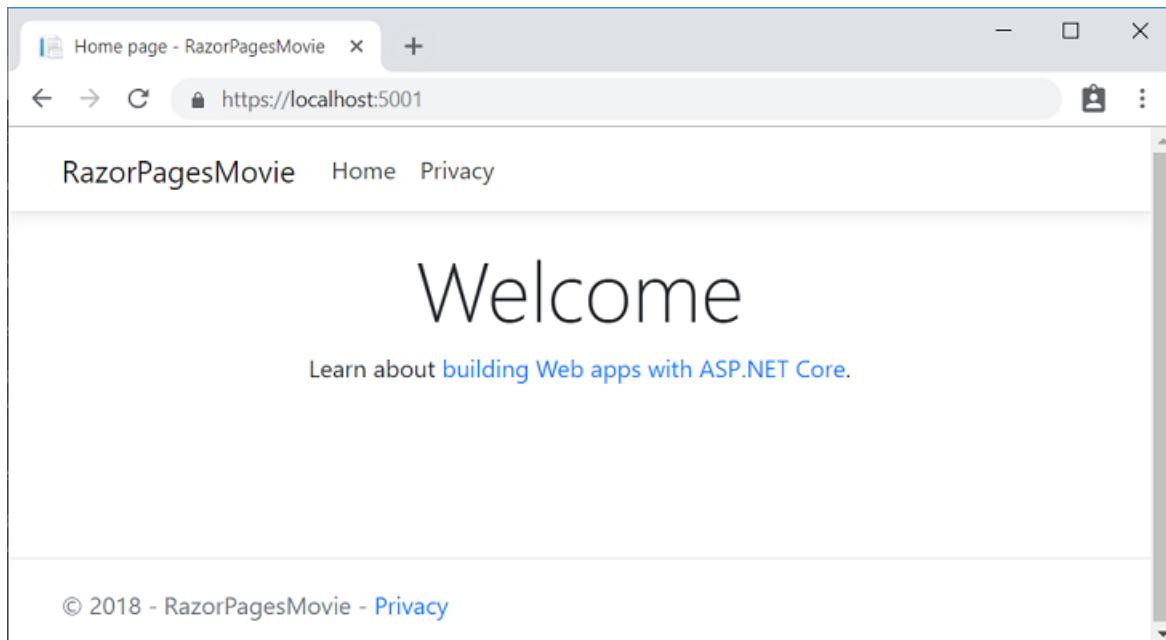
## Compactação de resposta

O ASP.NET Core 2.2 pode compactar respostas com o [formato de compactação Brotli](#).

Para obter mais informações, confira [O middleware de compactação de resposta dá suporte à compactação Brotli](#).

## Modelos de projeto

Os modelos de projeto Web ASP.NET Core foram atualizados para o [Bootstrap 4](#) e o [Angular 6](#). A nova aparência é visualmente mais simples e facilita a visualização das estruturas importantes do aplicativo.



## Desempenho de validação

O sistema de validação do MVC foi projetado para ser extensível e flexível, permitindo que você determine a cada solicitação quais validadores se aplicam a determinado modelo. Isso é ótimo para a criação de provedores de validação complexa. No entanto, na maioria dos casos, um aplicativo usa apenas os validadores internos e não exige essa flexibilidade extra. Validadores internos incluem DataAnnotations como `[Required]` e `[StringLength]`, e `IValidatableObject`.

No ASP.NET Core 2.2, o MVC poderá causar um curto-circuito na validação se ele determinar que um grafo de modelo fornecido não exige validação. Ignorar a validação resulta em melhorias significativas ao validar modelos que não podem ou não têm nenhum validador. Isso inclui objetos, como coleções de primitivos (como `byte[]`, `string[]`, `Dictionary<string, string>`), ou grafos de objeto complexo sem muitos validadores.

## Desempenho do Cliente HTTP

No ASP.NET Core 2.2, o desempenho do `SocketsHttpHandler` foi aprimorado, reduzindo a contenção de bloqueio do pool de conexão. Para aplicativos que fazem muitas solicitações HTTP de saída, como algumas arquiteturas de microserviços, a taxa de transferência foi aprimorada. Sob carga, a taxa de transferência do `HttpClient` pode ser

melhorada em até 60% no Linux e 20% no Windows.

Para obter mais informações, confira [a solicitação de pull que fez essa melhoria](#).

## Informações adicionais

Para obter a lista completa de alterações, confira as [Notas sobre a versão do ASP.NET Core 2.2](#).

# Novidades do ASP.NET Core 2.1

30/10/2018 • 12 minutes to read • [Edit Online](#)

Este artigo destaca as alterações mais significativas no ASP.NET Core 2.1, com links para a documentação relevante.

## SignalR

O SignalR foi reescrito para ASP.NET Core 2.1. O SignalR do ASP.NET Core inclui uma série de melhorias:

- Um modelo de expansão simplificado.
- Um novo cliente JavaScript sem dependência jQuery.
- Um novo protocolo binário compacto com base em MessagePack.
- Suporte para protocolos personalizados.
- Um novo modelo de resposta de transmissão.
- Suporte para clientes com base em WebSockets básicos.

Para obter mais informações, veja [ASP.NET Core SignalR](#).

## Biblioteca de classes Razor

O ASP.NET Core 2.1 torna mais fácil criar e incluir interface do usuário baseada em Razor em uma biblioteca e compartilhá-la em vários projetos. O novo SDK do Razor habilita o build de arquivos do Razor em um projeto de biblioteca de classes que podem ser empacotados em um pacote do NuGet. Exibições e páginas em bibliotecas são descobertas automaticamente e podem ser substituídas pelo aplicativo. Ao integrar a compilação do Razor ao build:

- O tempo de inicialização do aplicativo é significativamente mais rápido.
- Atualizações rápidas a modos de exibição e páginas Razor em tempo de execução ainda estão disponíveis como parte de um fluxo de trabalho de desenvolvimento iterativo.

Para obter mais informações, veja [Criar interface do usuário reutilizável usando o projeto de Biblioteca de Classes do Razor](#).

## Scaffolding e biblioteca de interface do usuário de Identidade

O ASP.NET Core 2.1 fornece a [Identidade do ASP.NET Core](#) como uma [Biblioteca de Classes do Razor](#).

Aplicativos que incluem Identidade podem aplicar o novo scaffolder de Identidade para adicionar seletivamente o código-fonte contido na RCL (Biblioteca de Classes do Razor) de Identidade. Talvez você queira gerar o código-fonte para que você possa modificar o código e alterar o comportamento. Por exemplo, você pode instruir o scaffolder a gerar o código usado no registro. Código gerado tem precedência sobre o mesmo código na RCL de Identidade.

Aplicativos que **não** incluem autenticação podem aplicar o scaffolder de Identidade para adicionar o pacote de Identidade RCL. Você tem a opção de selecionar o código de Identidade a ser gerado.

Para obter mais informações, veja [Identidade scaffold em projetos ASP.NET Core](#).

## HTTPS

Com o foco cada vez maior em segurança e privacidade, é importante habilitar HTTPS para aplicativos Web. A

imposição de HTTPS está se tornando cada vez mais rígida na Web. Sites que não usam HTTPS são considerados inseguros. Navegadores (Chrome, Mozilla) estão começando a impor o uso de recursos Web em um contexto de seguro. O [RGPD](#) requer o uso de HTTPS para proteger a privacidade do usuário. Quando usar HTTPS em produção é fundamental, usar HTTPS no desenvolvimento pode ajudar a evitar problemas de implantação (por exemplo, links inseguros). O ASP.NET Core 2.1 inclui uma série de melhorias que tornam mais fácil usar HTTPS em desenvolvimento e configurar HTTPS em produção. Para obter mais informações, veja [Impor HTTPS](#).

## Ativo por padrão

Para facilitar o desenvolvimento de site seguro, HTTPS está habilitado por padrão. Da versão 2.1 em diante, o Kestrel escuta em `https://localhost:5001` quando um certificado de desenvolvimento local está presente. Um certificado de desenvolvimento é criado:

- Como parte da experiência de primeira execução do SDK do .NET Core, quando você usa o SDK pela primeira vez.
- Manualmente usando a nova ferramenta `dotnet dev-certs`.

Execute `dotnet dev-certs https --trust` para confiar no certificado.

## Redirecionamento e imposição de HTTPS

Aplicativos Web geralmente precisam escutar em HTTP e HTTPS, mas, então redirecionam todo o tráfego HTTP para HTTPS. Na versão 2.1, foi introduzido o middleware de redirecionamento de HTTPS especializado que redireciona de modo inteligente com base na presença de portas do servidor associado ou de configuração.

O uso de HTTPS pode ser imposto ainda mais empregando [HSTS \(Protocolo de Segurança do Transporte Estrita HTTP\)](#). HSTS instrui navegadores a sempre acessarem o site por meio de HTTPS. O ASP.NET Core 2.1 adiciona middleware HSTS compatível com opções para idade máxima, subdomínios e a lista de pré-carregamento de HSTS.

## Configuração para produção

Em produção, HTTPS precisa ser configurado explicitamente. Na versão 2.1, foi adicionado o esquema de configuração padrão para configurar HTTPS para Kestrel. Os aplicativos podem ser configurados para usar:

- Vários pontos de extremidade incluindo as URLs. Para obter mais informações, veja [Implementação do servidor Web Kestrel: configuração de ponto de extremidade](#).
- O certificado a ser usado para HTTPS de um arquivo no disco ou de um repositório de certificados.

## RGPD

O ASP.NET Core fornece APIs e modelos para ajudar a atender alguns dos requisitos do [RGPD \(Regulamento de Proteção de Dados Geral\) da UE](#). Para obter mais informações, veja [Suporte RGPD no ASP.NET Core](#). Um [aplicativo de exemplo](#) mostra como usar e permite que você teste a maioria dos pontos de extensão RGPD e APIs adicionados aos modelos do ASP.NET Core 2.1.

## Testes de integração

É introduzido um novo pacote que simplifica a criação e a execução do teste. O pacote [Microsoft.AspNetCore.Mvc.Testing](#) lida com as seguintes tarefas:

- Copia o arquivo de dependência (\*.deps) do aplicativo testado para a pasta `bin` do projeto de teste.
- Define a raiz de conteúdo para a raiz do projeto do aplicativo testado para que arquivos estáticos e páginas/exibições sejam encontrados quando os testes forem executados.
- Fornece a classe [WebApplicationFactory](#) para simplificar a inicialização do aplicativo testado com [TestServer](#).

O teste a seguir usa [xUnit](#) para verificar se a página de índice é carregada com um código de status de êxito e o cabeçalho Content-Type correto:

```

public class BasicTests
    : IClassFixture<WebApplicationFactory<RazorPagesProject.Startup>>
{
    private readonly HttpClient _client;

    public BasicTests(WebApplicationFactory<RazorPagesProject.Startup> factory)
    {
        _client = factory.CreateClient();
    }

    [Fact]
    public async Task GetHomePage()
    {
        // Act
        var response = await _client.GetAsync("/");

        // Assert
        response.EnsureSuccessStatusCode(); // Status Code 200-299
        Assert.Equal("text/html; charset=utf-8",
            response.Content.Headers.ContentType.ToString());
    }
}

```

Para obter mais informações, veja o tópico [Testes de integração](#).

## [ApiController], ActionResult<T>

O ASP.NET Core 2.1 adiciona novas convenções de programação que facilitam o build de APIs Web limpas e descritivas. `ActionResult<T>` é um novo tipo adicionado para permitir que um aplicativo retorne um tipo de resposta ou qualquer outro resultado da ação (semelhante a `IActionResult`), enquanto ainda indica o tipo de resposta. O atributo `[ApiController]` também foi adicionado como a maneira de aceitar convenções e comportamentos específicos da API Web.

Para obter mais informações, veja [Criar APIs Web com ASP.NET Core](#).

## IHttpClientFactory

O ASP.NET Core 2.1 inclui um novo serviço `IHttpClientFactory` que torna mais fácil configurar e consumir instâncias de `HttpClient` em aplicativos. O `HttpClient` já tem o conceito de delegar manipuladores que podem ser vinculados uns aos outros para solicitações HTTP de saída. O alocador:

- Torna o registro de instâncias de `HttpClient` por cliente nomeado mais intuitivo.
- Implementa um manipulador Polly que permite que políticas Polly sejam usadas para Retry, CircuitBreakers etc.

Para obter mais informações, veja [Iniciar solicitações de HTTP](#).

## Configuração de transporte do Kestrel

Com a liberação do ASP.NET Core 2.1, o transporte padrão do Kestrel deixa de ser baseado no Libuv, baseando-se agora em soquetes gerenciados. Para obter mais informações, veja [Implementação do servidor Web Kestrel: configuração de transporte](#).

## Construtor de host genérico

O Construtor de Host Genérico (`HostBuilder`) foi introduzido. Este construtor pode ser usado para aplicativos que não processam solicitações HTTP (mensagens, tarefas em segundo plano etc.).

Para obter mais informações, veja [Host Genérico do .NET](#).

## Modelos do SPA atualizados

Os modelos de Aplicativo de Página Única para Angular, React e React com Redux são atualizados para usar estruturas de projeto padrão e criar sistemas para cada estrutura.

O modelo Angular se baseia na CLI Angular e os modelos React baseiam-se em create-react-app. Para obter mais informações, veja [Usar os modelos de aplicativo de página única com ASP.NET Core](#).

## Pesquisa de Razor Pages para ativos Razor

Na versão 2.1, as Razor Pages pesquisam ativos do Razor (como layouts e parciais) nos seguintes diretórios na ordem listada:

1. Pasta de Páginas atual.
2. `/Pages/Shared/`
3. `/Views/Shared/`

## Razor Pages em uma área

Razor Pages agora são compatíveis com [áreas](#). Para ver um exemplo de áreas, crie um novo aplicativo Web Razor Pages com contas de usuário individuais. Um aplicativo Web Razor Pages com contas de usuário individuais inclui `/Areas/Identity/Pages`.

## Versão de compatibilidade do MVC

O método [SetCompatibilityVersion](#) permite que um aplicativo aceite ou recuse as possíveis alterações da falha de comportamento introduzidas no ASP.NET Core MVC 2.1 ou posteriores.

Para obter mais informações, consulte [Versão de compatibilidade do ASP.NET Core MVC](#).

## Migrar de 2.0 para 2.1

Veja [Migrar do ASP.NET Core 2.0 para 2.1](#).

## Informações adicionais

Para obter uma lista de alterações, vejas as [Notas de versão do ASP.NET Core 2.1](#).

# Novidades do ASP.NET Core 2.0

30/10/2018 • 12 minutes to read • [Edit Online](#)

Este artigo destaca as alterações mais significativas no ASP.NET Core 2.0, com links para a documentação relevante.

## Páginas do Razor

Páginas do Razor é um novo recurso do ASP.NET Core MVC que torna a codificação de cenários focados em página mais fácil e produtiva.

Para obter mais informações, consulte a introdução e o tutorial:

- [Introdução a Páginas do Razor](#)
- [Introdução a Páginas do Razor](#)

## Metapacote do ASP.NET Core

Um novo metapacote do ASP.NET Core inclui todos os pacotes feitos e com suporte pelas equipes do ASP.NET Core e do Entity Framework Core, juntamente com as respectivas dependências internas e de terceiros. Você não precisa mais escolher recursos individuais do ASP.NET Core por pacote. Todos os recursos estão incluídos no pacote [Microsoft.AspNetCore.All](#). Os modelos padrão usam este pacote.

Para obter mais informações, consulte [Metapacote do Microsoft.AspNetCore.All para ASP.NET Core 2.0](#).

## Repositório de tempo de execução

Aplicativos que usam o metapacote `Microsoft.AspNetCore.All` aproveitam automaticamente o novo repositório de tempo de execução do .NET Core. O repositório contém todos os ativos de tempo de execução necessários para executar aplicativos ASP.NET Core 2.0. Quando você usa o metapacote `Microsoft.AspNetCore.All`, nenhum ativo dos pacotes NuGet do ASP.NET Core referenciados são implantados com o aplicativo porque eles já estão no sistema de destino. Os ativos no repositório de tempo de execução também são pré-compilados para melhorar o tempo de inicialização do aplicativo.

Para obter mais informações, consulte [Repositório de tempo de execução](#)

## .NET Standard 2.0

Os pacotes do ASP.NET Core 2.0 são direcionados ao .NET Standard 2.0. Os pacotes podem ser referenciados por outras bibliotecas do .NET Standard 2.0 e podem ser executados em implementações em conformidade com o .NET Standard 2.0, incluindo o .NET Core 2.0 e o .NET Framework 4.6.1.

O metapacote `Microsoft.AspNetCore.All` aborda apenas o .Net Core 2.0 porque ele foi projetado para ser utilizado com o repositório de tempo de execução do .Net Core 2.0.

## Atualização da configuração

Uma instância de `IConfiguration` é adicionada ao contêiner de serviços por padrão no ASP.NET Core 2.0. `IConfiguration` no contêiner de serviços torna mais fácil para aplicativos recuperarem valores de configuração do contêiner.

Para obter informações sobre o status da documentação planejada, consulte o [problema do GitHub](#).

## Atualização de registro em log

No ASP.NET Core 2.0, o log será incorporado no sistema de DI (injeção de dependência) por padrão. Você adiciona provedores e configura a filtragem no arquivo *Program.cs* em vez de usar o arquivo *Startup.cs*. E o `ILoggerFactory` padrão dá suporte à filtragem de forma que lhe permite usar uma abordagem flexível para filtragem entre provedores e filtragem específica do provedor.

Para obter mais informações, consulte [Introdução ao registro em log](#).

## Atualização de autenticação

Um novo modelo de autenticação torna mais fácil configurar a autenticação para um aplicativo usando a DI.

Novos modelos estão disponíveis para configurar a autenticação para aplicativos Web e APIs Web usando o [Azure AD B2C] (<https://azure.microsoft.com/services/active-directory-b2c/>).

Para obter informações sobre o status da documentação planejada, consulte o [problema do GitHub](#).

## Atualização de identidade

Tornamos mais fácil criar APIs Web seguras usando a identidade do ASP.NET Core 2.0. Você pode adquirir tokens de acesso para acessar suas APIs Web usando a [MSAL \(Biblioteca de Autenticação da Microsoft\)](#).

Para obter mais informações sobre alterações de autenticação no 2.0, consulte os seguintes recursos:

- [Confirmação de conta e de recuperação de senha no ASP.NET Core](#)
- [Habilitar a geração de código QR para aplicativos de autenticador no ASP.NET Core](#)
- [Migrar a autenticação e a identidade para o ASP.NET Core 2.0](#)

## Modelos do SPA

Modelos de projeto de SPA (aplicativo de página única) para Angular, Aurelia, Knockoutjs, React.js e React.js com Redux estão disponíveis. O modelo Angular foi atualizado para Angular 4. Os modelos Angular e React estão disponíveis por padrão. Para saber como obter os outros modelos, confira [Criar um novo projeto de SPA](#). Para obter informações de como criar um SPA no ASP.NET Core, confira [Usar JavaScriptServices para criar aplicativos de página única](#).

## Melhorias do Kestrel

O servidor Web Kestrel tem novos recursos que o tornam mais adequado como um servidor voltado para a Internet. Uma série de opções de configuração de restrição de servidor serão adicionadas na nova propriedade `Limits` da classe `KestrelServerOptions`. Adicione limites para o seguinte:

- Número máximo de conexões de cliente
- Tamanho máximo do corpo da solicitação
- Taxa de dados mínima do corpo da solicitação

Para obter mais informações, consulte [Implementação do servidor Web Kestrel no ASP.NET Core](#).

## WebListener renomeado para HTTP.sys

Os pacotes `Microsoft.AspNetCore.Server.WebListener` e `Microsoft.Net.Http.Server` foram mesclados em um novo pacote `Microsoft.AspNetCore.Server.HttpSys`. Os namespaces foram atualizados para corresponderem.

Para obter mais informações, consulte [Implementação do servidor Web HTTP.sys no ASP.NET Core](#).

## Supporte aprimorado a cabeçalho HTTP

Ao usar o MVC para transmitir um `FileStreamResult` ou um `FileContentResult`, agora você tem a opção de definir uma `ETag` ou uma data `LastModified` no conteúdo que você transmitir. Você pode definir esses valores no conteúdo retornado com código semelhante ao seguinte:

```
var data = Encoding.UTF8.GetBytes("This is a sample text from a binary array");
var entityTag = new EntityTagHeaderValue("\"MyCalculatedEtagValue\"");
return File(data, "text/plain", "downloadName.txt", lastModified: DateTime.UtcNow.AddSeconds(-5), entityTag: entityTag);
```

O arquivo retornado para os visitantes será decorado com os cabeçalhos HTTP apropriados para os valores `ETag` e `LastModified`.

Se um visitante do aplicativo solicitar o conteúdo com um cabeçalho de solicitação de intervalo, o ASP.NET Core reconhecerá a solicitação e lidará com o cabeçalho. Se parte do conteúdo solicitado puder ser entregue, o ASP.NET Core ignorará a parte em questão e retornará apenas o conjunto de bytes solicitado. Você não precisa gravar nenhum manipulador especial em seus métodos para adaptar ou manipular esse recurso; ele é manipulado automaticamente para você.

## Inicialização de hospedagem e o Application Insights

Ambientes de hospedagem podem injetar dependências de pacote extras e executar código durante a inicialização do aplicativo, sem que o aplicativo precise tomar uma dependência explicitamente ou chamar algum método. Esse recurso pode ser usado para habilitar determinados ambientes para recursos de "esclarecimento" exclusivos para esse ambiente sem que o aplicativo precise saber antecipadamente.

No ASP.NET Core 2.0, esse recurso é usado para habilitar o diagnóstico do Application Insights automaticamente durante a depuração no Visual Studio e (depois de optar por isto) quando em execução nos Serviços de Aplicativos do Azure. Como resultado, os modelos de projeto não adicionam mais código e pacotes do Application Insights por padrão.

Para obter informações sobre o status da documentação planejada, consulte o [problema do GitHub](#).

## Uso automático de tokens antifalsificação

O ASP.NET Core sempre ajudou a fazer a codificação HTML de seu conteúdo por padrão, mas com a nova versão, estamos dando um passo adicional para ajudar a impedir ataques de CSRF (falsificação de solicitação entre sites). O ASP.NET Core agora emitirá tokens antifalsificação por padrão e os validará em ações de POST do formulário e em páginas sem configuração adicional.

Para obter mais informações, confira [Impedir ataques de XSRF/CSRF \(solicitação intersite forjada\)](#).

## Pré-compilação automática

A pré-compilação da exibição do Razor é habilitada durante a publicação por padrão, reduzindo o tamanho da saída de publicação e o tempo de inicialização do aplicativo.

Para obter mais informações, confira [Compilação e pré-compilação de exibição Razor no ASP.NET Core](#).

## Supporte ao Razor para C# 7.1

O mecanismo de exibição Razor foi atualizado para funcionar com o novo compilador Roslyn. Isso inclui suporte para recursos do C# 7.1 como expressões padrão, nomes de tupla inferidos e correspondência de padrões com genéricos. Para usar o C# 7.1 em seu projeto, adicione a seguinte propriedade no arquivo de projeto e, em seguida,

recarregue a solução:

```
<LangVersion>latest</LangVersion>
```

Para obter informações sobre o status dos recursos do C# 7.1, consulte [o repositório GitHub do Roslyn](#).

## Outras atualizações de documentação para 2.0

- [Perfis de publicação do Visual Studio para a implantação do aplicativo ASP.NET Core](#)
- [Gerenciamento de chaves](#)
- [Configurar a autenticação do Facebook](#)
- [Configurar a autenticação do Twitter](#)
- [Configurar a autenticação do Google](#)
- [Configurar a autenticação da conta da Microsoft](#)

## Diretrizes de migração

Para obter diretrizes sobre como migrar aplicativos ASP.NET Core 1.x para o ASP.NET Core 2.0, consulte os seguintes recursos:

- [Migrar do ASP.NET Core 1.x para o ASP.NET Core 2.0](#)
- [Migrar a autenticação e a identidade para o ASP.NET Core 2.0](#)

## Informações adicionais

Para obter uma lista de alterações, consulte as [Notas de versão do ASP.NET Core 2.0](#).

Para se conectar ao progresso e aos planos da equipe de desenvolvimento do ASP.NET Core, fique ligado no [ASP.NET Community Standup](#).

# Novidades do ASP.NET Core 1.1

10/01/2019 • 2 minutes to read • [Edit Online](#)

O ASP.NET Core 1.1 inclui os seguintes novos recursos:

- [Middleware de regravação de URL](#)
- [Middleware de Cache de Resposta](#)
- [Componentes de exibição como auxiliares de marcação](#)
- [Middleware como filtros MVC](#)
- [Provedor de TempData baseado em cookie](#)
- [Provedor de logs do Serviço de Aplicativo do Azure](#)
- [Provedor de configuração do Azure Key Vault](#)
- [Repositórios de chaves da Proteção de Dados de armazenamento do Azure e do Redis](#)
- [Servidor WebListener para Windows](#)
- [Suporte a WebSockets](#)

## Escolhendo entre as versões 1.0 e 1.1 do ASP.NET Core

O ASP.NET Core 1.1 tem mais recursos do que o 1.0. Em geral, recomendamos o uso da última versão.

## Informações adicionais

- [Notas de versão do ASP.NET Core 1.1.0](#)
- Para se conectar ao progresso e aos planos da equipe de desenvolvimento do ASP.NET Core, fique ligado no [ASP.NET Community Standup](#).

# Tutorial: Criar uma API Web com o ASP.NET Core MVC

30/01/2019 • 28 minutes to read • [Edit Online](#)

Por [Rick Anderson](#) e [Mike Wasson](#)

Este tutorial ensina os conceitos básicos da criação de uma API Web com o ASP.NET Core.

Neste tutorial, você aprenderá como:

- Criar um projeto de API Web.
- Adicionar uma classe de modelo.
- Criar o contexto de banco de dados.
- Registrar o contexto de banco de dados.
- Adicionar um controlador.
- Adicionar métodos CRUD.
- Configurar o roteamento e caminhos de URL.
- Especificar os valores retornados.
- Chamar a API Web com o Postman.
- Chamar a API Web com o jQuery.

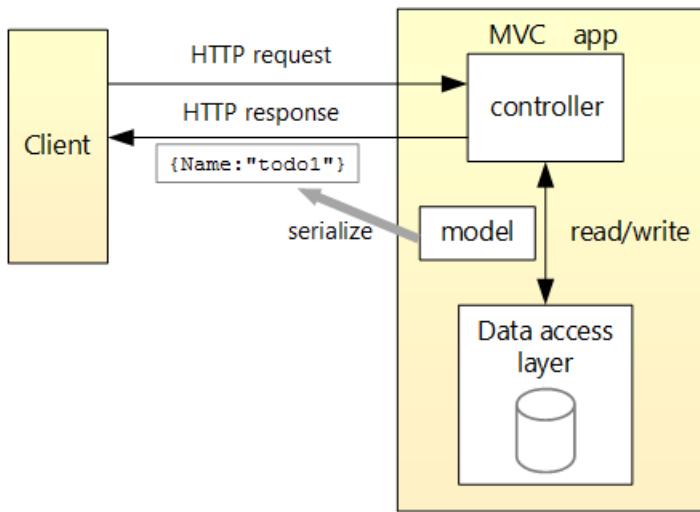
No final, você terá uma API Web que pode gerenciar itens de "tarefas pendentes" armazenados em um banco de dados relacional.

## Visão geral

Este tutorial cria a seguinte API:

API	Descrição	Corpo da solicitação	Corpo da resposta
GET /api/todo	Obter todos os itens de tarefas pendentes	Nenhum	Matriz de itens de tarefas pendentes
GET /api/todo/{id}	Obter um item por ID	Nenhum	Item de tarefas pendentes
POST /api/todo	Adicionar um novo item	Item de tarefas pendentes	Item de tarefas pendentes
PUT /api/todo/{id}	Atualizar um item existente	Item de tarefas pendentes	Nenhum
DELETE /api/todo/{id}	Excluir um item	Nenhum	Nenhum

O diagrama a seguir mostra o design do aplicativo.

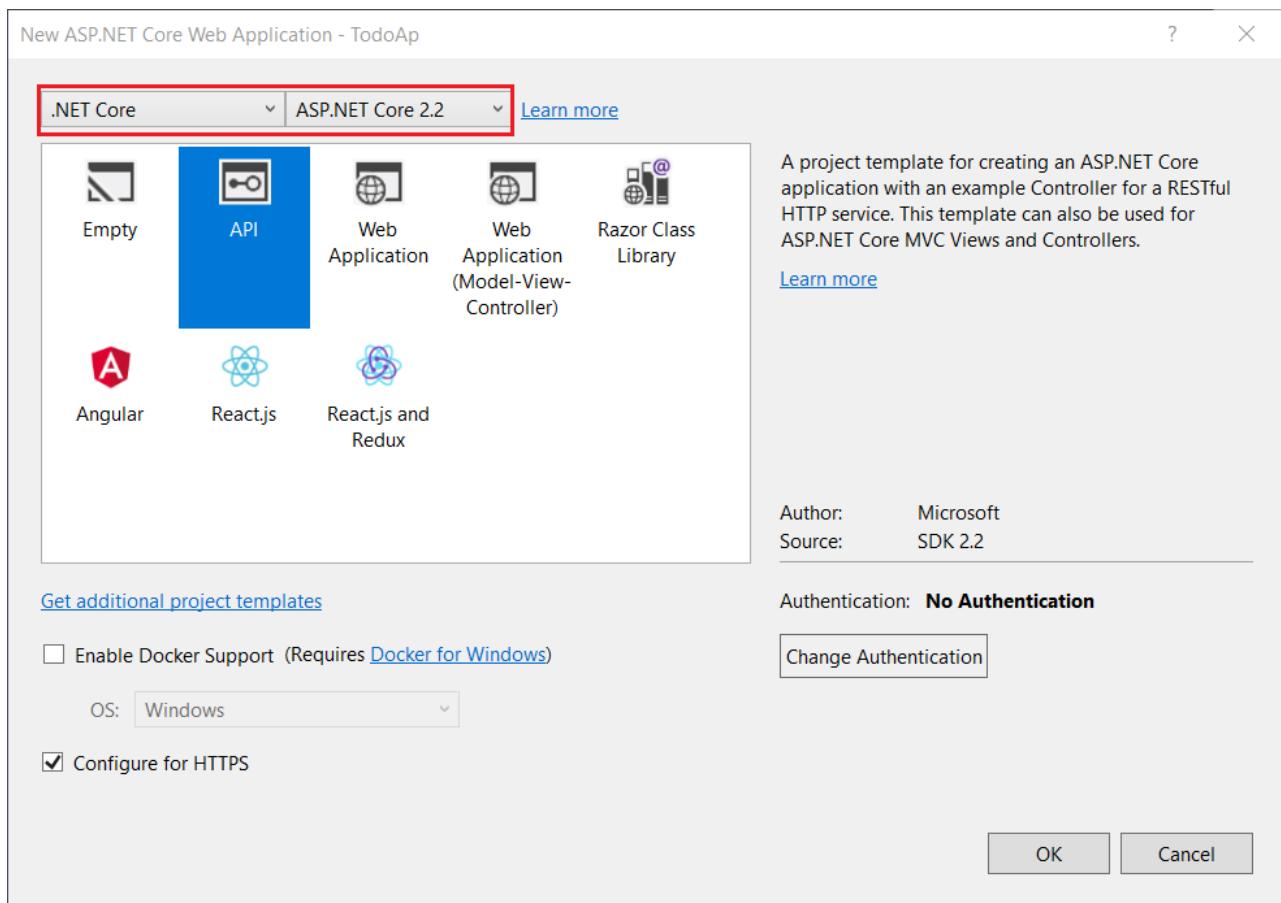


## Pré-requisitos

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- [Visual Studio 2017 versão 15.9 ou posterior](#) com a carga de trabalho **ASP.NET e desenvolvimento para a Web**
- [SDK 2.2 ou posterior do .NET Core](#)

## Criar um projeto Web

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- No menu **Arquivo**, selecione **Novo > Projeto**.
- Selecione o modelo **Aplicativo Web ASP.NET Core**. Nomeie o projeto como *TodoApi* e clique em **OK**.
- Na caixa de diálogo **Novo aplicativo Web ASP.NET Core – TodoApi** e escolha a versão do ASP.NET Core. Selecione o modelo **API** e clique em **OK**. **Não** selecione **Habilitar Suporte ao Docker**.



## Testar a API

O modelo de projeto cria uma API `values`. Chame o método `Get` em um navegador para testar o aplicativo.

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)

Pressione CTRL+F5 para executar o aplicativo. O Visual Studio inicia um navegador e navega para

`https://localhost:<port>/api/values`, em que `<port>` é um número de porta escolhido aleatoriamente.

Se você receber uma caixa de diálogo perguntando se você deve confiar no certificado do IIS Express, selecione **Sim**. Na caixa de diálogo **Aviso de Segurança** exibida em seguida, selecione **Sim**.

O seguinte JSON é retornado:

```
[ "value1", "value2" ]
```

## Adicionar uma classe de modelo

Um *modelo* é um conjunto de classes que representam os dados gerenciados pelo aplicativo. O modelo para esse aplicativo é uma única classe `TodoItem`.

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- No **Gerenciador de Soluções**, clique com o botão direito do mouse no projeto. Selecione **Adicionar > Nova Pasta**. Nomeie a pasta *Models*.
- Clique com o botão direito do mouse na pasta *Modelos* e selecione **Adicionar > Classe**. Dê à classe o nome

*TodoItem* e selecione **Adicionar**.

- Substitua o código do modelo pelo seguinte código:

```
namespace TodoApi.Models
{
    public class TodoItem
    {
        public long Id { get; set; }
        public string Name { get; set; }
        public bool IsComplete { get; set; }
    }
}
```

A propriedade `Id` funciona como a chave exclusiva em um banco de dados relacional.

As classes de modelo podem ser colocadas em qualquer lugar no projeto, mas a pasta *Models* é usada por convenção.

## Adicionar um contexto de banco de dados

O *contexto de banco de dados* é a classe principal que coordena a funcionalidade do Entity Framework para um modelo de dados. Essa classe é criada derivando-a da classe `Microsoft.EntityFrameworkCore.DbContext`.

- [Visual Studio](#)
- [Visual Studio Code/Visual Studio para Mac](#)
- Clique com o botão direito do mouse na pasta *Modelos* e selecione **Adicionar > Classe**. Nomeie a classe como *TodoContext* e clique em **Adicionar**.
- Substitua o código do modelo pelo seguinte código:

```
using Microsoft.EntityFrameworkCore;

namespace TodoApi.Models
{
    public class TodoContext : DbContext
    {
        public TodoContext(DbContextOptions<TodoContext> options)
            : base(options)
        {

        }

        public DbSet<TodoItem> TodoItems { get; set; }
    }
}
```

## Registrar o contexto de banco de dados

No ASP.NET Core, serviços como o contexto de BD precisam ser registrados no contêiner de [DI \(injeção de dependência\)](#). O contêiner fornece o serviço aos controladores.

Atualize *Startup.cs* com o seguinte código realçado:

```

// Unused usings removed
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using TodoApi.Models;

namespace TodoApi
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the
        // container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<TodoContext>(opt =>
                opt.UseInMemoryDatabase("TodoList"));
            services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
        }

        // This method gets called by the runtime. Use this method to configure the HTTP
        // request pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                // The default HSTS value is 30 days. You may want to change this for
                // production scenarios, see https://aka.ms/aspnetcore-hsts.
                app.UseHsts();
            }

            app.UseHttpsRedirection();
            app.UseMvc();
        }
    }
}

```

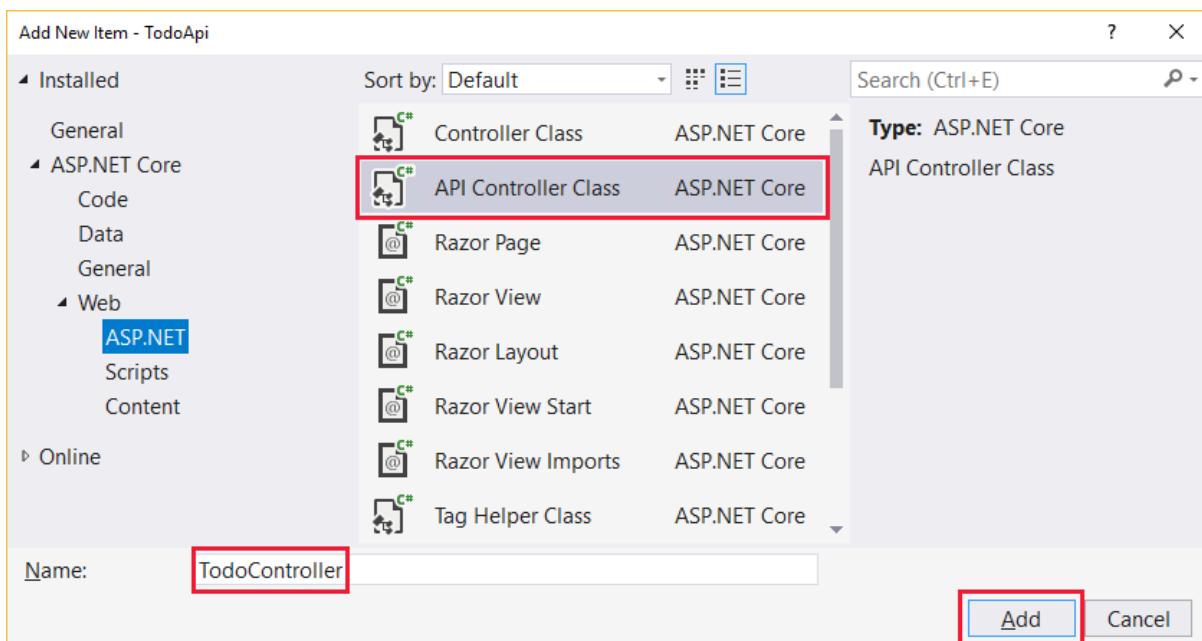
O código anterior:

- Remove as declarações `using` não utilizadas.
- Adiciona o contexto de banco de dados ao contêiner de DI.
- Especifica que o contexto de banco de dados usará um banco de dados em memória.

## Adicionar um controlador

- [Visual Studio](#)
- [Visual Studio Code/Visual Studio para Mac](#)
- Clique com o botão direito do mouse na pasta *Controllers*.
- Selecione **Adicionar > Novo Item**.

- Na caixa de diálogo **Adicionar Novo Item**, selecione o modelo **Classe do Controlador de API**.
- Dê à classe o nome *TodoController* e selecione **Adicionar**.



- Substitua o código do modelo pelo seguinte código:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using TodoApi.Models;

namespace TodoApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class TodoController : ControllerBase
    {
        private readonly TodoContext _context;

        public TodoController(TodoContext context)
        {
            _context = context;

            if (_context.TodoItems.Count() == 0)
            {
                // Create a new TodoItem if collection is empty,
                // which means you can't delete all TodoItems.
                _context.TodoItems.Add(new TodoItem { Name = "Item1" });
                _context.SaveChanges();
            }
        }
    }
}
```

O código anterior:

- Define uma classe de controlador de API sem métodos.
- Decore a classe com o atributo `[ApiController]`. Esse atributo indica se o controlador responde às solicitações da API Web. Para obter informações sobre comportamentos específicos habilitados pelo atributo, confira [Anotação com o atributo ApiController](#).

- Usa a DI para injetar o contexto de banco de dados (`TodoContext`) no controlador. O contexto de banco de dados é usado em cada um dos métodos **CRUD** no controlador.
- Adiciona um item chamado `Item1` ao banco de dados se o banco de dados está vazio. Esse código está no construtor, de modo que ele seja executado sempre que há uma nova solicitação HTTP. Se você excluir todos os itens, o construtor criará `Item1` novamente na próxima vez que um método de API for chamado. Portanto, pode parecer que a exclusão não funcionou quando ela realmente funcionou.

## Adicionar métodos Get

Para fornecer uma API que recupera itens pendentes, adicione os seguintes métodos à classe `TodoController`:

```
// GET: api/Todo
[HttpGet]
public async Task<ActionResult<IEnumerable<TodoItem>>> GetTodoItems()
{
    return await _context.TodoItems.ToListAsync();
}

// GET: api/Todo/5
[HttpGet("{id}")]
public async Task<ActionResult<TodoItem>> GetTodoItem(long id)
{
    var todoItem = await _context.TodoItems.FindAsync(id);

    if (todoItem == null)
    {
        return NotFound();
    }

    return todoItem;
}
```

Esses métodos implementam dois pontos de extremidade GET:

- `GET /api/todo`
- `GET /api/todo/{id}`

Teste o aplicativo chamando os dois pontos de extremidade em um navegador. Por exemplo:

- `https://localhost:<port>/api/todo`
- `https://localhost:<port>/api/todo/1`

A seguinte resposta HTTP é produzida pela chamada a `GetTodoItems`:

```
[
{
    "id": 1,
    "name": "Item1",
    "isComplete": false
}]
```

## Roteamento e caminhos de URL

O atributo `[HttpGet]` indica um método que responde a uma solicitação HTTP GET. O caminho da URL de cada método é construído da seguinte maneira:

- Comece com a cadeia de caracteres de modelo no atributo `Route` do controlador:

```

namespace TodoApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class TodoController : ControllerBase
    {
        private readonly TodoContext _context;

```

- Substitua `[controller]` pelo nome do controlador, que é o nome da classe do controlador menos o sufixo "Controlador" por convenção. Para esta amostra, o nome da classe do controlador é `TodoController` e, portanto, o nome do controlador é "todo". O [roteamento](#) do ASP.NET Core não diferencia maiúsculas de minúsculas.
- Se o atributo `[HttpGet]` tiver um modelo de rota (por exemplo, `[HttpGet("products")]`), acrescente isso ao caminho. Esta amostra não usa um modelo. Para obter mais informações, confira [Roteamento de atributo com atributos `Http\[Verb\]`](#).

No método `GetTodoItem` a seguir, `"{id}"` é uma variável de espaço reservado para o identificador exclusivo do item pendente. Quando `GetTodoItem` é invocado, o valor de `"{id}"` na URL é fornecido para o método no parâmetro `id`.

```

// GET: api/Todo/5
[HttpGet("{id}")]
public async Task<ActionResult<TodoItem>> GetTodoItem(long id)
{
    var todoItem = await _context.TodoItems.FindAsync(id);

    if (todoItem == null)
    {
        return NotFound();
    }

    return todoItem;
}

```

## Valores de retorno

O tipo de retorno dos métodos `GetTodoItems` e `GetTodoItem` é o [tipo `ActionResult<T>`](#). O ASP.NET Core serializa automaticamente o objeto em [JSON](#) e grava o JSON no corpo da mensagem de resposta. O código de resposta para esse tipo de retorno é 200, supondo que não haja nenhuma exceção sem tratamento. As exceções sem tratamento são convertidas em erros 5xx.

Os tipos de retorno `ActionResult` podem representar uma ampla variedade de códigos de status HTTP. Por exemplo, `GetTodoItem` pode retornar dois valores de status diferentes:

- Se nenhum item corresponder à ID solicitada, o método retornará um código de erro 404 `NotFound`.
- Caso contrário, o método retornará 200 com um corpo de resposta JSON. Retornar `item` resulta em uma resposta HTTP 200.

## Testar o método `GetTodosItems`

Este tutorial usa o Postman para testar a API Web.

- Instale o [Postman](#)
- Inicie o aplicativo Web.

- Inicie o Postman.
- Desabilite a **Verificação do certificado SSL**
  - Em **Arquivo > Configurações** (guia \*Geral), desabilite **Verificação do certificado SSL**.

**WARNING**

Habilite novamente a verificação do certificado SSL depois de testar o controlador.

- Crie uma solicitação.
  - Defina o método HTTP como **GET**.
  - Defina a URL de solicitação como `https://localhost:<port>/api/todo`. Por exemplo, `https://localhost:5001/api/todo`.
- Defina **Exibição de dois painéis** no Postman.
- Selecione **Enviar**.

The screenshot shows the Postman application window. At the top, there's a toolbar with 'File', 'Edit', 'View', 'Help', and various icons like 'New', 'Import', 'Runner', 'Invite', and 'Upgrade'. Below the toolbar, the main interface has a header with 'GET GetAll' and a 'No Environment' dropdown. On the left, there's a sidebar with a 'GetAll' section and a 'Params' tab. The main area shows a 'Body' tab with a JSON response:

```

1 [
2   {
3     "id": 1,
4     "name": "Item1",
5     "isComplete": false
6   }
7 ]
  
```

At the bottom right of the interface, there are several buttons: 'Build', 'Browse', a red-highlighted 'Save' button, and other icons.

## Adicionar um método Create

Adicione o seguinte método `PostTodoItem`:

```
// POST: api/Todo
[HttpPost]
public async Task<ActionResult<TodoItem>> PostTodoItem(TodoItem item)
{
    _context.TodoItems.Add(item);
    await _context.SaveChangesAsync();

    return CreatedAtAction(nameof(GetTodoItem), new { id = item.Id }, item);
}
```

O código anterior é um método HTTP POST, conforme indicado pelo atributo `[HttpPost]`. O método obtém o valor do item pendente no corpo da solicitação HTTP.

O método `CreatedAtAction`:

- retorna um código de status HTTP 201 em caso de êxito. HTTP 201 é a resposta padrão para um método HTTP POST que cria um novo recurso no servidor.
- Adiciona um cabeçalho `Location` à resposta. O cabeçalho `Location` especifica o URI do item de tarefas pendentes recém-criado. Para obter mais informações, confira [10.2.2 201 Criado](#).
- Faz referência à ação `GetTodoItem` para criar o URI de `Location` do cabeçalho. A palavra-chave `nameof` do C# é usada para evitar o hard-coding do nome da ação, na chamada `CreatedAtAction`.

```
// GET: api/Todo/
[HttpGet("{id}")]
public async Task<ActionResult<TodoItem>> GetTodoItem(long id)
{
    var todoItem = await _context.TodoItems.FindAsync(id);

    if (todoItem == null)
    {
        return NotFound();
    }

    return todoItem;
}
```

## Testar o método `PostTodoItem`

- Compile o projeto.
- No Postman, defina o método HTTP como `POST`.
- Selecione a guia **Corpo**.
- Selecione o botão de opção **bruto**.
- Defina o tipo como **JSON (aplicativo/json)**.
- No corpo da solicitação, insira JSON para um item pendente:

```
{
    "name": "walk dog",
    "isComplete": true
}
```

- Selecione **Enviar**.

The screenshot shows the Postman application interface. A POST request is being made to `https://localhost:5001/api/todo`. The request body is set to raw JSON, containing:

```
1 {  
2   "name": "walk dog",  
3   "isComplete": true  
4 }
```

The response status is `201 Created`, with a response body showing the created item:

```
1 {  
2   "id": 2,  
3   "name": "walk dog",  
4   "isComplete": true  
5 }
```

Se você receber um erro 405 Método Não Permitido, provavelmente, esse será o resultado da não compilação do projeto após a adição do método `PostTodoItem`.

### Testar o URI do cabeçalho de local

- Selecione a guia **Cabeçalhos** no painel **Resposta**.
- Copie o valor do cabeçalho **Local**:

The screenshot shows the Postman application interface again. This time, the **Headers** tab is selected in the response panel. The **Location** header is highlighted with a red box, showing its value as `https://localhost:5001/api/Todo/2`.

- Defina o método como GET.
- Cole o URI (por exemplo, `https://localhost:5001/api/Todo/2`)
- Selecione **Enviar**.

## Adicionar um método PutTodoItem

Adicione o seguinte método `PutTodoItem`:

```
// PUT: api/Todo/5
[HttpPut("{id}")]
public async Task<IActionResult> PutTodoItem(long id, TodoItem item)
{
    if (id != item.Id)
    {
        return BadRequest();
    }

    _context.Entry(item).State = EntityState.Modified;
    await _context.SaveChangesAsync();

    return NoContent();
}
```

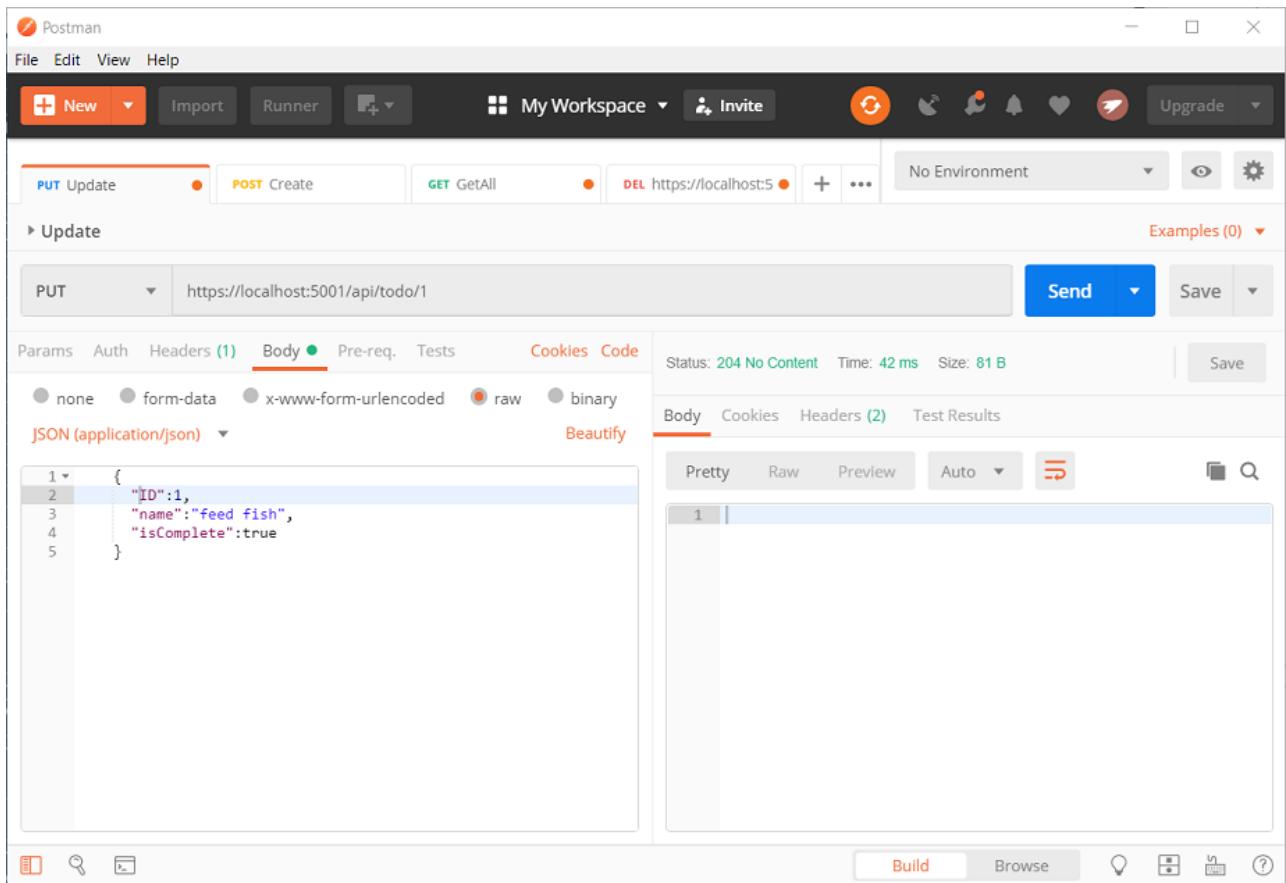
`PutTodoItem` é semelhante a `PostTodoItem`, exceto pelo uso de HTTP PUT. A resposta é [204 \(Sem conteúdo\)](#). De acordo com a especificação de HTTP, uma solicitação PUT exige que o cliente envie a entidade inteira atualizada, não apenas as alterações. Para dar suporte a atualizações parciais, use [HTTP PATCH](#).

### Testar o método PutTodoItem

Atualize o item pendente que tem a ID = 1 e defina seu nome como "feed fish":

```
{
    "ID":1,
    "name":"feed fish",
    "isComplete":true
}
```

A seguinte imagem mostra a atualização do Postman:



## Adicionar um método DeleteTodoItem

Adicione o seguinte método `DeleteTodoItem` :

```
// DELETE: api/Todo/5
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteTodoItem(long id)
{
    var todoItem = await _context.TodoItems.FindAsync(id);

    if (todoItem == null)
    {
        return NotFound();
    }

    _context.TodoItems.Remove(todoItem);
    await _context.SaveChangesAsync();

    return NoContent();
}
```

A resposta `DeleteTodoItem` é **204 (Sem conteúdo)**.

### Testar o método DeleteTodoItem

Use o Postman para excluir um item pendente:

- Defina o método como `DELETE`.
- Defina o URI do objeto a ser excluído, por exemplo, `https://localhost:5001/api/todo/1`
- Selecione **Enviar**

O aplicativo de exemplo permite que você exclua todos os itens, mas quando o último item é excluído, um novo é criado pelo construtor de classe de modelo na próxima vez que a API é chamada.

## Chamar a API com o jQuery

Nesta seção, uma página HTML que usa o jQuery para chamar a API Web é adicionada. O jQuery inicia a solicitação e atualiza a página com os detalhes da resposta da API.

Configure o aplicativo para [fornecer arquivos estáticos](#) e [habilitar o mapeamento de arquivo padrão](#):

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        // The default HSTS value is 30 days. You may want to change this for
        // production scenarios, see https://aka.ms/aspnetcore-hsts.
        app.UseHsts();
    }

    app.UseDefaultFiles();
    app.UseStaticFiles();
    app.UseHttpsRedirection();
    app.UseMvc();
}
```

Crie uma pasta `wwwroot` no diretório do projeto.

Adicione um arquivo HTML chamado `index.html` ao diretório `wwwroot`. Substitua seu conteúdo pela seguinte marcação:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>To-do CRUD</title>
    <style>
        input[type='submit'], button, [aria-label] {
            cursor: pointer;
        }

        #spoiler {
            display: none;
        }

        table {
            font-family: Arial, sans-serif;
            border: 1px solid;
            border-collapse: collapse;
        }

        th {
            background-color: #0066CC;
            color: white;
        }

        td {
            border: 1px solid;
            padding: 5px;
        }
    </style>
</head>
<body>
    <h1>To-do CRUD</h1>
    <h3>Add</h3>
```

```

<form action="javascript:void(0);" method="POST" onsubmit="addItem()">
    <input type="text" id="add-name" placeholder="New to-do">
    <input type="submit" value="Add">
</form>

<div id="spoiler">
    <h3>Edit</h3>
    <form class="my-form">
        <input type="hidden" id="edit-id">
        <input type="checkbox" id="edit-isComplete">
        <input type="text" id="edit-name">
        <input type="submit" value="Save">
        <a onclick="closeInput()" aria-label="Close">&#10006;</a>
    </form>
</div>

<p id="counter"></p>

<table>
    <tr>
        <th>Is Complete</th>
        <th>Name</th>
        <th></th>
        <th></th>
    </tr>
    <tbody id="todos"></tbody>
</table>

<script src="https://code.jquery.com/jquery-3.3.1.min.js"
        integrity="sha256-FgpCb/KJQ1LNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="
        crossorigin="anonymous"></script>
<script src="site.js"></script>
</body>
</html>

```

Adicione um arquivo JavaScript chamado *site.js* ao diretório *wwwroot*. Substitua seu conteúdo pelo código a seguir:

```

const uri = "api/todo";
let todos = null;
function getCount(data) {
    const el = $("#counter");
    let name = "to-do";
    if (data) {
        if (data > 1) {
            name = "to-dos";
        }
        el.text(data + " " + name);
    } else {
        el.text("No " + name);
    }
}

$(document).ready(function() {
    getData();
});

function getData() {
    $.ajax({
        type: "GET",
        url: uri,
        cache: false,
        success: function(data) {
            const tBody = $("#todos");
            $(tBody).empty();
            let count = 0;
            data.forEach(item => {
                const tr = $(`<tr><td>${item.isComplete}</td><td>${item.name}</td><td></td><td></td></tr>`);
                if (item.isComplete) {
                    tr.addClass("completed");
                }
                tBody.append(tr);
                count++;
            });
            getCount(count);
        }
    });
}

```

```

        getCount(data.length);

        $.each(data, function(key, item) {
            const tr = $("<tr></tr>")
                .append(
                    $("<td></td>").append(
                        $("<input/>", {
                            type: "checkbox",
                            disabled: true,
                            checked: item.isComplete
                        })
                    )
                )
                .append($("<td></td>").text(item.name))
                .append(
                    $("<td></td>").append(
                        "<button>Edit</button>").on("click", function() {
                            editItem(item.id);
                        })
                )
                .append(
                    $("<td></td>").append(
                        "<button>Delete</button>").on("click", function() {
                            deleteItem(item.id);
                        })
                )
            );
        });

        tr.appendTo(tBody);
    });

    todos = data;
}
});
}

function addItem() {
    const item = {
        name: $("#add-name").val(),
        isComplete: false
    };

    $.ajax({
        type: "POST",
        accepts: "application/json",
        url: uri,
        contentType: "application/json",
        data: JSON.stringify(item),
        error: function(jqXHR, textStatus, errorThrown) {
            alert("Something went wrong!");
        },
        success: function(result) {
            getData();
            $("#add-name").val("");
        }
    });
}

function deleteItem(id) {
    $.ajax({
        url: uri + "/" + id,
        type: "DELETE",
        success: function(result) {
            getData();
        }
    });
}

```

```

function editItem(id) {
    $.each(todos, function(key, item) {
        if (item.id === id) {
            $("#edit-name").val(item.name);
            $("#edit-id").val(item.id);
            $("#edit-isComplete")[0].checked = item.isComplete;
        }
    });
    $("#spoiler").css({ display: "block" });
}

$(".my-form").on("submit", function() {
    const item = {
        name: $("#edit-name").val(),
        isComplete: $("#edit-isComplete").is(":checked"),
        id: $("#edit-id").val()
    };

    $.ajax({
        url: uri + "/" + $("#edit-id").val(),
        type: "PUT",
        accepts: "application/json",
        contentType: "application/json",
        data: JSON.stringify(item),
        success: function(result) {
            getData();
        }
    });

    closeInput();
    return false;
});

function closeInput() {
    $("#spoiler").css({ display: "none" });
}

```

Uma alteração nas configurações de inicialização do projeto ASP.NET Core pode ser necessária para testar a página HTML localmente:

- Abra `Properties\launchSettings.json`.
- Remova a propriedade `launchUrl` para forçar o aplicativo a ser aberto em `index.html`, o arquivo padrão do projeto.

Há várias maneiras de obter o jQuery. No snippet anterior, a biblioteca é carregada de uma CDN.

Esta amostra chama todos os métodos CRUD da API. Veja a seguir explicações das chamadas à API.

### Obter uma lista de itens pendentes

A função `ajax` do jQuery envia uma solicitação `GET` para a API, que retorna o JSON que representa uma matriz de itens pendentes. A função de retorno de chamada `success` será invocada se a solicitação for bem-sucedida. No retorno de chamada, o DOM é atualizado com as informações do item pendente.

```

$(document).ready(function() {
    getData();
});

function getData() {
    $.ajax({
        type: "GET",
        url: uri,
        cache: false,
        success: function(data) {
            const tBody = $("#todos");

            $(tBody).empty();

            getCount(data.length);

            $.each(data, function(key, item) {
                const tr = $("<tr></tr>")
                    .append(
                        $("<td></td>").append(
                            "<input/>", {
                                type: "checkbox",
                                disabled: true,
                                checked: item.isComplete
                            })
                    )
                )
                .append($("<td></td>").text(item.name))
                .append(
                    $("<td></td>").append(
                        "<button>Edit</button>").on("click", function() {
                            editItem(item.id);
                        })
                )
            )
            .append(
                $("<td></td>").append(
                    "<button>Delete</button>").on("click", function() {
                        deleteItem(item.id);
                    })
            );
        });

        tr.appendTo(tBody);
    });
}

todos = data;
}
});
}

```

## Adicionar um item pendente

A função `ajax` envia uma solicitação `POST` com o item pendente no corpo da solicitação. As opções `accepts` e `contentType` são definidas como `application/json` para especificar o tipo de mídia que está sendo recebido e enviado. O item pendente é convertido em JSON usando `JSON.stringify`. Quando a API retorna um código de status de êxito, a função `getData` é invocada para atualizar a tabela HTML.

```

function addItem() {
    const item = {
        name: $("#add-name").val(),
        isComplete: false
    };

    $.ajax({
        type: "POST",
        accepts: "application/json",
        url: uri,
        contentType: "application/json",
        data: JSON.stringify(item),
        error: function(jqXHR, textStatus, errorThrown) {
            alert("Something went wrong!");
        },
        success: function(result) {
            getData();
            $("#add-name").val("");
        }
    });
}

```

## Atualizar um item pendente

A atualização de um item pendente é semelhante à adição de um. A `url` é alterada para adicionar o identificador exclusivo do item, e o `type` é `PUT`.

```

$.ajax({
    url: uri + "/" + $("#edit-id").val(),
    type: "PUT",
    accepts: "application/json",
    contentType: "application/json",
    data: JSON.stringify(item),
    success: function(result) {
        getData();
    }
});

```

## Excluir um item pendente

A exclusão de um item pendente é feita definindo o `type` na chamada do AJAX como `DELETE` e especificando o identificador exclusivo do item na URL.

```

$.ajax({
    url: uri + "/" + id,
    type: "DELETE",
    success: function(result) {
        getData();
    }
});

```

## Recursos adicionais

[Exibir ou baixar o código de exemplo para este tutorial](#). Consulte [como baixar](#).

Para obter mais informações, consulte os seguintes recursos:

- [Criar APIs Web com o ASP.NET Core](#)
- [Páginas de ajuda da API Web ASP.NET Core com o Swagger/OpenAPI](#)
- [Páginas Razor do ASP.NET Core com EF Core – série de tutoriais](#)

- Roteamento para ações do controlador no ASP.NET Core
- Tipos de retorno de ação do controlador na API Web ASP.NET Core
- Implantar aplicativos ASP.NET Core no Serviço de Aplicativo do Azure
- Hospedar e implantar o ASP.NET Core

## Próximas etapas

Neste tutorial, você aprendeu como:

- Criar um projeto de API Web.
- Adicionar uma classe de modelo.
- Criar o contexto de banco de dados.
- Registrar o contexto de banco de dados.
- Adicionar um controlador.
- Adicionar métodos CRUD.
- Configurar o roteamento e caminhos de URL.
- Especificar os valores retornados.
- Chamar a API Web com o Postman.
- Chamar a API Web com o jQuery.

Avance para o próximo tutorial para saber como gerar páginas de ajuda da API:

[Introdução ao Swashbuckle e ao ASP.NET Core](#)

# Criar uma API Web com o ASP.NET Core e o MongoDB

06/02/2019 • 13 minutes to read • [Edit Online](#)

Por [Pratik Khandelwal](#) e [Scott Addie](#)

Este tutorial cria uma API Web que executa as operações CRUD (criar, ler, atualizar e excluir) em um banco de dados NoSQL do [MongoDB](#).

Neste tutorial, você aprenderá como:

- Configurar o MongoDB
- Criar um banco de dados do MongoDB
- Definir uma coleção e um esquema do MongoDB
- Executar operações CRUD do MongoDB a partir de uma API Web

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Pré-requisitos

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- [SDK 2.2 ou posterior do .NET Core](#)
- [Visual Studio 2017 versão 15.9 ou posterior](#) com a carga de trabalho **ASP.NET e desenvolvimento para a Web**
- [MongoDB](#)

## Configurar o MongoDB

Se usar o Windows, o MongoDB será instalado em *C:\Arquivos de Programas\MongoDB* por padrão. Adicione *C:\Arquivos de Programas\MongoDB\Servidor\<número\_de\_versão>\bin* à variável de ambiente `Path`. Essa alteração possibilita o acesso ao MongoDB a partir de qualquer lugar em seu computador de desenvolvimento.

Use o Shell do mongo nas etapas a seguir para criar um banco de dados, fazer coleções e armazenar documentos. Para saber mais sobre os comandos de Shell do mongo, consulte [Como trabalhar com o Shell do mongo](#).

1. Escolha um diretório no seu computador de desenvolvimento para armazenar os dados. Por exemplo, *C:\BooksData* no Windows. Crie o diretório se não houver um. O Shell do mongo não cria novos diretórios.
2. Abra um shell de comando. Execute o comando a seguir para se conectar ao MongoDB na porta padrão 27017. Lembre-se de substituir `<data_directory_path>` pelo diretório escolhido na etapa anterior.

```
mongod --dbpath <data_directory_path>
```

3. Abra outra instância do shell de comando. Conecte-se ao banco de dados de testes padrão executando o seguinte comando:

```
mongo
```

4. Execute o seguinte em um shell de comando:

```
use BookstoreDb
```

Se ele ainda não existir, um banco de dados chamado *BookstoreDb* será criado. Se o banco de dados existir, a conexão dele será aberta para transações.

5. Crie uma coleção `Books` usando o seguinte comando:

```
db.createCollection('Books')
```

O seguinte resultado é exibido:

```
{ "ok" : 1 }
```

6. Defina um esquema para a coleção `Books` e insira dois documentos usando o seguinte comando:

```
db.Books.insertMany([{"Name": "Design Patterns", "Price": 54.93, "Category": "Computers", "Author": "Ralph Johnson"}, {"Name": "Clean Code", "Price": 43.15, "Category": "Computers", "Author": "Robert C. Martin"}])
```

O seguinte resultado é exibido:

```
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("5bfd996f7b8e48dc15ff215d"),
    ObjectId("5bfd996f7b8e48dc15ff215e")
  ]
}
```

7. Visualize os documentos no banco de dados usando o seguinte comando:

```
db.Books.find({}).pretty()
```

O seguinte resultado é exibido:

```
{
  "_id" : ObjectId("5bfd996f7b8e48dc15ff215d"),
  "Name" : "Design Patterns",
  "Price" : 54.93,
  "Category" : "Computers",
  "Author" : "Ralph Johnson"
}
{
  "_id" : ObjectId("5bfd996f7b8e48dc15ff215e"),
  "Name" : "Clean Code",
  "Price" : 43.15,
  "Category" : "Computers",
  "Author" : "Robert C. Martin"
}
```

O esquema adiciona uma propriedade `_id` gerada automaticamente do tipo `ObjectId` para cada documento.

O banco de dados está pronto. Você pode começar a criar a API Web do ASP.NET Core.

## Criar o projeto da API Web do ASP.NET Core

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)

1. Acesse **Arquivo > Novo > Projeto**.
2. Selecione **Aplicativo Web do ASP.NET Core**, nomeie o projeto como *BooksApi* e clique em **OK**.
3. Selecione a estrutura de destino **.NET Core** e **ASP.NET Core 2.1**. Selecione o modelo de projeto **API** e clique em **OK**:
4. Visite a [Galeria do NuGet: MongoDB.Driver](#) para determinar a versão estável mais recente do driver .NET para MongoDB. Na janela **Console do Gerenciador de Pacotes**, navegue até a raiz do projeto. Execute o seguinte comando para instalar o driver .NET para MongoDB:

```
Install-Package MongoDB.Driver -Version {VERSION}
```

## Adicionar um modelo

1. Adicione um diretório *Modelos* à raiz do projeto.
2. Adicione uma classe `Book` ao diretório *Modelos* com o seguinte código:

```
using MongoDB.Bson;
using MongoDB.Bson.Serialization.Attributes;

namespace BooksApi.Models
{
    public class Book
    {
        [BsonId]
        [BsonRepresentation(BsonType.ObjectId)]
        public string Id { get; set; }

        [BsonElement("Name")]
        public string BookName { get; set; }

        [BsonElement("Price")]
        public decimal Price { get; set; }

        [BsonElement("Category")]
        public string Category { get; set; }

        [BsonElement("Author")]
        public string Author { get; set; }
    }
}
```

Na classe anterior, a propriedade `Id`:

- É necessária para mapear o objeto CLR (Common Language Runtime) para a coleção do MongoDB.
- É anotada com `[BsonId]` para designar essa propriedade como a chave primária do documento.

- É anotada com `[BsonRepresentation(BsonType.ObjectId)]` para permitir a passagem do parâmetro como tipo `string`, em vez de `ObjectId`. O Mongo processa a conversão de `string` para `ObjectId`.

Outras propriedades na classe são anotadas com o atributo `[BsonElement]`. O valor do atributo representa o nome da propriedade da coleção do MongoDB.

## Adicionar uma classe de operações CRUD

1. Adicione um diretório `Serviços` à raiz do projeto.
2. Adicione uma classe `BookService` ao diretório `Serviços` com o seguinte código:

```
using System.Collections.Generic;
using System.Linq;
using BooksApi.Models;
using Microsoft.Extensions.Configuration;
using MongoDB.Driver;

namespace BooksApi.Services
{
    public class BookService
    {
        private readonly IMongoCollection<Book> _books;

        public BookService(IConfiguration config)
        {
            var client = new MongoClient(config.GetConnectionString("BookstoreDb"));
            var database = client.GetDatabase("BookstoreDb");
            _books = database.GetCollection<Book>("Books");
        }

        public List<Book> Get()
        {
            return _books.Find(book => true).ToList();
        }

        public Book Get(string id)
        {
            return _books.Find<Book>(book => book.Id == id).FirstOrDefault();
        }

        public Book Create(Book book)
        {
            _books.InsertOne(book);
            return book;
        }

        public void Update(string id, Book bookIn)
        {
            _books.ReplaceOne(book => book.Id == id, bookIn);
        }

        public void Remove(Book bookIn)
        {
            _books.DeleteOne(book => book.Id == bookIn.Id);
        }

        public void Remove(string id)
        {
            _books.DeleteOne(book => book.Id == id);
        }
    }
}
```

3. Adicione uma cadeia de conexão do MongoDB a `appsettings.json`:

```
{  
    "ConnectionStrings": {  
        "BookstoreDb": "mongodb://localhost:27017"  
    },  
    "Logging": {  
        "IncludeScopes": false,  
        "Debug": {  
            "LogLevel": {  
                "Default": "Warning"  
            }  
        },  
        "Console": {  
            "LogLevel": {  
                "Default": "Warning"  
            }  
        }  
    }  
}
```

A propriedade `BookstoreDb` anterior é acessada no construtor da classe `BookService`.

4. No `Startup.ConfigureServices`, registre a classe `BookService` com o sistema de Injeção de Dependência:

```
public void ConfigureServices(IServiceCollection services)  
{  
    services.AddScoped<BookService>();  
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);  
}
```

O registro de serviço anterior é necessário para dar suporte à injeção do construtor no consumo de classes.

A classe `BookService` usa os seguintes membros `MongoDB.Driver` para executar operações CRUD em relação ao banco de dados:

- `MongoClient` – Lê a instância do servidor para executar operações de banco de dados. O construtor dessa classe é fornecido na cadeia de conexão do MongoDB:

```
public BookService(IConfiguration config)  
{  
    var client = new MongoClient(config.GetConnectionString("BookstoreDb"));  
    var database = client.GetDatabase("BookstoreDb");  
    _books = database.GetCollection<Book>("Books");  
}
```

- `IMongoDatabase` – Representa o banco de dados Mongo para execução de operações. Este tutorial usa o método genérico `GetCollection<T>(collection)` na interface para obter acesso a dados em uma coleção específica. Operações CRUD podem ser executadas em relação à coleção depois que esse método é chamado. Na chamada de método `GetCollection<T>(collection)` :

- `collection` representa o nome da coleção.
- `T` representa o tipo de objeto CLR armazenado na coleção.

`GetCollection<T>(collection)` retorna um objeto `MongoCollection` que representa a coleção. Neste tutorial, os seguintes métodos são invocados na coleção:

- `Find<T>` – Retorna todos os documentos na coleção que correspondem aos critérios de pesquisa fornecidos.
- `InsertOne` – Insere o objeto fornecido como um novo documento na coleção.

- `ReplaceOne` – Substitui o documento único que corresponde aos critérios de pesquisa definidos com o objeto fornecido.
- `DeleteOne` – Exclui um documento único que corresponde aos critérios de pesquisa fornecidos.

## Adicionar um controlador

1. Adicione uma classe `BooksController` ao diretório `Controladores` com o seguinte código:

```
using System.Collections.Generic;
using BooksApi.Models;
using BooksApi.Services;
using Microsoft.AspNetCore.Mvc;

namespace BooksApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class BooksController : ControllerBase
    {
        private readonly BookService _bookService;

        public BooksController(BookService bookService)
        {
            _bookService = bookService;
        }

        [HttpGet]
        public ActionResult<List<Book>> Get()
        {
            return _bookService.Get();
        }

        [HttpGet("{id:length(24)}", Name = "GetBook")]
        public ActionResult<Book> Get(string id)
        {
            var book = _bookService.Get(id);

            if (book == null)
            {
                return NotFound();
            }

            return book;
        }

        [HttpPost]
        public ActionResult<Book> Create(Book book)
        {
            _bookService.Create(book);

            return CreatedAtRoute("GetBook", new { id = book.Id.ToString() }, book);
        }

        [HttpPut("{id:length(24)}")]
        public IActionResult Update(string id, Book bookIn)
        {
            var book = _bookService.Get(id);

            if (book == null)
            {
                return NotFound();
            }

            _bookService.Update(id, bookIn);

            return NoContent();
        }
    }
}
```

```

        }

        [HttpDelete("{id:length(24)}")]
        public IActionResult Delete(string id)
        {
            var book = _bookService.Get(id);

            if (book == null)
            {
                return NotFound();
            }

            _bookService.Remove(book.Id);

            return NoContent();
        }
    }
}

```

O controlador da API Web anterior:

- Usa a classe `BookService` para executar operações CRUD.
- Contém métodos de ação para dar suporte a solicitações GET, POST, PUT e DELETE HTTP.

2. Compile e execute o aplicativo.

3. Navegue até `http://localhost:<port>/api/books` no seu navegador. A seguinte resposta JSON é exibida:

```
[
  {
    "id": "5bfd996f7b8e48dc15ff215d",
    "bookName": "Design Patterns",
    "price": 54.93,
    "category": "Computers",
    "author": "Ralph Johnson"
  },
  {
    "id": "5bfd996f7b8e48dc15ff215e",
    "bookName": "Clean Code",
    "price": 43.15,
    "category": "Computers",
    "author": "Robert C. Martin"
  }
]
```

## Próximas etapas

Para saber mais sobre a criação de APIs Web do ASP.NET Core, confira os seguintes recursos:

- [Criar APIs Web com o ASP.NET Core](#)
- [Tipos de retorno de ação do controlador na API Web ASP.NET Core](#)

# Criar serviços de back-end para aplicativos móveis nativos com o ASP.NET Core

01/10/2018 • 14 minutes to read • [Edit Online](#)

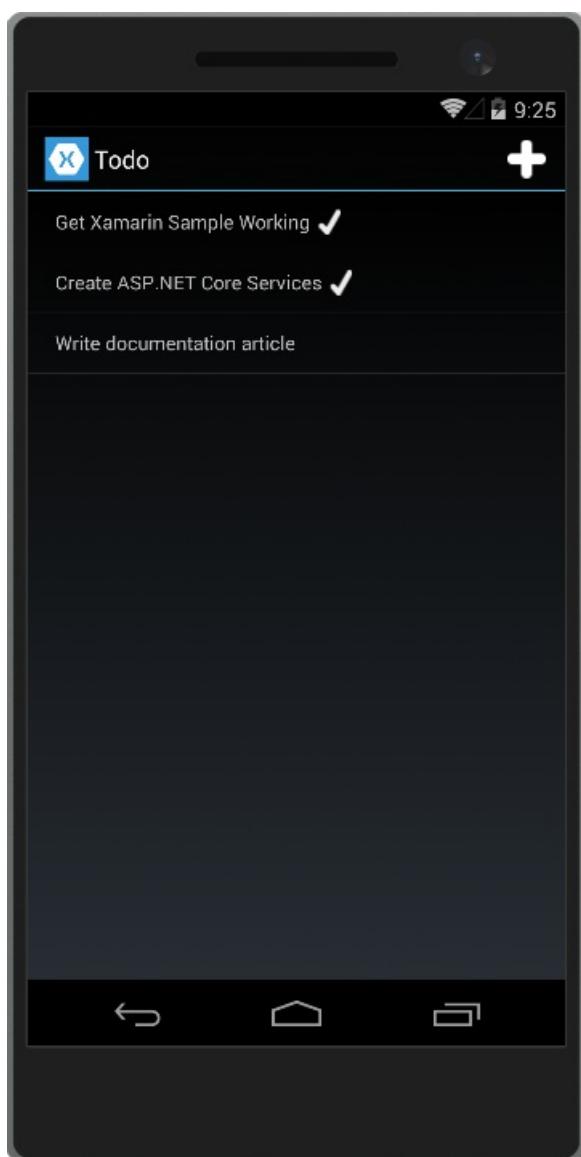
Por [Steve Smith](#)

Os aplicativos móveis podem se comunicar com facilidade com os serviços de back-end do ASP.NET Core.

[Exibir ou baixar o código de exemplo dos serviços de back-end](#)

## Exemplo do aplicativo móvel nativo

Este tutorial demonstra como criar serviços de back-end usando o ASP.NET Core MVC para dar suporte a aplicativos móveis nativos. Ele usa o [aplicativo Xamarin Forms ToDoRest](#) como seu cliente nativo, que inclui clientes nativos separados para dispositivos Android, iOS, Universal do Windows e Windows Phone. Siga o tutorial com links para criar o aplicativo nativo (e instale as ferramentas do Xamarin gratuitas necessárias), além de baixar a solução de exemplo do Xamarin. A amostra do Xamarin inclui um projeto de serviços do ASP.NET Web API 2, que substitui o aplicativo ASP.NET Core deste artigo (sem nenhuma alteração exigida pelo cliente).

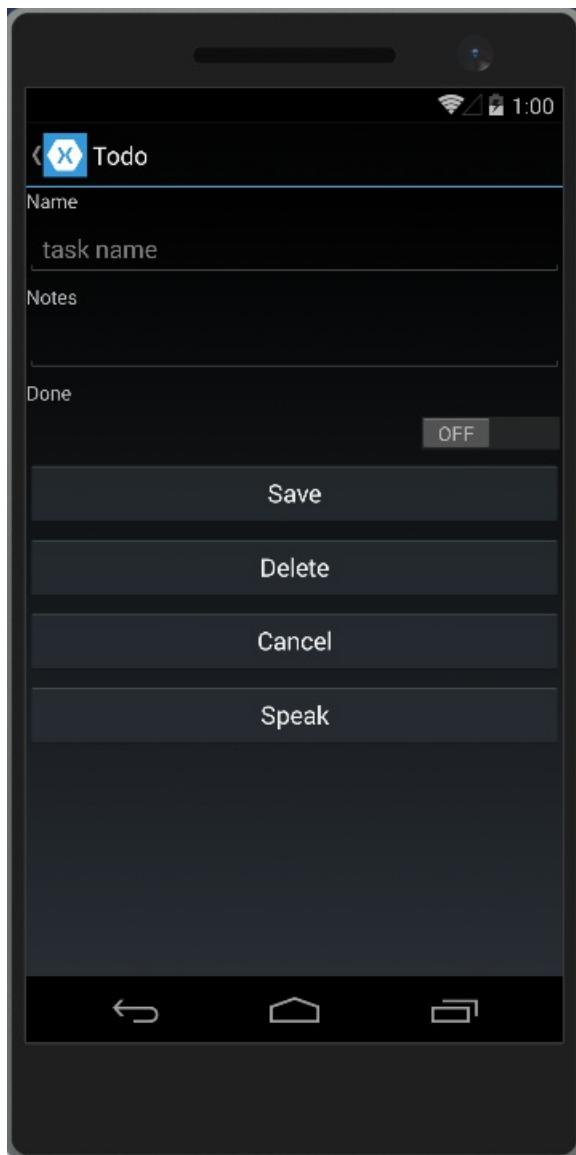


**Recursos**

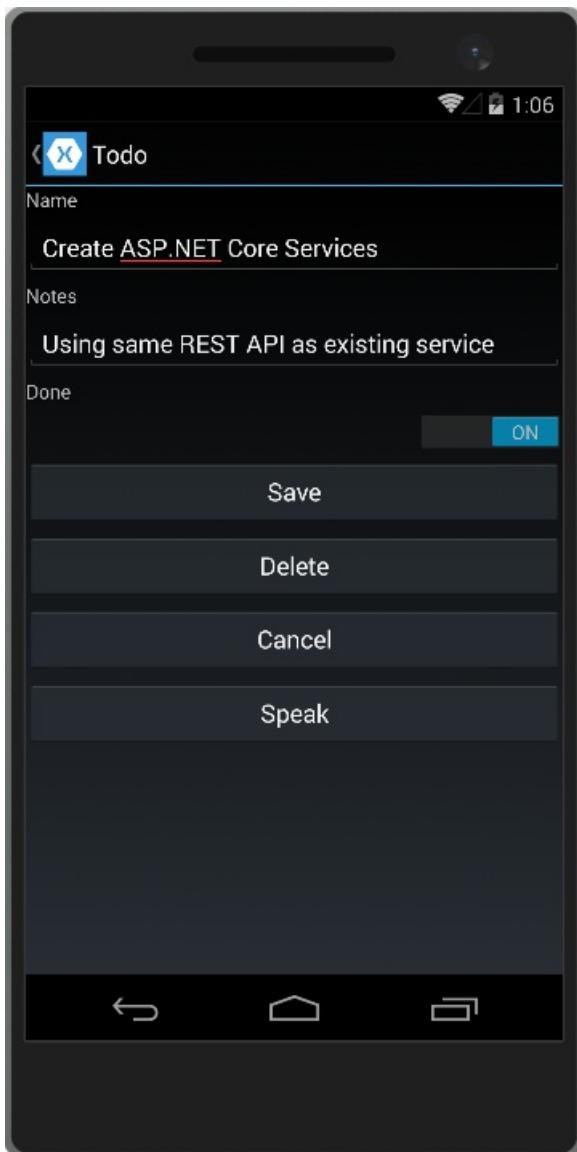
O aplicativo ToDoRest é compatível com listagem, adição, exclusão e atualização de itens de tarefas pendentes. Cada item tem uma ID, um Nome, Observações e uma propriedade que indica se ele já foi Concluído.

A exibição principal dos itens, conforme mostrado acima, lista o nome de cada item e indica se ele foi concluído com uma marca de seleção.

Tocar no ícone  abre uma caixa de diálogo de adição de itens:



Tocar em um item na tela da lista principal abre uma caixa de diálogo de edição, na qual o Nome do item, Observações e configurações de Concluído podem ser modificados, ou o item pode ser excluído:



Esta amostra é configurada por padrão para usar os serviços de back-end hospedados em developer.xamarin.com, que permitem operações somente leitura. Para testá-la por conta própria no aplicativo ASP.NET Core criado na próxima seção em execução no computador, você precisará atualizar a constante `RestUrl` do aplicativo. Navegue para o projeto `ToDoREST` e abra o arquivo `Constants.cs`. Substitua o `RestUrl` por uma URL que inclui o endereço IP do computador (não localhost ou 127.0.0.1, pois esse endereço é usado no emulador do dispositivo, não no computador). Inclua o número da porta também (5000). Para testar se os serviços funcionam com um dispositivo, verifique se você não tem um firewall ativo bloqueando o acesso a essa porta.

```
// URL of REST service (Xamarin ReadOnly Service)
//public static string RestUrl = "http://developer.xamarin.com:8081/api/todoitems{0}";

// use your machine's IP address
public static string RestUrl = "http://192.168.1.207:5000/api/todoitems{0}";
```

## Criando o projeto ASP.NET Core

Crie um novo aplicativo Web do ASP.NET Core no Visual Studio. Escolha o modelo de Web API sem autenticação. Nomeie o projeto como `ToDoApi`.

Select a template:

**ASP.NET Core Templates**

Empty

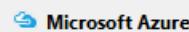


Web API



Web Application

A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET MVC Views and Controllers.

[Learn more](#)[Change Authentication](#)Authentication: **No Authentication** Host in the cloud

App Service

[OK](#)[Cancel](#)

O aplicativo deve responder a todas as solicitações feitas através da porta 5000. Atualize o *Program.cs* para incluir

```
.UseUrls("http://*:5000")
```

 para ficar assim:

```
var host = newWebHostBuilder()
    .UseKestrel()
    .UseUrls("http://*:5000")
    .UseContentRoot(Directory.GetCurrentDirectory())
    .UseIISIntegration()
    .UseStartup<Startup>()
    .Build();
```

**NOTE**

Execute o aplicativo diretamente, em vez de por trás do IIS Express, que ignora solicitações não local por padrão. Execute `dotnet run` em um prompt de comando ou escolha o perfil de nome do aplicativo no menu suspenso Destino de Depuração na barra de ferramentas do Visual Studio.

Adicione uma classe de modelo para representar itens pendentes. Marque os campos obrigatórios usando o atributo `[Required]`:

```

using System.ComponentModel.DataAnnotations;

namespace ToDoApi.Models
{
    public class ToDoItem
    {
        [Required]
        public string ID { get; set; }

        [Required]
        public string Name { get; set; }

        [Required]
        public string Notes { get; set; }

        public bool Done { get; set; }
    }
}

```

Os métodos da API exigem alguma maneira de trabalhar com dados. Use a mesma interface `IToDoRepository` nos usos de exemplo originais do Xamarin:

```

using System.Collections.Generic;
using ToDoApi.Models;

namespace ToDoApi.Interfaces
{
    public interface IToDoRepository
    {
        bool DoesItemExist(string id);
        IEnumerable<ToDoItem> All { get; }
        ToDoItem Find(string id);
        void Insert(ToDoItem item);
        void Update(ToDoItem item);
        void Delete(string id);
    }
}

```

Para esta amostra, a implementação apenas usa uma coleção particular de itens:

```

using System.Collections.Generic;
using System.Linq;
using ToDoApi.Interfaces;
using ToDoApi.Models;

namespace ToDoApi.Services
{
    public class ToDoRepository : IToDoRepository
    {
        private List<ToDoItem> _ToDoList;

        public ToDoRepository()
        {
            InitializeData();
        }

        public IEnumerable<ToDoItem> All
        {
            get { return _ToDoList; }
        }

        public bool DoesItemExist(string id)
        {
            return _ToDoList.Any(item => item.ID == id);
        }
    }
}

```

```

    }

    public ToDoItem Find(string id)
    {
        return _ToDoList.FirstOrDefault(item => item.ID == id);
    }

    public void Insert(ToDoItem item)
    {
        _ToDoList.Add(item);
    }

    public void Update(ToDoItem item)
    {
        var todoItem = this.Find(item.ID);
        var index = _ToDoList.IndexOf(todoItem);
        _ToDoList.RemoveAt(index);
        _ToDoList.Insert(index, item);
    }

    public void Delete(string id)
    {
        _ToDoList.Remove(this.Find(id));
    }

    private void InitializeData()
    {
        _ToDoList = new List<ToDoItem>();

        var todoItem1 = new ToDoItem
        {
            ID = "6bb8a868-dba1-4f1a-93b7-24ebce87e243",
            Name = "Learn app development",
            Notes = "Attend Xamarin University",
            Done = true
        };

        var todoItem2 = new ToDoItem
        {
            ID = "b94afb54-a1cb-4313-8af3-b7511551b33b",
            Name = "Develop apps",
            Notes = "Use Xamarin Studio/Visual Studio",
            Done = false
        };

        var todoItem3 = new ToDoItem
        {
            ID = "ecfa6f80-3671-4911-aabe-63cc442c1ecf",
            Name = "Publish apps",
            Notes = "All app stores",
            Done = false,
        };

        _ToDoList.Add(todoItem1);
        _ToDoList.Add(todoItem2);
        _ToDoList.Add(todoItem3);
    }
}
}

```

Configure a implementação em *Startup.cs*:

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddMvc();

    services.AddSingleton<IToDoRepository, ToDoRepository>();
}
```

Neste ponto, você está pronto para criar o *ToDoItemsController*.

**TIP**

Saiba mais sobre como criar APIs Web em [Criar sua primeira API Web com o ASP.NET Core MVC e o Visual Studio](#).

## Criando o controlador

Adicione um novo controlador ao projeto, *ToDoItemsController*. Ele deve herdar de `Microsoft.AspNetCore.Mvc.Controller`. Adicione um atributo `Route` para indicar que o controlador manipulará as solicitações feitas para caminhos que começam com `api/todoitems`. O token `[controller]` na rota é substituído pelo nome do controlador (com a omissão do sufixo `Controller`) e é especialmente útil para rotas globais. Saiba mais sobre o [roteamento](#).

O controlador requer um `IToDoRepository` para a função; solicite uma instância desse tipo usando o construtor do controlador. No tempo de execução, esta instância será fornecida com suporte do framework para [injeção de dependência](#).

```
using System;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using ToDoApi.Interfaces;
using ToDoApi.Models;

namespace ToDoApi.Controllers
{
    [Route("api/[controller]")]
    public class ToDoItemsController : Controller
    {
        private readonly IToDoRepository _ToDoRepository;

        public ToDoItemsController(IToDoRepository ToDoRepository)
        {
            _ToDoRepository = ToDoRepository;
        }
    }
}
```

Essa API é compatível com quatro verbos HTTP diferentes para executar operações CRUD (Criar, Ler, Atualizar, Excluir) na fonte de dados. A mais simples delas é a operação Read, que corresponde a uma solicitação HTTP GET.

### Lendo itens

A solicitação de uma lista de itens é feita com uma solicitação GET ao método `List`. O atributo `[HttpGet]` no método `List` indica que esta ação só deve lidar com as solicitações GET. A rota para esta ação é a rota especificada no controlador. Você não precisa necessariamente usar o nome da ação como parte da rota. Você precisa garantir que cada ação tem uma rota exclusiva e não ambígua. Os atributos de roteamento podem ser aplicados nos níveis de método e controlador para criar rotas específicas.

```
[HttpGet]
public IActionResult List()
{
    return Ok(_todoRepository.All);
}
```

O método `List` retorna um código de resposta OK 200 e todos os itens de tarefas, serializados como JSON.

Você pode testar o novo método de API usando uma variedade de ferramentas, como [Postman](#). Veja abaixo:

The screenshot shows the Postman application interface. At the top, there are tabs for 'Runner', 'Import', 'Builder' (which is selected), 'Team Library', and various status indicators like 'SYNC OFF'. Below the header, the URL is set to `http://192.168.1.207:5000/api/todoitems`. The request method is set to 'GET'. On the right side of the request bar, there are 'Send' and 'Save' buttons. Under the request bar, there are tabs for 'Authorization', 'Headers' (which is selected), 'Body', 'Pre-request Script', and 'Tests'. Below these tabs, there's a table for setting key-value pairs. The 'Body' tab is selected, showing a 'Pretty' view of the JSON response. The response body contains the following JSON array:

```

1 [ ]
2 [
3     {
4         "id": "6bb8a868-dba1-4f1a-93b7-24ebce87e243",
5         "name": "Learn app development",
6         "notes": "Attend Xamarin University",
7         "done": true
8     },
9     {
10        "id": "b94afb54-a1cb-4313-8af3-b7511551b33b",
11        "name": "Develop apps",
12        "notes": "Use Xamarin Studio/Visual Studio",
13        "done": false
14    },
15    {
16        "id": "ecfa6f80-3671-4911-aabe-63cc442c1ecf",
17        "name": "Publish apps",
18        "notes": "All app stores",
19        "done": false
20    }
]
```

The response status is 200 OK and the time taken was 69 ms.

## Criando itens

Por convenção, a criação de novos itens de dados é mapeada para o verbo HTTP POST. O método `Create` tem um atributo `[HttpPost]` aplicado a ele e aceita uma instância `ToDoItem`. Como o argumento `item` será enviado no corpo de POST, este parâmetro será decorado com o atributo `[FromBody]`.

Dentro do método, o item é verificado quanto à validade e existência anterior no armazenamento de dados e, se nenhum problema ocorrer, ele será adicionado usando o repositório. A verificação de `ModelState.IsValid` executa a [validação do modelo](#) e deve ser feita em todos os métodos de API que aceitam a entrada do usuário.

```

[HttpPost]
public IActionResult Create([FromBody] ToDoItem item)
{
    try
    {
        if (item == null || !ModelState.IsValid)
        {
            return BadRequest(ErrorCode.TodoItemNameAndNotesRequired.ToString());
        }
        bool itemExists = _ToDoRepository.DoesItemExist(item.ID);
        if (itemExists)
        {
            return StatusCode(StatusCodes.Status409Conflict, ErrorCode.TodoItemIDInUse.ToString());
        }
        _ToDoRepository.Insert(item);
    }
    catch (Exception)
    {
        return BadRequest(ErrorCode.CouldNotCreateItem.ToString());
    }
    return Ok(item);
}

```

A amostra usa uma enumeração que contém códigos de erro que são passados para o cliente móvel:

```

public enum ErrorCode
{
    TodoItemNameAndNotesRequired,
    TodoItemIDInUse,
    RecordNotFound,
    CouldNotCreateItem,
    CouldNotUpdateItem,
    CouldNotDeleteItem
}

```

Teste a adição de novos itens usando Postman escolhendo o verbo POST fornecendo o novo objeto no formato JSON no corpo da solicitação. Você também deve adicionar um cabeçalho de solicitação que especifica um

de .

The screenshot shows the Postman application interface. At the top, there are tabs for Runner, Import, Builder (which is selected), and Team Library. There are also icons for Sync (Sync Off), Notifications, and a dropdown menu.

In the main area, a request is being built:

- Method:** POST
- URL:** http://192.168.1.207:5000/api/todoitems
- Body:** (selected tab) contains the following JSON payload:

```
1 {  
2   "ID": "6bb8b868-dba1-4f1a-93b7-24ebce87243",  
3   "Name": "A Test Item",  
4   "Notes": "asdf",  
5   "Done": false  
6 }
```
- Params:** None
- Send:** (blue button)
- Save:** (button with dropdown)

Below the request, the response is displayed:

- Status:** 200 OK
- Time:** 227 ms
- Body:** (selected tab) shows the same JSON response as the request:

```
1 {  
2   "id": "6bb8b868-dba1-4f1a-93b7-24ebce87243",  
3   "name": "A Test Item",  
4   "notes": "asdf",  
5   "done": false  
6 }
```
- Cookies:** None
- Headers:** (4) (dropdown)
- Tests:** None

O método retorna o item recém-criado na resposta.

## Atualizando itens

A modificação de registros é feita com as solicitações HTTP PUT. Além desta mudança, o método `Edit` é quase idêntico ao `Create`. Observe que, se o registro não for encontrado, a ação `Edit` retornará uma resposta `NotFound` (404).

```
[HttpPut]
public IActionResult Edit([FromBody] ToDoItem item)
{
    try
    {
        if (item == null || !ModelState.IsValid)
        {
            return BadRequest(ErrorCode.TodoItemNameAndNotesRequired.ToString());
        }
        var existingItem = _ToDoRepository.Find(item.ID);
        if (existingItem == null)
        {
            return NotFound(ErrorCode.RecordNotFound.ToString());
        }
        _ToDoRepository.Update(item);
    }
    catch (Exception)
    {
        return BadRequest(ErrorCode.CouldNotUpdateItem.ToString());
    }
    return NoContent();
}
```

Para testar com Postman, altere o verbo para PUT. Especifique os dados do objeto atualizado no corpo da solicitação.

The screenshot shows the Postman application interface. The top navigation bar includes 'Runner', 'Import', 'Builder' (which is highlighted in orange), 'Team Library', and various status indicators. The main workspace displays a 'PUT' request to the URL `http://192.168.1.207:5000/api/todoitems`. The 'Body' tab is selected, showing a raw JSON payload:

```
1 {
2     "ID": "6bb8b868-dba1-4f1a-93b7-24ebce87243",
3     "Name": "An UPDATED Test Item",
4     "Notes": "Some updated notes",
5     "Done": true
6 }
```

Below the request details, the response pane shows a successful `204 No Content` status with a duration of `91 ms`.

Este método retornará uma resposta `NoContent` (204) quando obtiver êxito, para manter a consistência com a API já existente.

## Excluindo itens

A exclusão de registros é feita por meio da criação de solicitações de exclusão para o serviço e por meio do envio do ID do item a ser excluído. Assim como as atualizações, as solicitações de itens que não existem receberão respostas `NotFound`. Caso contrário, uma solicitação bem-sucedida receberá uma resposta `NoContent` (204).

```
[HttpDelete("{id}")]
public IActionResult Delete(string id)
{
    try
    {
        var item = _todoRepository.Find(id);
        if (item == null)
        {
            return NotFound(ErrorCode.RecordNotFound.ToString());
        }
        _todoRepository.Delete(id);
    }
    catch (Exception)
    {
        return BadRequest(ErrorCode.CouldNotDeleteItem.ToString());
    }
    return NoContent();
}
```

Observe que, ao testar a funcionalidade de exclusão, nada é necessário no Corpo da solicitação.

The screenshot shows the Postman application interface. At the top, there are tabs for Runner, Import, and Builder, with Builder selected. Below the tabs, the URL is set to `http://192.168.1.207:5000/api/todoitems/6bb8b8`. The method is set to `DELETE`. The Headers tab shows two entries: `Authorization` and `Content-Type` (`application/json`). The Body tab is selected and contains a JSON object with a single key-value pair: `1`. The response section at the bottom shows a status of `204 No Content` and a time of `91 ms`.

## Convenções de Web API comuns

À medida que você desenvolve serviços de back-end para seu aplicativo, desejará criar um conjunto consistente de convenções ou políticas para lidar com preocupações paralelas. Por exemplo, no serviço mostrado acima, as solicitações de registros específicos que não foram encontrados receberam uma resposta `NotFound`, em vez de uma resposta `BadRequest`. Da mesma forma, os comandos feitos para esse serviço que passaram tipos associados a um modelo sempre verificaram `ModelState.IsValid` e retornaram um `BadRequest` para tipos de modelo inválidos.

Depois de identificar uma diretiva comum para suas APIs, você geralmente pode encapsulá-la em um [filtro](#). Saiba mais sobre [como encapsular políticas comuns da API em aplicativos ASP.NET Core MVC](#).

## Recursos adicionais

- [Autenticação e autorização](#)

# Tutorial: Criar um aplicativo Web de Páginas do Razor com o ASP.NET Core

04/02/2019 • 2 minutes to read • [Edit Online](#)

Esta série de tutoriais explica as noções básicas sobre a criação de um aplicativo Web Razor Pages.

Para obter uma introdução mais avançada direcionada a desenvolvedores experientes, confira [Introdução a Razor Pages](#).

Esta série inclui os seguintes tutoriais:

1. [Criar um aplicativo Web de Páginas do Razor](#)
2. [Adicionar um modelo a um aplicativo de Páginas do Razor](#)
3. [Gerar páginas do Razor por scaffolding](#)
4. [Trabalhar com um banco de dados](#)
5. [Atualizar páginas do Razor](#)
6. [Adicionar pesquisa](#)
7. [Adicionar um novo campo](#)
8. [Adicionar validação](#)

No final, você terá um aplicativo que pode exibir e gerenciar um banco de dados de filmes.

The screenshot shows a browser window with the title "Index - Movie". The address bar contains the URL <https://localhost:5001/Movies?SearchString=ghost>. The page content is titled "Index" and includes a "Create New" button. Below it is a search bar with the placeholder "Title: ghost" and a "Filter" button. A table lists two movie entries:

Title	Release Date	Genre	Price	
Ghostbusters	3/13/1984	Comedy	8.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Ghostbusters 2	2/23/1986	Comedy	9.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

At the bottom of the page, there is a copyright notice: "© 2019 - RazorPagesMovie - [Privacy](#)".

# Criar um aplicativo Web com o ASP.NET Core MVC

23/01/2019 • 2 minutes to read • [Edit Online](#)

Este tutorial ensina a usar o desenvolvimento Web do ASP.NET Core MVC com controladores e exibições. Se você é novo no desenvolvimento da Web ASP.NET Core, considere a versão [Razor Pages](#) deste tutorial, que oferece um ponto inicial mais simples.

A série de tutoriais inclui o seguinte:

1. [Introdução](#)
2. [Adicionar um controlador](#)
3. [Adicionar uma exibição](#)
4. [Adicionar um modelo](#)
5. [Trabalhar com o SQL Server LocalDB](#)
6. [Exibições e métodos do controlador](#)
7. [Adicionar pesquisa](#)
8. [Adicionar um novo campo](#)
9. [Adicionar validação](#)
10. [Examinar os métodos Details e Delete](#)

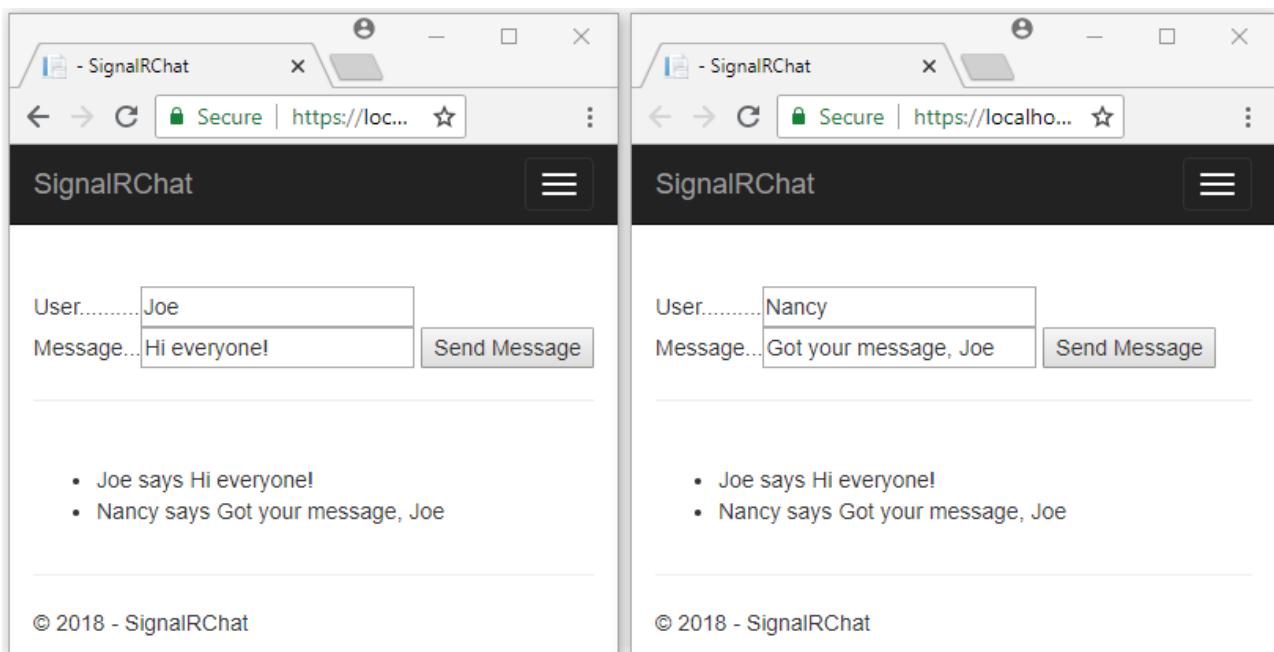
# Tutorial: Introdução ao SignalR para ASP.NET Core

28/01/2019 • 11 minutes to read • [Edit Online](#)

Este tutorial ensina as noções básicas da criação de um aplicativo em tempo real usando o SignalR. Você aprenderá como:

- Crie um projeto Web.
- Adicionar uma biblioteca de clientes do SignalR.
- Criar um hub do SignalR.
- Configurar o projeto para usar o SignalR.
- Adicione o código que envia mensagens de qualquer cliente para todos os clientes conectados.

No final, você terá um aplicativo de chat funcionando:



[Exibir ou baixar um código de exemplo \(como baixar\).](#)

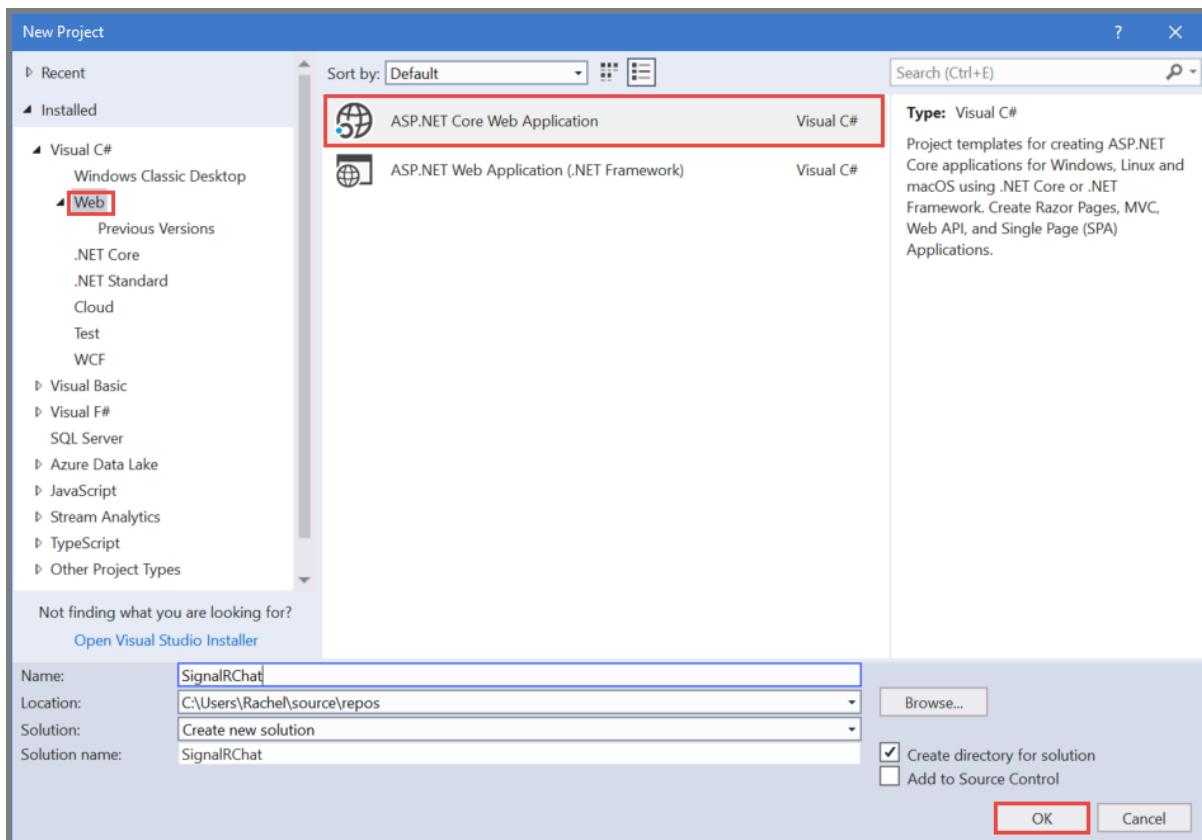
## Pré-requisitos

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- [Visual Studio 2017 versão 15.9 ou posterior](#) com a carga de trabalho **ASP.NET e desenvolvimento para a Web**
- [SDK 2.2 ou posterior do .NET Core](#)

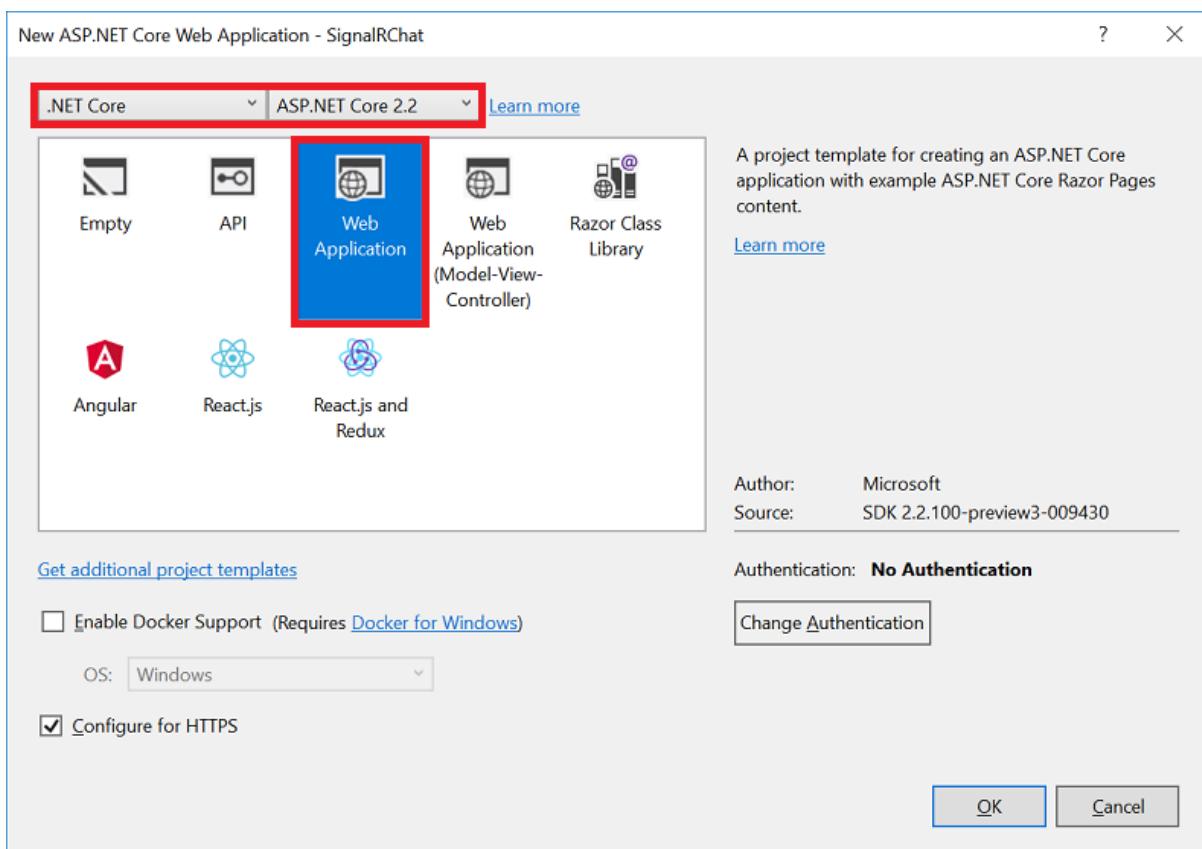
## Criar um projeto Web

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- No menu, selecione **Arquivo > Novo Projeto**.

- Na caixa de diálogo **Novo Projeto**, selecione **Instalado > Visual C# > Web > Aplicativo Web ASP.NET Core**. Dê ao projeto o nome de *SignalRChat*.



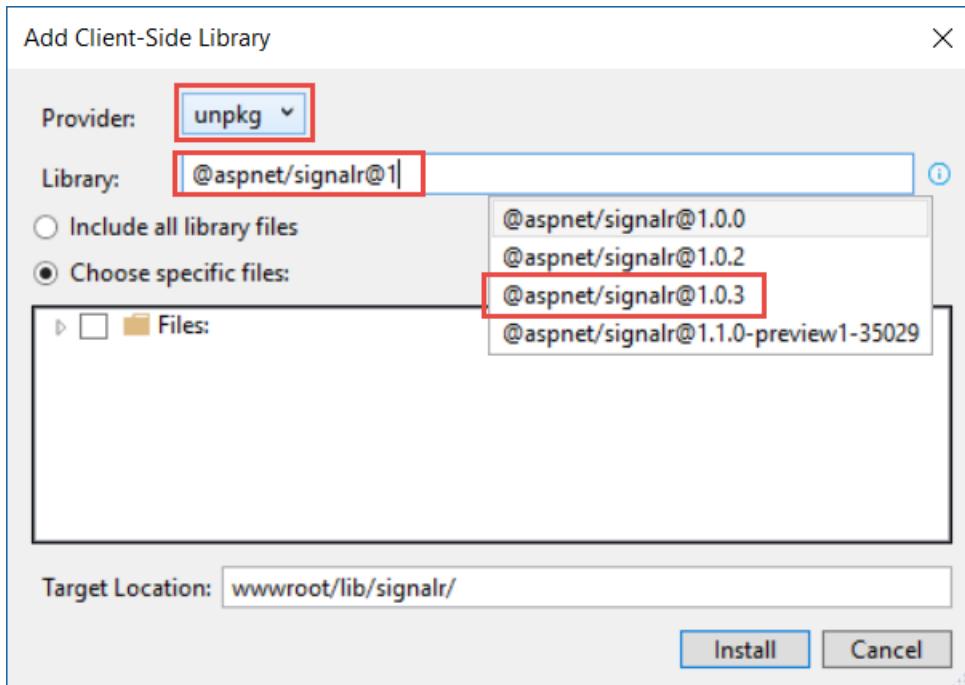
- Selecione **Aplicativo Web** para criar um projeto que usa Razor Pages.
- Selecione uma estrutura de destino do **.NET Core**, selecione **ASP.NET Core 2.2** e clique em **OK**.



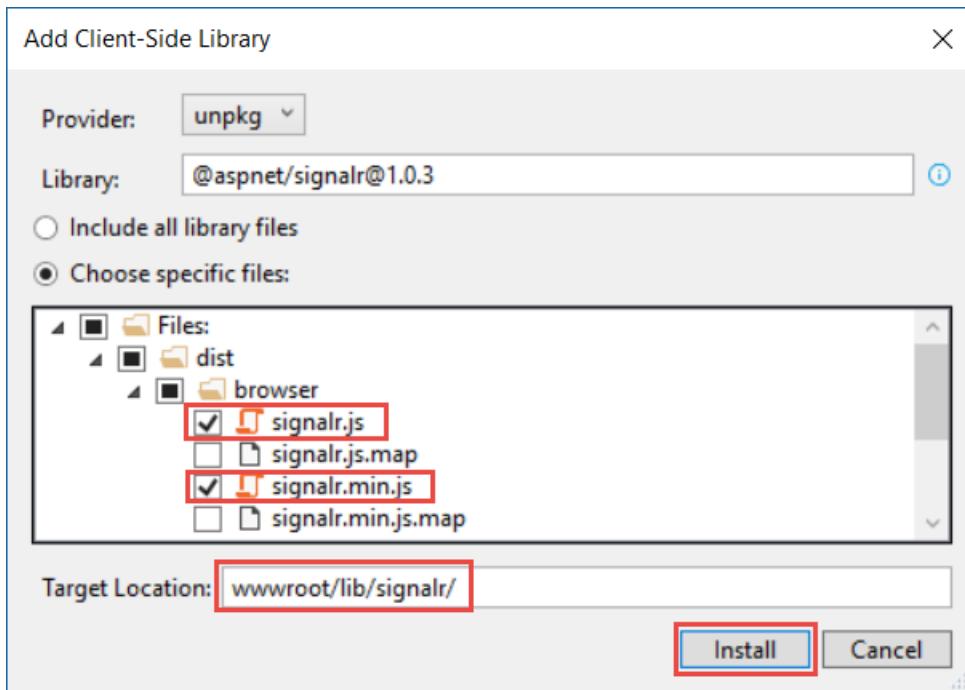
Adicionar a biblioteca de clientes do SignalR

A biblioteca do servidor SignalR está incluída no metapacote `Microsoft.AspNetCore.App`. A biblioteca de clientes do JavaScript não é incluída automaticamente no projeto. Neste tutorial, você usará o LibMan (Library Manager) para obter a biblioteca de clientes de `unpkg`. `unpkg` é uma CDN (rede de distribuição de conteúdo) que pode distribuir qualquer conteúdo do npm, o gerenciador de pacotes do Node.js.

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- No **Gerenciador de Soluções**, clique com o botão direito do mouse no projeto e selecione **Adicionar > Biblioteca do Lado do Cliente**.
- Na caixa de diálogo **Adicionar Biblioteca do Lado do Cliente**, para **Provedor**, selecione **unpkg**.
- Para **Biblioteca**, insira `@aspnet/signalr@1` e selecione a versão mais recente que não seja uma versão prévia.



- Selecione **Escolher arquivos específicos**, expanda a pasta *distribuidor/navegador* e selecione `signalr.js` e `signalr.min.js`.
- Defina **Localização de Destino** como `wwwroot/lib/signalr/` e selecione **Instalar**.



O LibMan cria uma pasta `wwwroot/lib/signalr` e copia os arquivos selecionados para ela.

## Criar um hub do SignalR

Um *hub* é uma classe que funciona como um pipeline de alto nível que lida com a comunicação entre cliente e servidor.

- Na pasta do projeto SignalRChat, crie uma pasta *Hubs*.
- Na pasta *Hubs*, crie um arquivo *ChatHub.cs* com o código a seguir:

```
using Microsoft.AspNetCore.SignalR;
using System.Threading.Tasks;

namespace SignalRChat.Hubs
{
    public class ChatHub : Hub
    {
        public async Task SendMessage(string user, string message)
        {
            await Clients.All.SendAsync("ReceiveMessage", user, message);
        }
    }
}
```

A classe `ChatHub` é herda da classe `Hub` do SignalR. A classe `Hub` gerencia conexões, grupos e sistemas de mensagens.

O método `SendMessage` pode ser chamado por um cliente conectado para enviar uma mensagem a todos os clientes. O código cliente do JavaScript que chama o método é mostrado posteriormente no tutorial. O código do SignalR é assíncrono para fornecer o máximo de escalabilidade.

## Configurar o SignalR

O servidor do SignalR precisa ser configurado para passar solicitações do SignalR ao SignalR.

- Adicione o seguinte código realçado ao arquivo *Startup.cs*.

```

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using SignalRChat.Hubs;

namespace SignalRChat
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.Configure<CookiePolicyOptions>(options =>
            {
                // This lambda determines whether user consent for non-essential cookies is needed for a
                given request.
                options.CheckConsentNeeded = context => true;
                options.MinimumSameSitePolicy = SameSiteMode.None;
            });

            services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

            services.AddSignalR();
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request
        pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Error");
                app.UseHsts();
            }

            app.UseHttpsRedirection();
            app.UseStaticFiles();
            app.UseCookiePolicy();
            app.UseSignalR(routes =>
            {
                routes.MapHub<ChatHub>("/chatHub");
            });
            app.UseMvc();
        }
    }
}

```

Essas alterações adicionam o SignalR ao sistema de injeção de dependência e ao pipeline do middleware do ASP.NET Core.

## Adicionar o código de cliente do SignalR

- Substitua o conteúdo *Pages\Index.cshtml* pelo código a seguir:

```
@page
<div class="container">
    <div class="row">&nbsp;</div>
    <div class="row">
        <div class="col-6">&nbsp;</div>
        <div class="col-6">
            User.....<input type="text" id="userInput" />
            <br />
            Message...<input type="text" id="messageInput" />
            <input type="button" id="sendButton" value="Send Message" />
        </div>
    </div>
    <div class="row">
        <div class="col-12">
            <hr />
        </div>
    </div>
    <div class="row">
        <div class="col-6">&nbsp;</div>
        <div class="col-6">
            <ul id="messagesList"></ul>
        </div>
    </div>
</div>
<script src("~/lib/signalr/dist/browser/signalr.js")></script>
<script src "~/js/chat.js"></script>
```

O código anterior:

- Cria as caixas de texto para o nome e a mensagem de texto e um botão Enviar.
- Cria uma lista com `id="messagesList"` para exibir as mensagens recebidas do hub do SignalR.
- Inclui referências de script ao SignalR e ao código do aplicativo *chat.js* que você criará na próxima etapa.
- Na pasta *wwwroot/js*, crie um arquivo *chat.js* com o código a seguir:

```

"use strict";

var connection = new signalR.HubConnectionBuilder().withUrl("/chatHub").build();

//Disable send button until connection is established
document.getElementById("sendButton").disabled = true;

connection.on("ReceiveMessage", function (user, message) {
    var msg = message.replace(/&/g, "&").replace(/</g, "<").replace(/>/g, ">");
    var encodedMsg = user + " says " + msg;
    var li = document.createElement("li");
    li.textContent = encodedMsg;
    document.getElementById("messagesList").appendChild(li);
});

connection.start().then(function(){
    document.getElementById("sendButton").disabled = false;
}).catch(function (err) {
    return console.error(err.toString());
});

document.getElementById("sendButton").addEventListener("click", function (event) {
    var user = document.getElementById("userInput").value;
    var message = document.getElementById("messageInput").value;
    connection.invoke("SendMessage", user, message).catch(function (err) {
        return console.error(err.toString());
    });
    event.preventDefault();
});

```

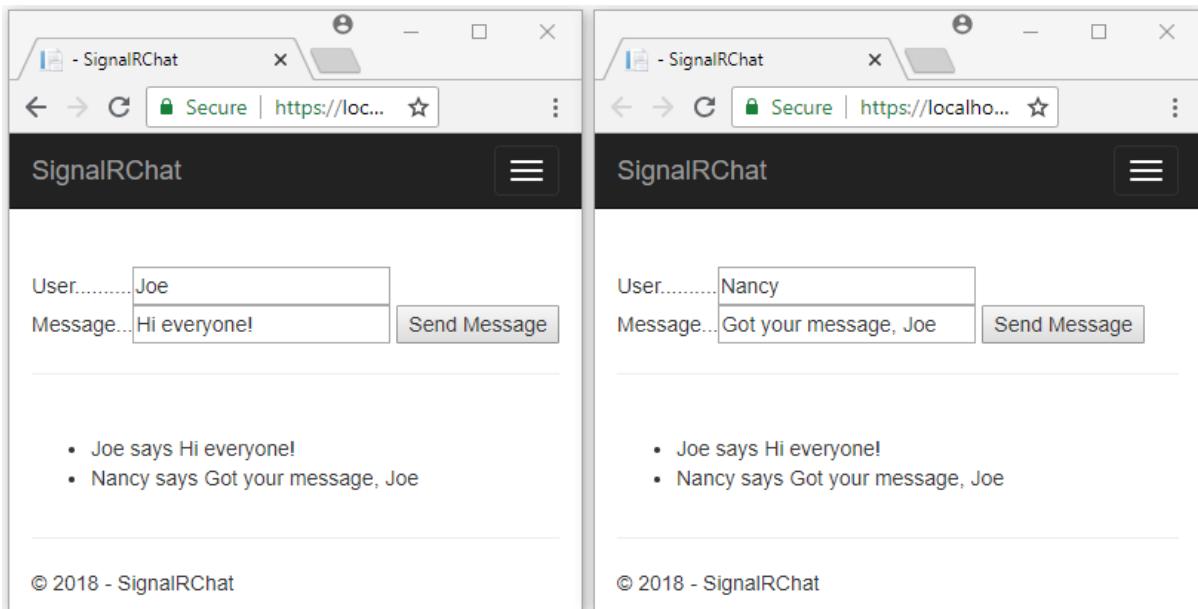
O código anterior:

- Cria e inicia uma conexão.
- Adiciona no botão Enviar um manipulador que envia mensagens ao hub.
- Adiciona no objeto de conexão um manipulador que recebe mensagens do hub e as adiciona à lista.

## Executar o aplicativo

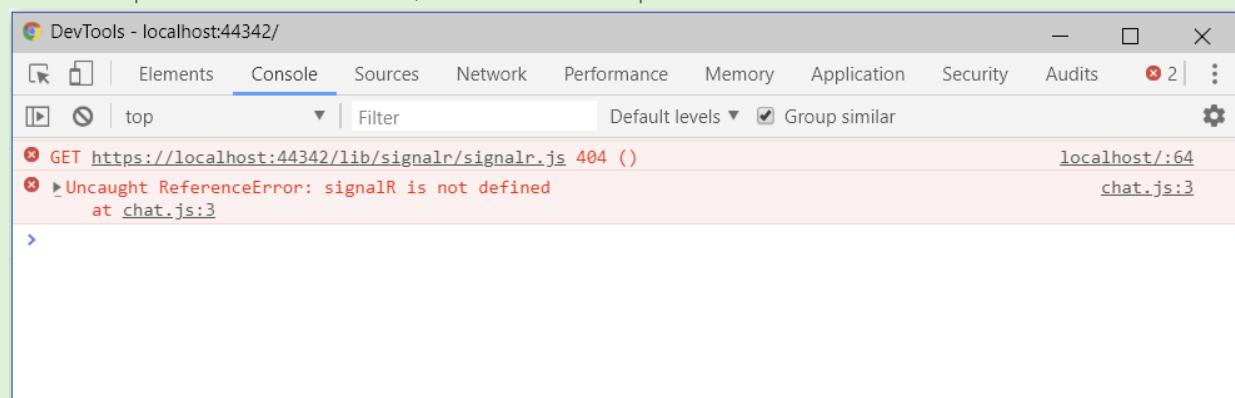
- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- Pressione **CTRL + F5** para executar o aplicativo sem depuração.
- Copie a URL da barra de endereços, abra outra instância ou guia do navegador e cole a URL na barra de endereços.
- Escolha qualquer navegador, insira um nome e uma mensagem e selecione o botão **Enviar Mensagem**.

O nome e a mensagem são exibidos em ambas as páginas instantaneamente.



#### TIP

Se o aplicativo não funcionar, abra as ferramentas para desenvolvedores do navegador (F12) e accese o console. Você pode encontrar erros relacionados ao código HTML e JavaScript. Por exemplo, suponha que você coloque `signalr.js` em uma pasta diferente daquela direcionada. Nesse caso, a referência a esse arquivo não funcionará e ocorrerá um erro 404 no console.



## Próximas etapas

Neste tutorial, você aprendeu como:

- Criar um projeto de aplicativo Web.
- Adicionar uma biblioteca de clientes do SignalR.
- Criar um hub do SignalR.
- Configurar o projeto para usar o SignalR.
- Adicionar o código que usa o hub para enviar mensagens de qualquer cliente para todos os clientes conectados.

Para saber mais sobre o SignalR, confira a introdução:

[Introdução ao ASP.NET Core SignalR](#)

# Usar o SignalR do ASP.NET Core com TypeScript e Webpack

28/01/2019 • 19 minutes to read • [Edit Online](#)

Por [Sébastien Sougnez](#) e [Scott Addie](#)

O [Webpack](#) habilita os desenvolvedores a agrupar e criar recursos de um aplicativo Web do lado do cliente. Este tutorial demonstra como usar o Webpack em um aplicativo Web SignalR do ASP.NET Core cujo cliente é escrito em [TypeScript](#).

Neste tutorial, você aprenderá como:

- Gerar um aplicativo inicial SignalR do ASP.NET Core por scaffold
- Configurar o cliente TypeScript do SignalR
- Configurar um pipeline de build usando o Webpack
- Configurar o servidor SignalR
- Habilitar a comunicação entre o cliente e o servidor

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Prerequisites

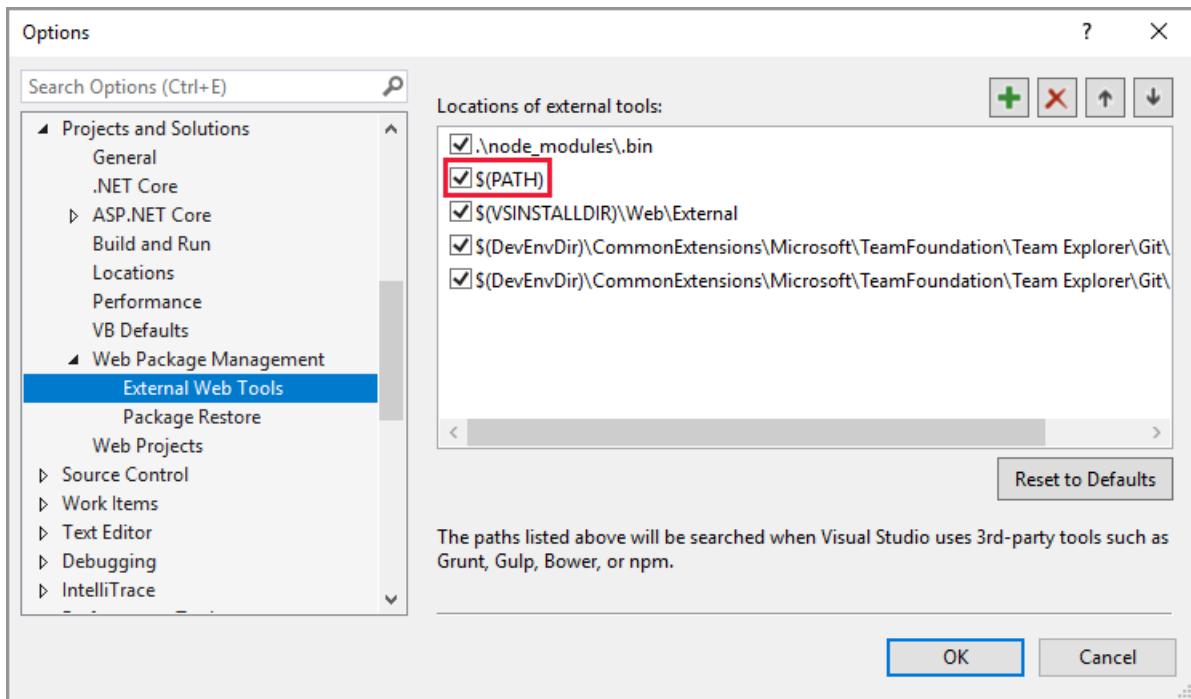
- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio 2017 versão 15.9 ou posterior](#) com a carga de trabalho **ASP.NET e desenvolvimento para a Web**
- [SDK 2.2 ou posterior do .NET Core](#)

## Criar o aplicativo Web do ASP.NET Core

- [Visual Studio](#)
- [Visual Studio Code](#)

Configure o Visual Studio para pesquisar o npm na variável de ambiente *PATH*. Por padrão, o Visual Studio usa a versão do npm encontrada no diretório de instalação. Siga estas instruções no Visual Studio:

1. Navegue para **Ferramentas > Opções > Projetos e Soluções > Gerenciamento de Pacotes da Web > Ferramentas da Web Externas**.
2. Selecione a entrada `$(PATH)` na lista. Clique na seta para cima para mover a entrada para a segunda posição da lista.



A configuração do Visual Studio foi concluída. É hora de criar o projeto.

1. Use a opção do menu **Arquivo > Novo > Projeto** e escolha o modelo **Aplicativo Web do ASP.NET Core**.
2. Dê ao projeto o nome *SignalRWebPack* e selecione **OK**.
3. Selecione *.NET Core* no menu suspenso da estrutura de destino e selecione *ASP.NET Core 2.2* no menu suspenso do seletor de estrutura. Selecione o modelo **Vazio** e selecione **OK**.

## Configurar Webpack e TypeScript

As etapas a seguir configuram a conversão do TypeScript para JavaScript e o agrupamento de recursos no lado do cliente.

1. Execute o seguinte comando na raiz do projeto para criar um arquivo *package.json*:

```
npm init -y
```

2. Adicione a propriedade destacada ao arquivo *package.json*:

```
{
  "name": "SignalRWebPack",
  "version": "1.0.0",
  "private": true,
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Definir a propriedade `private` como `true` evita avisos de instalação de pacote na próxima etapa.

3. Instalar os pacotes de npm exigidos. Execute o seguinte comando na raiz do projeto:

```
npm install -D -E clean-webpack-plugin@0.1.19 css-loader@0.28.11 html-webpack-plugin@3.2.0 mini-css-extract-plugin@0.4.0 ts-loader@4.4.1 typescript@2.9.2 webpack@4.12.0 webpack-cli@3.0.6
```

Alguns detalhes de comando a se observar:

- Um número de versão segue o sinal `@` para cada nome de pacote. O npm instala essas versões específicas do pacote.
- A opção `-E` desabilita o comportamento padrão do npm de escrever os operadores de intervalo de [versão semântica](#) para `package.json`. Por exemplo, `"webpack": "4.12.0"` é usado em vez de `"webpack": "^4.12.0"`. Essa opção evita atualizações não intencionais para versões de pacote mais recentes.

Veja os documentos oficiais de [instalação de npm](#) para saber mais detalhes.

#### 4. Substitua a propriedade `scripts` do arquivo `package.json` com o seguinte snippet:

```
"scripts": {  
  "build": "webpack --mode=development --watch",  
  "release": "webpack --mode=production",  
  "publish": "npm run release && dotnet publish -c Release"  
},
```

Uma explicação sobre os scripts:

- `build`: agrupa os recursos do lado do cliente no modo de desenvolvimento e busca alterações no arquivo. O observador de arquivos faz com que o lote se regenere toda vez que um arquivo de projeto é alterado. A opção `mode` desabilita otimizações de produção, como tree shaking e minificação. Use somente `build` no desenvolvimento.
- `release`: agrupa recursos do lado do cliente no modo de produção.
- `publish`: executa o script `release` para agrupar recursos do lado do cliente no modo de produção. Chama o comando [publicar](#) da CLI do .NET Core para publicar o aplicativo.

#### 5. Crie um arquivo chamado `webpack.config.js` na raiz do projeto, com o seguinte conteúdo:

```

const path = require("path");
const HtmlWebpackPlugin = require("html-webpack-plugin");
const CleanWebpackPlugin = require("clean-webpack-plugin");
const MiniCssExtractPlugin = require("mini-css-extract-plugin");

module.exports = {
  entry: "./src/index.ts",
  output: {
    path: path.resolve(__dirname, "wwwroot"),
    filename: "[name].[chunkhash].js",
    publicPath: "/"
  },
  resolve: {
    extensions: [".js", ".ts"]
  },
  module: {
    rules: [
      {
        test: /\.ts$/,
        use: "ts-loader"
      },
      {
        test: /\.css$/,
        use: [MiniCssExtractPlugin.loader, "css-loader"]
      }
    ]
  },
  plugins: [
    new CleanWebpackPlugin(["wwwroot/*"]),
    new HtmlWebpackPlugin({
      template: "./src/index.html"
    }),
    new MiniCssExtractPlugin({
      filename: "css/[name].[chunkhash].css"
    })
  ]
};


```

O arquivo precedente configura a compilação Webpack. Detalhes de configuração a serem notados:

- A propriedade `output` substitui o valor padrão do `dist`. Em vez disso, o lote é emitido no diretório `wwwroot`.
  - A matriz `resolve.extensions` inclui `.js` para importar o JavaScript do cliente SignalR.
6. Crie um novo diretório `src` na raiz do projeto. O propósito é armazenar os ativos do lado do cliente do projeto.
  7. Crie `src/index.html` com o seguinte conteúdo.

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>ASP.NET Core SignalR</title>
</head>
<body>
    <div id="divMessages" class="messages">
    </div>
    <div class="input-zone">
        <label id="lblMessage" for="tbMessage">Message:</label>
        <input id="tbMessage" class="input-zone-input" type="text" />
        <button id="btnSend">Send</button>
    </div>
</body>
</html>

```

A HTML precedente define a marcação clichê da página inicial.

8. Crie um novo diretório `src/css`. Seu propósito é armazenar os arquivos do projeto `.css`.
9. Crie `src/css/main.css` com o seguinte conteúdo:

```

*, *::before, *::after {
    box-sizing: border-box;
}

html, body {
    margin: 0;
    padding: 0;
}

.input-zone {
    align-items: center;
    display: flex;
    flex-direction: row;
    margin: 10px;
}

.input-zone-input {
    flex: 1;
    margin-right: 10px;
}

.message-author {
    font-weight: bold;
}

.messages {
    border: 1px solid #000;
    margin: 10px;
    max-height: 300px;
    min-height: 300px;
    overflow-y: auto;
    padding: 5px;
}

```

O arquivo precedente `main.css` define o estilo do aplicativo.

10. Crie `src/tsconfig.json` com o seguinte conteúdo:

```
{
  "compilerOptions": {
    "target": "es5"
  }
}
```

O código precedente configura o compilador TypeScript para produzir um JavaScript compatível com [ECMAScript](#) 5.

11. Crie `src/index.ts` com o seguinte conteúdo:

```
import "./css/main.css";

const divMessages: HTMLDivElement = document.querySelector("#divMessages");
const tbMessage: HTMLInputElement = document.querySelector("#tbMessage");
const btnSend: HTMLButtonElement = document.querySelector("#btnSend");
const username = new Date().getTime();

tbMessage.addEventListener("keyup", (e: KeyboardEvent) => {
  if (e.keyCode === 13) {
    send();
  }
});

btnSend.addEventListener("click", send);

function send() {
```

O TypeScript precedente recupera referências a elementos DOM e anexa dois manipuladores de eventos:

- `keyup` : esse evento é acionado quando o usuário digita algo na caixa de texto identificada como `tbMessage`. A função `send` é chamada quando o usuário pressionar a tecla **Enter**.
- `click` : esse evento é acionado quando o usuário clica no botão **Enviar**. A função `send` é chamada.

## Configurar o aplicativo do ASP.NET Core

1. O código fornecido no método `Startup.Configure` exibe *Olá, Mundo*. Substitua a chamada para o método `app.Run` com chamadas para [UseDefaultFiles](#) e [UseStaticFiles](#).

```
app.UseDefaultFiles();
app.UseStaticFiles();
```

O código precedente permite que o servidor localize e forneça o arquivo *index.html*, se o usuário inserir a URL completa ou a URL raiz do aplicativo Web.

2. Chame [AddSignalR](#) no método `Startup.ConfigureServices`. Adiciona serviços SignalR ao seu projeto.

```
services.AddSignalR();
```

3. Mapeie uma rota `/hub` para o hub `chatHub`. Adicione as linhas a seguir ao final do método `Startup.Configure`:

```
app.UseSignalR(options =>
{
    options.MapHub<ChatHub>("/hub");
});
```

4. Crie um novo diretório, chamado *Hubs*, na raiz do projeto. A finalidade é armazenar o hub SignalR, que é criado na próxima etapa.
5. Crie o hub *Hubs/ChatHub.cs* com o código a seguir:

```
using Microsoft.AspNetCore.SignalR;
using System.Threading.Tasks;

namespace SignalRWebPack.Hubs
{
    public class ChatHub : Hub
    {
    }
}
```

6. Adicione o código a seguir ao topo do arquivo *Startup.cs* para resolver a referência `chatHub`:

```
using SignalRWebPack.Hubs;
```

## Habilitar a comunicação entre o cliente e o servidor

Atualmente, o aplicativo exibe um formulário simples para enviar mensagens. Nada acontece quando você tenta fazer alguma coisa. O servidor está escutando uma rota específica, mas não faz nada com as mensagens enviadas.

1. Execute o comando a seguir na raiz do projeto:

```
npm install @aspnet/signalr
```

O comando precedente instala o [cliente TypeScript do SignalR](#), que permite ao cliente enviar mensagens para o servidor.

2. Adicione o código destacado ao arquivo *src/index.ts*:

```

import "./css/main.css";
import * as signalR from "@aspnet/signalr";

const divMessages: HTMLDivElement = document.querySelector("#divMessages");
const tbMessage: HTMLInputElement = document.querySelector("#tbMessage");
const btnSend: HTMLButtonElement = document.querySelector("#btnSend");
const username = new Date().getTime();

const connection = new signalR.HubConnectionBuilder()
    .withUrl("/hub")
    .build();

connection.start().catch(err => document.write(err));

connection.on("messageReceived", (username: string, message: string) => {
    let m = document.createElement("div");

    m.innerHTML =
        `<div class="message-author">${username}</div><div>${message}</div>`;

    divMessages.appendChild(m);
    divMessages.scrollTop = divMessages.scrollHeight;
});

tbMessage.addEventListener("keyup", (e: KeyboardEvent) => {
    if (e.keyCode === 13) {
        send();
    }
});

btnSend.addEventListener("click", send);

function send() {
}

```

O código precedente é compatível com o recebimento de mensagens do servidor. A classe `HubConnectionBuilder` cria um novo construtor para configurar a conexão do servidor. A função `withUrl` configura a URL do hub.

O SignalR habilita a troca de mensagens entre um cliente e um servidor. Cada mensagem tem um nome específico. Por exemplo, você pode ter mensagens com o nome `messageReceived` que executam a lógica responsável por exibir a nova mensagem na zona de mensagens. É possível escutar uma mensagem específica por meio da função `on`. Você pode escutar qualquer número de nomes de mensagem. Também é possível passar parâmetros para a mensagem, como o nome do autor e o conteúdo da mensagem recebida. Quando o cliente recebe a mensagem, um novo elemento `div` é criado com o nome do autor e o conteúdo da mensagem em seu atributo `innerHTML`. Ele é adicionado ao elemento principal `div` que exibe as mensagens.

3. Agora que o cliente pode receber mensagens, configure-o para enviá-las. Adicione o código destacado ao arquivo `src/index.ts`:

```

import "./css/main.css";
import * as signalR from "@aspnet/signalr";

const divMessages: HTMLDivElement = document.querySelector("#divMessages");
const tbMessage: HTMLInputElement = document.querySelector("#tbMessage");
const btnSend: HTMLButtonElement = document.querySelector("#btnSend");
const username = new Date().getTime();

const connection = new signalR.HubConnectionBuilder()
    .withUrl("/hub")
    .build();

connection.start().catch(err => document.write(err));

connection.on("messageReceived", (username: string, message: string) => {
    let messageContainer = document.createElement("div");

    messageContainer.innerHTML =
        `<div class="message-author">${username}</div><div>${message}</div>`;

    divMessages.appendChild(messageContainer);
    divMessages.scrollTop = divMessages.scrollHeight;
});

tbMessage.addEventListener("keyup", (e: KeyboardEvent) => {
    if (e.keyCode === 13) {
        send();
    }
});

btnSend.addEventListener("click", send);

function send() {
    connection.send("newMessage", username, tbMessage.value)
        .then(() => tbMessage.value = "");
}

```

Enviar uma mensagem por meio da conexão WebSockets exige uma chamada para o método `send`. O primeiro parâmetro do método é o nome da mensagem. Os dados da mensagem residem nos outros parâmetros. Neste exemplo, uma mensagem identificada como `newMessage` é enviada ao servidor. A mensagem é composta do nome de usuário e da entrada em uma caixa de texto. Se o envio for bem-sucedido, o valor da caixa de texto será limpo.

4. Adicione o método em destaque à classe `ChatHub`:

```

using Microsoft.AspNetCore.SignalR;
using System.Threading.Tasks;

namespace SignalRWebPack.Hubs
{
    public class ChatHub : Hub
    {
        public async Task NewMessage(string username, string message)
        {
            await Clients.All.SendAsync("messageReceived", username, message);
        }
    }
}

```

O código precedente transmite as mensagens recebidas para todos os usuários conectados quando o servidor as recebe. Não é necessário ter um método genérico `on` para receber todas as mensagens. Um método nomeado com o nome da mensagem é suficiente.

Neste exemplo, o cliente TypeScript envia uma mensagem identificada como `newMessage`. O método `NewMessage` de C# espera os dados enviados pelo cliente. Uma chamada é feita para o método `SendAsync` em `Clients.All`. As mensagens recebidas são enviadas a todos os clientes conectados ao hub.

## Testar o aplicativo

Confirme que o aplicativo funciona com as seguintes etapas.

- [Visual Studio](#)
- [Visual Studio Code](#)

1. Execute o Webpack no modo de *versão*. Usando a janela **Console do Gerenciador de Pacotes**, execute o comando a seguir na raiz do projeto. Se você não estiver na raiz do projeto, insira `cd SignalRWebPack` antes de inserir o comando.

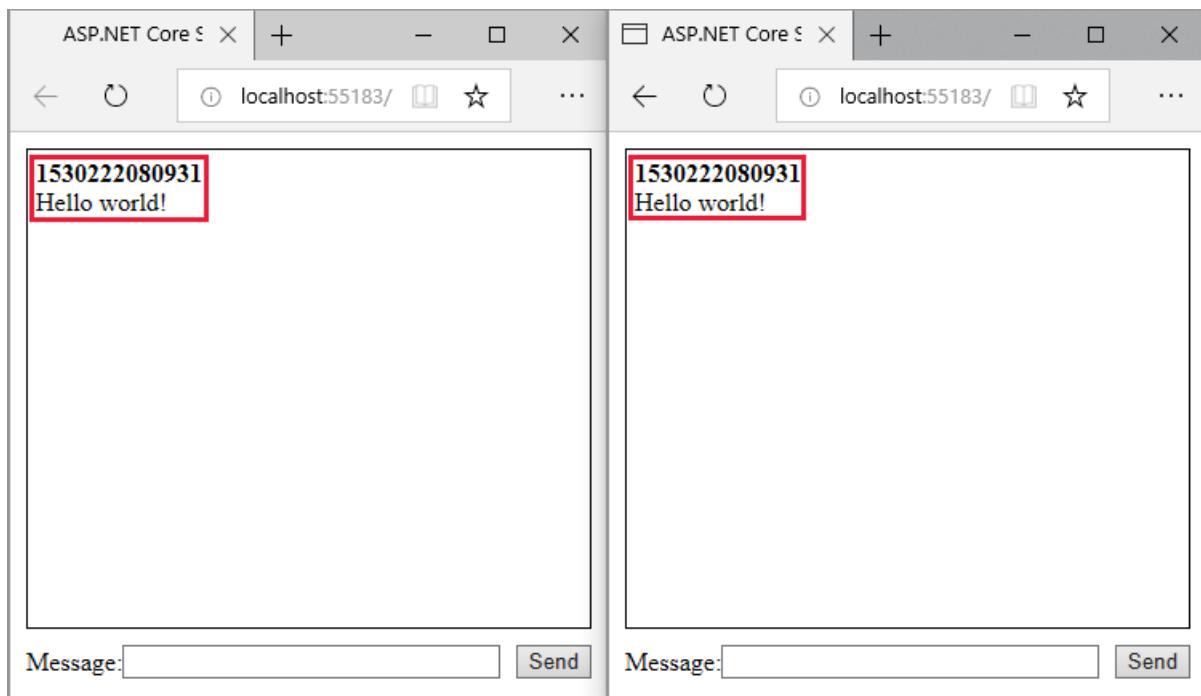
```
npm run release
```

Este comando suspende o fornecimento dos ativos do lado do cliente ao executar o aplicativo. Os ativos são colocados na pasta `wwwroot`.

O Webpack concluiu as seguintes tarefas:

- Limpou os conteúdos do diretório `wwwroot`.
- Converteu o TypeScript para JavaScript, um processo conhecido como *transpilação*.
- Reduziu o tamanho do arquivo JavaScript gerado, um processo conhecido como *minificação*.
- Copiou os arquivos JavaScript, CSS e HTML processados do `src` para o diretório `wwwroot`.
- Injetou os seguintes elementos no arquivo `wwwroot/index.html`:
  - Uma marca `<link>`, que referencia o arquivo `wwwroot/main.<hash>.css`. Essa marca é colocada imediatamente antes do fim da marca `</head>`.
  - Uma marca `<script>`, que referencia o arquivo minificado `wwwroot/main.<hash>.js`. Essa marca é colocada imediatamente antes do fim da marca `</body>`.

2. Selecione **Debug > Iniciar sem depuração** para iniciar o aplicativo em um navegador sem anexar o depurador. O arquivo `wwwroot/index.html` é fornecido em `http://localhost:<port_number>`.
3. Abra outra instância do navegador (qualquer navegador). Cole a URL na barra de endereços.
4. Escolha qualquer navegador, digite algo na caixa de texto **Mensagem** e clique no botão **Enviar**. O nome de usuário exclusivo e a mensagem são exibidas em ambas as páginas instantaneamente.



## Recursos adicionais

- [ASP.NET Core SignalR JavaScript cliente](#)
- [Usando os hubs de SignalR do ASP.NET Core](#)

# Páginas Razor do ASP.NET Core com EF Core – série de tutoriais

11/07/2018 • 2 minutes to read • [Edit Online](#)

Esta série de tutoriais ensina a criar aplicativos Web de Razor Pages do ASP.NET Core que usam o EF (Entity Framework) Core para o acesso a dados.

1. [Introdução](#)
2. [Operações Create, Read, Update e Delete](#)
3. [Classificação, filtragem, paginação e agrupamento](#)
4. [Migrações](#)
5. [Criar um modelo de dados complexo](#)
6. [Lendo dados relacionados](#)
7. [Atualizando dados relacionados](#)
8. [Tratar conflitos de simultaneidade](#)

# ASP.NET Core MVC com EF Core – série de tutoriais

21/06/2018 • 2 minutes to read • [Edit Online](#)

Este tutorial ensina a usar o ASP.NET Core MVC e o Entity Framework Core com controladores e exibições. As Páginas Razor é uma nova alternativa no ASP.NET Core 2.0, um modelo de programação baseado em página que torna a criação da interface do usuário da Web mais fácil e produtiva. É recomendável tentar o tutorial das [Páginas Razor](#) na versão do MVC. O tutorial Páginas do Razor:

- É mais fácil de acompanhar.
- Fornece mais práticas recomendadas do EF Core.
- Usa consultas mais eficientes.
- É mais atual com a API mais recente.
- Aborda mais recursos.

1. [Introdução](#)
2. [Operações Create, Read, Update e Delete](#)
3. [Classificação, filtragem, paginação e agrupamento](#)
4. [Migrações](#)
5. [Criar um modelo de dados complexo](#)
6. [Lendo dados relacionados](#)
7. [Atualizando dados relacionados](#)
8. [Tratar conflitos de simultaneidade](#)
9. [Herança](#)
10. [Tópicos avançados](#)

# Conceitos básicos do ASP.NET Core

09/01/2019 • 14 minutes to read • [Edit Online](#)

Um aplicativo ASP.NET Core é um aplicativo de console que cria um servidor Web em seu método `Program.Main`. O método `Main` é o *ponto de entrada gerenciado* do aplicativo:

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>();
}
```

O host do .NET Core:

- Carrega o [tempo de execução do .NET Core](#).
- Usa o primeiro argumento de linha de comando como o caminho para o binário gerenciado que contém o ponto de entrada (`Main`) e inicia a execução do código.

O método `Main` invoca `WebHost.CreateDefaultBuilder`, que segue o [padrão de construtor](#) para criar um host da Web. O construtor tem métodos que definem um servidor Web (por exemplo, `UseKestrel`) e a classe de inicialização (`UseStartup`). No exemplo anterior, o servidor Web `Kestrel` é alocado automaticamente. O host Web do ASP.NET Core tenta executar no [IIS \(Serviços de Informações da Internet\)](#), se disponível. Outros servidores Web como [HTTP.sys](#) podem ser usados ao chamar o método de extensão apropriado. `useStartup` é explicado em mais detalhes na seção [Inicialização](#).

`IWebHostBuilder`, o tipo de retorno da invocação de `WebHost.CreateDefaultBuilder`, fornece muitos métodos opcionais. Alguns desses métodos incluem `UseHttpSys` para hospedar o aplicativo em `HTTP.sys` e `UseContentRoot` para especificar o diretório de conteúdo raiz. Os métodos `Build` e `Run` compilam o objeto `IWebHost` que hospeda o aplicativo e começa a escutar solicitações HTTP.

```
public class Program
{
    public static void Main(string[] args)
    {
        var host = new WebHostBuilder()
            .UseKestrel()
            .UseStartup<Startup>()
            .Build();

        host.Run();
    }
}
```

O host do .NET Core:

- Carrega o [tempo de execução do .NET Core](#).
- Usa o primeiro argumento de linha de comando como o caminho para o binário gerenciado que contém o ponto de entrada (`Main`) e inicia a execução do código.

O método `Main` usa `WebHostBuilder`, que segue o [padrão de construtor](#) para criar um host de aplicativo Web. O construtor tem métodos que definem o servidor Web (por exemplo, `UseKestrel`) e a classe de inicialização (`UseStartup`). No exemplo anterior, o servidor Web `Kestrel` é usado. Outros servidores Web como `HTTP.sys` podem ser usados ao chamar o método de extensão apropriado. `UseStartup` é explicado em mais detalhes na seção [Inicialização](#).

O `WebHostBuilder` fornece muitos métodos opcionais, incluindo `UseIISIntegration`, para hospedagem no IIS e no IIS Express, e `UseContentRoot`, para especificar o diretório do conteúdo raiz. Os métodos `Build` e `Run` compilam o objeto `IWebHost` que hospeda o aplicativo e começa a escutar solicitações HTTP.

## Inicialização

O método `UseStartup` em `WebHostBuilder` especifica a classe `Startup` para seu aplicativo:

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>();
}
```

```
public class Program
{
    public static void Main(string[] args)
    {
        var host = new WebHostBuilder()
            .UseKestrel()
            .UseStartup<Startup>()
            .Build();

        host.Run();
    }
}
```

A classe `Startup` é onde os serviços exigidos pelo aplicativo são configurados e o pipeline de tratamento de solicitações é definido. A classe `Startup` deve ser pública e geralmente contém os seguintes métodos.

`Startup.ConfigureServices` é opcional.

```
public class Startup
{
    // This method gets called by the runtime. Use this method
    // to add services to the container.
    public void ConfigureServices(IServiceCollection services)
    {

    }

    // This method gets called by the runtime. Use this method
    // to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app)
    {
    }
}
```

```
public class Startup
{
    // This method gets called by the runtime. Use this method
    // to add services to the container.
    public void ConfigureServices(IServiceCollection services)
    {
    }

    // This method gets called by the runtime. Use this method
    // to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app)
    {
    }
}
```

[ConfigureServices](#) define os [Serviços](#) usados pelo seu aplicativo (por exemplo, ASP.NET Core MVC, Entity Framework Core, Identity etc.). [Configure](#) define o [middleware](#) chamado no pipeline de solicitação.

Para obter mais informações, consulte [Inicialização de aplicativo no ASP.NET Core](#).

## Raiz do conteúdo

A raiz do conteúdo é o caminho base para qualquer conteúdo usado pelo aplicativo, tal como [Razor Pages](#), exibições do MVC e ativos estáticos. Por padrão, a raiz do conteúdo é a mesma localização que o caminho base do aplicativo para o executável que hospeda o aplicativo.

## Diretório base (webroot)

O webroot de um aplicativo é o diretório do projeto que contém recursos públicos e estáticos como CSS, JavaScript e arquivos de imagem. Por padrão, `wwwroot` é o webroot.

Para arquivos (`.cshtml`) do Razor, o til-barra `~/` aponta para o webroot. Caminhos que começam com `~/` são denominados caminhos virtuais.

## Injeção de dependência (serviços)

Um [serviço](#) é um componente que é destinado ao consumo comum em um aplicativo. Os serviços são disponibilizados por meio de DI ([injeção de dependência](#)). O ASP.NET Core inclui um contêiner nativo de IoC (Inversão de Controle) que dá suporte à [injeção de construtor](#) por padrão. É possível substituir o contêiner padrão, se desejado. Além do [benefício de seu acoplamento flexível](#), a DI disponibiliza serviços, tais como [registro em log](#), por todo o seu aplicativo.

Para obter mais informações, consulte [Injeção de dependência no ASP.NET Core](#).

## Middleware

No ASP.NET Core, você compõe o pipeline de solicitação usando o [middleware](#). O middleware do ASP.NET Core executa operações assíncronas em um `HttpContext` e então invoca o próximo middleware no pipeline ou encerra a solicitação.

Por convenção, um componente de middleware chamado "XYZ" é adicionado ao pipeline invocando-se um método de extensão `UseXYZ` no método `Configure`.

O ASP.NET Core inclui um conjunto de middleware interno, e você pode escrever seu próprio middleware personalizado. A [OWIN \(Open Web Interface para .NET\)](#), que permite que aplicativos Web sejam desacoplados de servidores Web, é compatível com aplicativos ASP.NET Core.

Para obter mais informações, consulte [Middleware do ASP.NET Core e OWIN \(Open Web Interface para .NET\) com o ASP.NET Core](#).

## Iniciar solicitações HTTP

O [IHttpClientFactory](#) está disponível para acessar instâncias de [HttpClient](#) para fazer solicitações HTTP.

Para obter mais informações, consulte [Iniciar solicitações HTTP](#).

## Ambientes

Os ambientes, como *Desenvolvimento* e *Produção*, são uma noção de primeira classe no ASP.NET Core e podem ser definidos usando uma variável de ambiente, um arquivo de configurações e um argumento de linha de comando.

Para obter mais informações, consulte [Usar vários ambientes no ASP.NET Core](#).

## Hospedagem

Os aplicativos ASP.NET Core configuram e iniciam um *host*, que é responsável pelo gerenciamento de inicialização e de tempo de vida do aplicativo.

Para obter mais informações, consulte [Host da Web e Host Genérico no ASP.NET Core](#).

## Servidores

O modelo de hospedagem do ASP.NET Core não escuta diretamente as solicitações. O modelo de host se baseia em uma implementação do servidor HTTP para encaminhar a solicitação ao aplicativo.

- [Windows](#)
- [macOS](#)
- [Linux](#)

O ASP.NET Core vem com as seguintes implementações de servidor:

- O servidor [Kestrel](#) é um servidor Web gerenciado multiplataforma. O Kestrel normalmente é executado em uma configuração de proxy reverso que usa o [IIS](#). O Kestrel também pode ser executado como um servidor de borda voltado para o público exposto diretamente à Internet no ASP.NET Core 2.0 ou posterior.
  - O servidor HTTP do IIS (`IISHttpServer`) é um [servidor em processo](#) do IIS.
  - O servidor [HTTP.sys](#) é um servidor Web do ASP.NET Core no Windows.
- 
- [Windows](#)
  - [macOS](#)
  - [Linux](#)

O ASP.NET Core vem com as seguintes implementações de servidor:

- O servidor [Kestrel](#) é um servidor Web gerenciado multiplataforma. O Kestrel normalmente é executado em uma configuração de proxy reverso que usa o [IIS](#). O Kestrel também pode ser executado como um servidor de borda voltado para o público exposto diretamente à Internet no ASP.NET Core 2.0 ou posterior.
- O servidor [HTTP.sys](#) é um servidor Web do ASP.NET Core no Windows.

Para obter mais informações, consulte [Implementações de servidor Web em ASP.NET Core](#).

## Configuração

O ASP.NET Core usa um modelo de configuração com base nos pares de nome-valor. O modelo de configuração não baseado em [System.Configuration](#) ou em `web.config`. A configuração obtém as definições de um conjunto ordenado de provedores de configuração. Os provedores internos de configuração dão suporte a uma variedade de formatos de arquivo (XML, JSON, INI), variáveis de ambiente e argumentos de linha de comando. Você também pode escrever seus próprios provedores de configuração personalizados.

Para obter mais informações, consulte [Configuração no ASP.NET Core](#).

## Registro em log

O ASP.NET Core dá suporte a uma API de registro em log que funciona com uma variedade de provedores de logs. Os provedores internos dão suporte ao envio de logs para um ou mais destinos. As estruturas de registro em log de terceiros podem ser usadas.

Para obter mais informações, consulte [Registro em log no ASP.NET Core](#).

## Tratamento de erros

O ASP.NET Core tem cenários internos para tratamento de erros em aplicativos, incluindo uma página de exceção de desenvolvedor, páginas de erro personalizadas, páginas de código de status estático e tratamento de exceções de inicialização.

Para obter mais informações, consulte [Tratar erros no ASP.NET Core](#).

## Roteamento

O ASP.NET Core oferece cenários para roteamento de solicitações de aplicativo para manipuladores de rotas.

Para obter mais informações, consulte [Roteamento no ASP.NET Core](#).

## Tarefas em segundo plano

As tarefas em segundo plano são implementadas como *serviços hospedados*. Um serviço hospedado é uma classe com lógica de tarefa em segundo plano que implementa a interface [IHostedService](#).

Para obter mais informações, consulte [Tarefas em segundo plano com serviços hospedados no ASP.NET Core](#).

## Acessar o HttpContext

`HttpContext` está disponível automaticamente durante o processamento de solicitações com Razor Pages e com o MVC. Em circunstâncias em que `HttpContext` não está prontamente disponível, você pode acessar o `HttpContext` por meio da interface [IHttpContextAccessor](#) e sua implementação padrão, [HttpContextAccessor](#).

Para obter mais informações, consulte [Acessar o HttpContext no ASP.NET Core](#).

# Inicialização de aplicativo no ASP.NET Core

21/01/2019 • 12 minutes to read • [Edit Online](#)

Por Tom Dykstra, Luke Latham e Steve Smith

A classe `Startup` configura serviços e o pipeline de solicitação do aplicativo.

## A classe Startup

Os aplicativos do ASP.NET Core usam uma classe `Startup`, que é chamada de `Startup` por convenção. A classe `Startup`:

- Opcionalmente, inclua um método `ConfigureServices` para configurar os serviços do aplicativo. Um serviço é um componente reutilizável que fornece a funcionalidade do aplicativo. Os serviços estão configurados—também descritos como *registrados*—em `ConfigureServices` e consumidos no aplicativo por meio da [DI \(injeção de dependência\)](#) ou [ApplicationServices](#).
- Inclui um método `Configure` para criar o pipeline de processamento de solicitações do aplicativo.

`ConfigureServices` e `Configure` são chamados pelo tempo de execução quando o aplicativo é iniciado:

```
public class Startup
{
    // Use this method to add services to the container.
    public void ConfigureServices(IServiceCollection services)
    {
        ...
    }

    // Use this method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app)
    {
        ...
    }
}
```

A classe `Startup` é especificada para o aplicativo quando o [host](#) do aplicativo é criado. O host do aplicativo é compilado quando `Build` é chamado no construtor do host na classe `Program`. A classe `Startup` geralmente é especificada chamando o método `WebHostBuilderExtensions.UseStartup<TStartup>` no construtor do host:

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>();
}
```

O host fornece serviços que estão disponíveis para o construtor de classe `Startup`. O aplicativo adiciona serviços adicionais por meio de `ConfigureServices`. Os serviços de aplicativos e o host ficam, então, disponíveis em `Configure` e em todo o aplicativo.

Um uso comum da [injeção de dependência](#) na classe `Startup` é injetar:

- `IHostingEnvironment` para configurar serviços pelo ambiente.
- `IConfigurationBuilder` para ler a configuração.
- `ILoggerFactory` para criar um agente em `Startup.ConfigureServices`.

```
public class Startup
{
    private readonly IHostingEnvironment _env;
    private readonly IConfiguration _config;
    private readonly ILoggerFactory _loggerFactory;

    public Startup(IHostingEnvironment env, IConfiguration config,
                   ILoggerFactory loggerFactory)
    {
        _env = env;
        _config = config;
        _loggerFactory = loggerFactory;
    }

    public void ConfigureServices(IServiceCollection services)
    {
        var logger = _loggerFactory.CreateLogger<Startup>();

        if (_env.IsDevelopment())
        {
            // Development service configuration

            logger.LogInformation("Development environment");
        }
        else
        {
            // Non-development service configuration

            logger.LogInformation($"Environment: {_env.EnvironmentName}");
        }

        // Configuration is available during startup.
        // Examples:
        //   _config["key"]
        //   _config["subsection:suboption1"]
    }
}
```

Uma alternativa a injetar `IHostingEnvironment` é usar uma abordagem baseada em convenções. Quando o aplicativo define classes `Startup` separadas para ambientes diferentes (por exemplo, `StartupDevelopment`), a classe `Startup` apropriada é selecionada no tempo de execução. A classe cujo sufixo do nome corresponde ao ambiente atual é priorizada. Se o aplicativo for executado no ambiente de desenvolvimento e incluir uma classe `Startup` e uma classe `StartupDevelopment`, a classe `StartupDevelopment` será usada. Para obter mais informações, veja [Usar vários ambientes](#).

Para saber mais sobre o host, confira [Host da Web e Host Genérico no ASP.NET Core](#). Para obter informações sobre como tratar erros durante a inicialização, consulte [Tratamento de exceção na inicialização](#).

## O método ConfigureServices

O método `ConfigureServices` é:

- Opcional.
- Chamado pelo host antes do método `Configure` para configurar os serviços do aplicativo.
- Onde as [opções de configuração](#) são definidas por convenção.

O padrão típico consiste em chamar todos os métodos `Add{Service}` e, em seguida, chamar todos os métodos `services.Configure{Service}`. Por exemplo, veja o tópico [Configurar serviços de identidade](#).

O host pode configurar alguns serviços antes que métodos `Startup` sejam chamados. Para obter mais informações, consulte [Host da Web e Host Genérico no ASP.NET Core](#).

Para recursos que exigem uma configuração significativa, há métodos de extensão `Add{Service}` em [IServiceCollection](#). Um aplicativo ASP.NET Core típico registra serviços para o Entity Framework, Identity e MVC:

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddDbContext<ApplicationContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationContext>()
        .AddDefaultTokenProviders();

    services.AddMvc();

    // Add application services.
    services.AddTransient<IEmailSender, AuthMessageSender>();
    services.AddTransient<ISmsSender, AuthMessageSender>();
}
```

Adicionar serviços ao contêiner de serviços os torna disponíveis dentro do aplicativo e no método `Configure`. Os serviços são resolvidos por meio da [injeção de dependência](#) ou de [ApplicationServices](#).

## O método Configure

O método `Configure` é usado para especificar como o aplicativo responde às solicitações HTTP. O pipeline de solicitação é configurado adicionando componentes de [middleware](#) a uma instância de [IAApplicationBuilder](#).

`IAApplicationBuilder` está disponível para o método `Configure`, mas não é registrado no contêiner de serviço.

A hospedagem cria um `IAApplicationBuilder` e o passa diretamente para `Configure`.

Os [modelos do ASP.NET Core](#) configuram o pipeline com suporte para:

- [Página de exceção do desenvolvedor](#)
- [Manipulador de exceção](#)
- [Segurança de Transporte Estrita de HTTP \(HSTS\)](#)
- [Redirecionamento de HTTPS](#)
- [Arquivos estáticos](#)
- [RGPD \(Regulamento Geral sobre a Proteção de Dados\) da UE](#)
- [MVC do ASP.NET Core e Razor Pages](#)

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseCookiePolicy();

    app.UseMvc();
}
```

Cada método de extensão `Use` adiciona um ou mais componentes de middleware ao pipeline de solicitação. Por exemplo, o método de extensão `UseMvc` adiciona o [Middleware de roteamento](#) ao pipeline de solicitação e configura o [MVC](#) como o manipulador padrão.

Cada componente de middleware no pipeline de solicitação é responsável por invocar o próximo componente no pipeline ou causar um curto-circuito da cadeia, se apropriado. Se o curto-circuito não ocorrer ao longo da cadeia de middleware, cada middleware terá uma segunda chance de processar a solicitação antes que ela seja enviada ao cliente.

Serviços adicionais, como `IHostingEnvironment` e `ILoggerFactory`, também podem ser especificados na assinatura do método `Configure`. Quando especificados, serviços adicionais são injetados quando estão disponíveis.

Para obter mais informações de como usar o `IApplicationBuilder` e a ordem de processamento de middleware, confira [Middleware do ASP.NET Core](#).

## Métodos de conveniência

Para configurar serviços e o pipeline de processamento de solicitação sem usar uma classe `Startup`, chame os métodos de conveniência `ConfigureServices` e `Configure` no construtor do host. Diversas chamadas para `ConfigureServices` são acrescentadas umas às outras. Se houver várias chamadas de método `Configure`, a última chamada de `Configure` será usada.

```

public class Program
{
    public static IHostingEnvironment HostingEnvironment { get; set; }
    public static IConfiguration Configuration { get; set; }

    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .ConfigureAppConfiguration((hostingContext, config) =>
            {
                HostingEnvironment = hostingContext.HostingEnvironment;
                Configuration = config.Build();
            })
            .ConfigureServices(services =>
            {
                ...
            })
            .Configure(app =>
            {
                var loggerFactory = app.ApplicationServices
                    .GetRequiredService<ILoggerFactory>();
                var logger = loggerFactory.CreateLogger<Program>();

                logger.LogInformation("Logged in Configure");

                if (HostingEnvironment.IsDevelopment())
                {
                    ...
                }
                else
                {
                    ...
                }

                var configValue = Configuration["subsection:suboption1"];

                ...
            });
    }
}

```

## Estender a inicialização com filtros de inicialização

Use [IStartupFilter](#) para configurar o middleware no início ou no final do pipeline do middleware [Configure](#) de um aplicativo. [IStartupFilter](#) é útil para garantir que um middleware seja executado antes ou depois do middleware adicionado pelas bibliotecas no início ou no final do pipeline de processamento de solicitação do aplicativo.

[IStartupFilter](#) implementa um único método, [Configure](#), que recebe e retorna um [Action<IAApplicationBuilder>](#). Um [IAApplicationBuilder](#) define uma classe para configurar o pipeline de solicitação do aplicativo. Para obter mais informações, confira [Criar um pipeline de middleware com o IAApplicationBuilder](#).

Cada [IStartupFilter](#) implementa uma ou mais middlewares no pipeline de solicitação. Os filtros são invocados na ordem em que foram adicionados ao contêiner de serviço. Filtros podem adicionar middleware antes ou depois de passar o controle para o filtro seguinte, de modo que eles são acrescentados ao início ou no final do pipeline de aplicativo.

O exemplo a seguir demonstra como registrar um middleware com [IStartupFilter](#).

O middleware `RequestSetOptionsMiddleware` define um valor de opção de um parâmetro de cadeia de caracteres de consulta:

```
public class RequestSetOptionsMiddleware
{
    private readonly RequestDelegate _next;
    private IOptions<AppOptions> _injectedOptions;

    public RequestSetOptionsMiddleware(
        RequestDelegate next, IOptions<AppOptions> injectedOptions)
    {
        _next = next;
        _injectedOptions = injectedOptions;
    }

    public async Task Invoke(HttpContext httpContext)
    {
        Console.WriteLine("RequestSetOptionsMiddleware.Invoke");

        var option = httpContext.Request.Query["option"];

        if (!string.IsNullOrWhiteSpace(option))
        {
            _injectedOptions.Value.Option = WebUtility.HtmlEncode(option);
        }

        await _next(httpContext);
    }
}
```

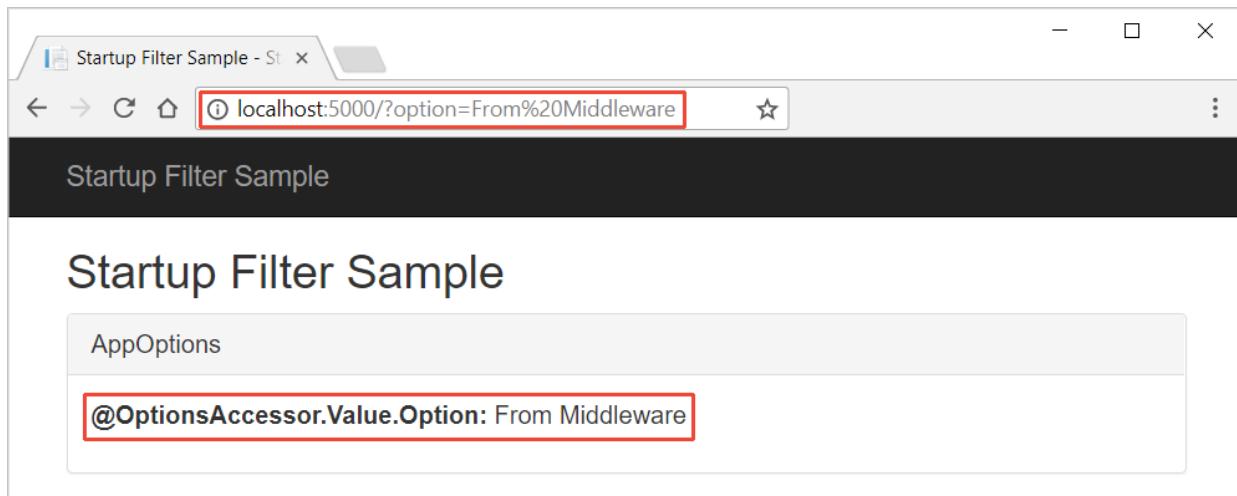
O `RequestSetOptionsMiddleware` é configurado na classe `RequestSetOptionsStartupFilter`:

```
public class RequestSetOptionsStartupFilter : IStartupFilter
{
    public Action<IApplicationBuilder> Configure(Action<IApplicationBuilder> next)
    {
        return builder =>
        {
            builder.UseMiddleware<RequestSetOptionsMiddleware>();
            next(builder);
        };
    }
}
```

O `IStartupFilter` é registrado no contêiner de serviço em `ConfigureServices` e aumenta `Startup` de fora da classe `Startup`:

```
WebHost.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        services.AddTransient<IStartupFilter,
            RequestSetOptionsStartupFilter>();
    })
    .UseStartup<Startup>()
    .Build();
```

Quando um parâmetro de cadeia de caracteres de consulta para `option` é fornecido, o middleware processa a atribuição de valor antes que o middleware do MVC renderize a resposta:



A ordem de execução do middleware é definida pela ordem dos registros `IStartupFilter`:

- Várias implementações de `IStartupFilter` podem interagir com os mesmos objetos. Se a ordem for importante, ordene seus registros de serviço `IStartupFilter` para corresponder à ordem em que os middlewares devem ser executados.
- Bibliotecas podem adicionar middleware com uma ou mais implementações de `IStartupFilter` que são executadas antes ou depois de outro middleware de aplicativo registrado com `IStartupFilter`. Para invocar um middleware `IStartupFilter` antes de um middleware adicionado pelo `IStartupFilter` de uma biblioteca, posicione o registro do serviço antes da biblioteca ser adicionada ao contêiner de serviço. Para invocá-lo posteriormente, posicione o registro do serviço após a biblioteca ser adicionada.

## Adicionar configuração na inicialização usando um assembly externo

Uma implementação `IHostingStartup` permite adicionar melhorias a um aplicativo durante a inicialização de um assembly externo fora da classe `Startup` do aplicativo. Para obter mais informações, consulte [Usar assemblies de inicialização de hospedagem no ASP.NET Core](#).

## Recursos adicionais

- [Host da Web e Host Genérico no ASP.NET Core](#)
- [Usar vários ambientes no ASP.NET Core](#)
- [Middleware do ASP.NET Core](#)
- [Registro em log no ASP.NET Core](#)
- [Configuração no ASP.NET Core](#)

# Injeção de dependência no ASP.NET Core

10/11/2018 • 31 minutes to read • [Edit Online](#)

Por [Steve Smith](#), [Scott Addie](#) e [Luke Latham](#)

O ASP.NET Core dá suporte ao padrão de design de software de DI (injeção de dependência), que é uma técnica para conseguir [IoC \(inversão de controle\)](#) entre classes e suas dependências.

Para obter mais informações específicas sobre injeção de dependência em controladores de MVC, consulte [Injeção de dependência em controladores no ASP.NET Core](#).

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Visão geral da injeção de dependência

Uma *dependência* é qualquer objeto exigido por outro objeto. Examine a classe `MyDependency` a seguir com um método `WriteMessage` do qual outras classes em um aplicativo dependem:

```
public class MyDependency
{
    public MyDependency()
    {
    }

    public Task WriteMessage(string message)
    {
        Console.WriteLine(
            $"MyDependency.WriteMessage called. Message: {message}");

        return Task.FromResult(0);
    }
}
```

Uma instância da classe `MyDependency` pode ser criada para tornar o método `WriteMessage` disponível para uma classe. A classe `MyDependency` é uma dependência da classe `IndexModel`:

```
public class IndexModel : PageModel
{
    MyDependency _dependency = new MyDependency();

    public async Task OnGetAsync()
    {
        await _dependency.WriteMessage(
            "IndexModel.OnGetAsync created this message.");
    }
}
```

Uma instância da classe `MyDependency` pode ser criada para tornar o método `WriteMessage` disponível para uma classe. A classe `MyDependency` é uma dependência da classe `HomeController`:

```

public class HomeController : Controller
{
    MyDependency _dependency = new MyDependency();

    public async Task<IActionResult> Index()
    {
        await _dependency.WriteMessage(
            "HomeController.Index created this message.");

        return View();
    }
}

```

A classe cria e depende diretamente da instância `MyDependency`. As dependências de código (como no exemplo anterior) são problemáticas e devem ser evitadas por estes motivos:

- Para substituir `MyDependency` por uma implementação diferente, a classe deve ser modificada.
- Se `MyDependency` tiver dependências, elas deverão ser configuradas pela classe. Em um projeto grande com várias classes dependendo da `MyDependency`, o código de configuração fica pulverizado por todo o aplicativo.
- É difícil testar a unidade dessa implementação. O aplicativo deve usar uma simulação ou stub da classe `MyDependency`, o que não é possível com essa abordagem.

Injeção de dependência trata desses problemas da seguinte maneira:

- Usando uma interface para abstrair a implementação da dependência.
- Registrando a dependência em um contêiner de serviço. O ASP.NET Core fornece um contêiner de serviço interno, o `IServiceProvider`. Os serviços são registrados no método `Startup.ConfigureServices` do aplicativo.
- *Injeção* do serviço no construtor da classe na qual ele é usado. A estrutura assume a responsabilidade de criar uma instância da dependência e de descartá-la quando não for mais necessária.

No [exemplo de aplicativo](#), a interface `IMyDependency` define um método que o serviço fornece ao aplicativo:

```

public interface IMyDependency
{
    Task WriteMessage(string message);
}

```

```

public interface IMyDependency
{
    Task WriteMessage(string message);
}

```

Essa interface é implementada por um tipo concreto, `MyDependency`:

```
public class MyDependency : IMyDependency
{
    private readonly ILogger<MyDependency> _logger;

    public MyDependency(ILogger<MyDependency> logger)
    {
        _logger = logger;
    }

    public Task WriteMessage(string message)
    {
        _logger.LogInformation(
            "MyDependency.WriteMessage called. Message: {MESSAGE}",
            message);

        return Task.FromResult(0);
    }
}
```

```
public class MyDependency : IMyDependency
{
    private readonly ILogger<MyDependency> _logger;

    public MyDependency(ILogger<MyDependency> logger)
    {
        _logger = logger;
    }

    public Task WriteMessage(string message)
    {
        _logger.LogInformation(
            "MyDependency.WriteMessage called. Message: {MESSAGE}",
            message);

        return Task.FromResult(0);
    }
}
```

`MyDependency` solicita um `ILogger<TCategoryName>` em seu construtor. Não é incomum usar a injeção de dependência de uma maneira encadeada. Por sua vez, cada dependência solicitada solicita suas próprias dependências. O contêiner resolve as dependências no grafo e retorna o serviço totalmente resolvido. O conjunto de dependências que precisa ser resolvido normalmente é chamado de *árvore de dependência*, *grafo de dependência* ou *grafo de objeto*.

`IMyDependency` e `ILogger<TCategoryName>` devem ser registrados no contêiner do serviço. `IMyDependency` está registrado em `Startup.ConfigureServices`. `ILogger<TCategoryName>` é registrado pela infraestrutura de abstrações de registro em log, portanto, é um [serviço fornecido por estrutura](#) registrado por padrão pela estrutura.

No exemplo de aplicativo, o serviço `IMyDependency` está registrado com o tipo concreto `MyDependency`. O registro define o escopo do tempo de vida do serviço para o tempo de vida de uma única solicitação. Descreveremos posteriormente neste tópico os [tempos de vida do serviço](#).

```

public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    services.AddScoped<IMyDependency, MyDependency>();
    services.AddTransient<IOperationTransient, Operation>();
    services.AddScoped<IOperationScoped, Operation>();
    services.AddSingleton<IOperationSingleton, Operation>();
    services.AddSingleton<IOperationSingletonInstance>(new Operation(Guid.Empty));

    // OperationService depends on each of the other Operation types.
    services.AddTransient<OperationService, OperationService>();
}

```

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    services.AddScoped<IMyDependency, MyDependency>();
    services.AddTransient<IOperationTransient, Operation>();
    services.AddScoped<IOperationScoped, Operation>();
    services.AddSingleton<IOperationSingleton, Operation>();
    services.AddSingleton<IOperationSingletonInstance>(new Operation(Guid.Empty));

    // OperationService depends on each of the other Operation types.
    services.AddTransient<OperationService, OperationService>();
}

```

#### NOTE

Cada método de extensão `services.Add{SERVICE_NAME}` adiciona (e possivelmente configura) serviços. Por exemplo, `services.AddMvc()` adiciona os serviços exigidos pelo MVC e por Razor Pages. Recomendamos que os aplicativos sigam essa convenção. Coloque os métodos de extensão no namespace [Microsoft.Extensions.DependencyInjection](#) para encapsular grupos de registros de serviço.

Se o construtor do serviço exigir um primitivo, como um `string`, o primitivo poderá ser injetado usando a [configuração](#) ou o [padrão de opções](#):

```

public class MyDependency : IMyDependency
{
    public MyDependency(IConfiguration config)
    {
        var myStringValue = config["MyStringKey"];

        // Use myStringValue
    }

    ...
}

```

Uma instância do serviço é solicitada por meio do construtor de uma classe, onde o serviço é usado e atribuído a um campo particular. O campo é usado para acessar o serviço

conforme o necessário na classe.

No exemplo de aplicativo, a instância `IMyDependency` é solicitada e usada para chamar o método `WriteMessage` do serviço:

```
public class IndexModel : PageModel
{
    private readonly IMyDependency _myDependency;

    public IndexModel(
        IMyDependency myDependency,
        OperationService operationService,
        IOperationTransient transientOperation,
        IOperationScoped scopedOperation,
        IOperationSingleton singletonOperation,
        IOperationSingletonInstance singletonInstanceOperation)
    {
        _myDependency = myDependency;
        OperationService = operationService;
        TransientOperation = transientOperation;
        ScopedOperation = scopedOperation;
        SingletonOperation = singletonOperation;
        SingletonInstanceOperation = singletonInstanceOperation;
    }

    public OperationService OperationService { get; }
    public IOperationTransient TransientOperation { get; }
    public IOperationScoped ScopedOperation { get; }
    public IOperationSingleton SingletonOperation { get; }
    public IOperationSingletonInstance SingletonInstanceOperation { get; }

    public async Task OnGetAsync()
    {
        await _myDependency.WriteMessage(
            "IndexModel.OnGetAsync created this message.");
    }
}
```

```
public class MyDependencyController : Controller
{
    private readonly IMyDependency _myDependency;

    public MyDependencyController(IMyDependency myDependency)
    {
        _myDependency = myDependency;
    }

    // GET: /mydependency/
    public async Task<IActionResult> Index()
    {
        await _myDependency.WriteMessage(
            "MyDependencyController.Index created this message.");

        return View();
    }
}
```

## Serviços fornecidos pela estrutura

O método `Startup.ConfigureServices` é responsável por definir os serviços usados pelo aplicativo, incluindo recursos de plataforma como o Entity Framework Core e o ASP.NET Core MVC. Inicialmente, a `IServiceCollection` fornecida ao `ConfigureServices` tem os

seguintes serviços definidos (dependendo de como o host foi configurado):

TIPO DE SERVIÇO	TEMPO DE VIDA
Microsoft.AspNetCore.Hosting.Builder.IApplicationBuilderFactory	Transitório
Microsoft.AspNetCore.Hosting.IApplicationLifetime	Singleton
Microsoft.AspNetCore.Hosting.IHostingEnvironment	Singleton
Microsoft.AspNetCore.Hosting.IStartup	Singleton
Microsoft.AspNetCore.Hosting.IStartupFilter	Transitório
Microsoft.AspNetCore.Hosting.Server.IServer	Singleton
Microsoft.AspNetCore.Http.IHttpContextFactory	Transitório
Microsoft.Extensions.Logging.ILogger<T>	Singleton
Microsoft.Extensions.Logging.ILoggerFactory	Singleton
Microsoft.Extensions.ObjectPool.ObjectPoolProvider	Singleton
Microsoft.Extensions.Options.IContainerOptions<T>	Transitório
Microsoft.Extensions.Options.IOptions<T>	Singleton
System.Diagnostics.DiagnosticSource	Singleton
System.Diagnostics.DiagnosticListener	Singleton

Quando um método de extensão de coleta do serviço estiver disponível para registrar um serviço (e seus serviços dependentes, se for necessário), a convenção é usar um único método de extensão `Add{SERVICE_NAME}` para registrar todos os serviços exigidos por esse serviço. O código a seguir é um exemplo de como adicionar outros serviços ao contêiner usando os métodos de extensão `AddDbContext`, `AddIdentity` e `AddMvc`:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationContext>()
        .AddDefaultTokenProviders();

    services.AddMvc();
}
```

Para saber mais, confira a [Classe ServiceCollection](#) na documentação da API.

# Tempos de vida do serviço

Escolha um tempo de vida apropriado para cada serviço registrado. Os serviços do ASP.NET Core podem ser configurados com os seguintes tempos de vida:

## Transitório

Os serviços com tempo de vida transitório são criados sempre que são solicitados. Esse tempo de vida funciona melhor para serviços leves e sem estado.

## Com escopo

Os serviços com tempo de vida com escopo são criados uma vez por solicitação.

### WARNING

Ao usar um serviço com escopo em um middleware, injete o serviço no método `Invoke` ou `InvokeAsync`. Não injete por meio de injeção de construtor porque isso força o serviço a se comportar como um singleton. Para obter mais informações, consulte [Middleware do ASP.NET Core](#).

## Singleton

Serviços de tempo de vida singleton são criados na primeira solicitação (ou quando `ConfigureServices` é executado e uma instância é especificada com o registro do serviço). Cada solicitação subsequente usa a mesma instância. Se o aplicativo exigir um comportamento de singleton, recomendamos permitir que o contêiner do serviço gerencie o tempo de vida do serviço. Não implemente o padrão de design singleton e forneça o código de usuário para gerenciar o tempo de vida do objeto na classe.

### WARNING

É perigoso resolver um serviço com escopo de um singleton. Pode fazer com que o serviço tenha um estado incorreto durante o processamento das solicitações seguintes.

## Comportamento da injeção de construtor

Os serviços podem ser resolvidos por dois mecanismos:

- `IServiceProvider`
- `ActivatorUtilities` – Permite a criação de objetos sem registro do serviço no contêiner de injeção de dependência. `ActivatorUtilities` é usado com abstrações voltadas ao usuário, como Auxiliares de Marca, controladores MVC e associadores de modelo.

Os construtores podem aceitar argumentos que não são fornecidos pela injeção de dependência, mas que precisam atribuir valores padrão.

Quando os serviços são resolvidos por `IServiceProvider` ou `ActivatorUtilities`, a injeção do construtor exige um construtor *público*.

Quando os serviços são resolvidos por `ActivatorUtilities`, a injeção de construtor exige a existência de apenas de um construtor aplicável. Há suporte para sobrecargas de construtor, mas somente uma sobrecarga pode existir, cujos argumentos podem ser todos atendidos pela injeção de dependência.

## Contextos de Entity Framework

Os contextos do Entity Framework devem ser adicionados ao contêiner de serviços usando o tempo de vida com escopo. Isso é tratado automaticamente com uma chamada para o método `AddDbContext` ao registrar o contexto do banco de dados. Os serviços que usam o contexto de banco de dados também devem usar o tempo de vida com escopo.

## Opções de tempo de vida e de registro

Para demonstrar a diferença entre as opções de tempo de vida e de registro, considere as interfaces a seguir, que representam tarefas como uma operação com um identificador exclusivo, `OperationId`. Dependendo de como o tempo de vida de um serviço de operações é configurado para as interfaces a seguir, o contêiner fornece a mesma instância do serviço, ou outra diferente, quando recebe uma solicitação de uma classe:

```
public interface IOperation
{
    Guid OperationId { get; }
}

public interface IOperationTransient : IOperation
{
}

public interface IOperationScoped : IOperation
{
}

public interface IOperationSingleton : IOperation
{
}

public interface IOperationSingletonInstance : IOperation
{
}
```

```
public interface IOperation
{
    Guid OperationId { get; }
}

public interface IOperationTransient : IOperation
{
}

public interface IOperationScoped : IOperation
{
}

public interface IOperationSingleton : IOperation
{
}

public interface IOperationSingletonInstance : IOperation
{
}
```

As interfaces são implementadas na classe `Operation`. O construtor `Operation` gera um GUID, caso um não seja fornecido:

```

public class Operation : IOperationTransient,
    IOperationScoped,
    IOperationSingleton,
    IOperationSingletonInstance
{
    public Operation() : this(Guid.NewGuid())
    {

    }

    public Operation(Guid id)
    {
        OperationId = id;
    }

    public Guid OperationId { get; private set; }
}

```

```

public class Operation : IOperationTransient,
    IOperationScoped,
    IOperationSingleton,
    IOperationSingletonInstance
{
    public Operation() : this(Guid.NewGuid())
    {

    }

    public Operation(Guid id)
    {
        OperationId = id;
    }

    public Guid OperationId { get; private set; }
}

```

Há um `OperationService` registrado que depende de cada um dos outros `Operation` tipos. Quando `OperationService` é solicitado por meio da injeção de dependência, ele recebe a uma nova instância de cada serviço, ou uma instância existente com base no tempo de vida do serviço dependente.

- Se os serviços temporários forem criados mediante solicitação, o `OperationId` do serviço `IOperationTransient` será diferente do `OperationId` do `OperationService`. `OperationService` recebe uma nova instância da classe `IOperationTransient`. A nova instância produz outro `OperationId`.
- Se os serviços com escopo forem criados por solicitação, o `OperationId` do serviço `IOperationScoped` será o mesmo de `OperationService` dentro de uma solicitação. Em todas as solicitações, os dois serviços compartilham um valor de `OperationId` diferente.
- Se os serviços singleton e de instância singleton forem criados uma vez e usados em todas as solicitações e em todos os serviços, o `OperationId` será constante entre todas as solicitações de serviço.

```

public class OperationService
{
    public OperationService(
        IOperationTransient transientOperation,
        IOperationScoped scopedOperation,
        IOperationSingleton singletonOperation,
        IOperationSingletonInstance instanceOperation)
    {
        TransientOperation = transientOperation;
        ScopedOperation = scopedOperation;
        SingletonOperation = singletonOperation;
        SingletonInstanceOperation = instanceOperation;
    }

    public IOperationTransient TransientOperation { get; }
    public IOperationScoped ScopedOperation { get; }
    public IOperationSingleton SingletonOperation { get; }
    public IOperationSingletonInstance SingletonInstanceOperation { get; }
}

```

```

public class OperationService
{
    public OperationService(IOperationTransient transientOperation,
        IOperationScoped scopedOperation,
        IOperationSingleton singletonOperation,
        IOperationSingletonInstance instanceOperation)
    {
        TransientOperation = transientOperation;
        ScopedOperation = scopedOperation;
        SingletonOperation = singletonOperation;
        SingletonInstanceOperation = instanceOperation;
    }

    public IOperationTransient TransientOperation { get; }
    public IOperationScoped ScopedOperation { get; }
    public IOperationSingleton SingletonOperation { get; }
    public IOperationSingletonInstance SingletonInstanceOperation { get; }
}

```

Em `Startup.ConfigureServices`, cada tipo é adicionado ao contêiner de acordo com seu tempo de vida nomeado:

```

public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    services.AddScoped<IMyDependency, MyDependency>();
    services.AddTransient<IOperationTransient, Operation>();
    services.AddScoped<IOperationScoped, Operation>();
    services.AddSingleton<IOperationSingleton, Operation>();
    services.AddSingleton<IOperationSingletonInstance>(new Operation(Guid.Empty));

    // OperationService depends on each of the other Operation types.
    services.AddTransient<OperationService, OperationService>();
}

```

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    services.AddScoped<IMyDependency, MyDependency>();
    services.AddTransient<IOperationTransient, Operation>();
    services.AddScoped<IOperationScoped, Operation>();
    services.AddSingleton<IOperationSingleton, Operation>();
    services.AddSingleton<IOperationSingletonInstance>(new Operation(Guid.Empty));

    // OperationService depends on each of the other Operation types.
    services.AddTransient<OperationService, OperationService>();
}

```

O serviço `IOperationSingletonInstance` está usando uma instância específica com uma ID conhecida de `Guid.Empty`. Fica claro quando esse tipo está em uso (seu GUID contém apenas zeros).

O exemplo de aplicativo demonstra os tempos de vida do objeto dentro e entre solicitações individuais. O exemplo de aplicativo `IndexModel` solicita cada tipo `IOperation` e o `OperationService`. Depois, a página exibe todos os valores de `OperationId` da classe de modelo de página e do serviço por meio de atribuições de propriedade:

```

public class IndexModel : PageModel
{
    private readonly IMyDependency _myDependency;

    public IndexModel(
        IMyDependency myDependency,
        OperationService operationService,
        IOperationTransient transientOperation,
        IOperationScoped scopedOperation,
        IOperationSingleton singletonOperation,
        IOperationSingletonInstance singletonInstanceOperation)
    {
        _myDependency = myDependency;
        OperationService = operationService;
        TransientOperation = transientOperation;
        ScopedOperation = scopedOperation;
        SingletonOperation = singletonOperation;
        SingletonInstanceOperation = singletonInstanceOperation;
    }

    public OperationService OperationService { get; }
    public IOperationTransient TransientOperation { get; }
    public IOperationScoped ScopedOperation { get; }
    public IOperationSingleton SingletonOperation { get; }
    public IOperationSingletonInstance SingletonInstanceOperation { get; }

    public async Task OnGetAsync()
    {
        await _myDependency.WriteMessage(
            "IndexModel.OnGetAsync created this message.");
    }
}

```

O exemplo de aplicativo demonstra os tempos de vida do objeto dentro e entre solicitações individuais. O exemplo de aplicativo inclui um `OperationsController` que solicita cada tipo de `IOperation` e o `OperationService`. A ação `Index` define os serviços no `ViewBag` para exibição dos valores `OperationId` do serviço:

```

public class OperationsController : Controller
{
    private readonly OperationService _operationService;
    private readonly IOperationTransient _transientOperation;
    private readonly IOperationScoped _scopedOperation;
    private readonly IOperationSingleton _singletonOperation;
    private readonly IOperationSingletonInstance _singletonInstanceOperation;

    public OperationsController(
        OperationService operationService,
        IOperationTransient transientOperation,
        IOperationScoped scopedOperation,
        IOperationSingleton singletonOperation,
        IOperationSingletonInstance singletonInstanceOperation)
    {
        _operationService = operationService;
        _transientOperation = transientOperation;
        _scopedOperation = scopedOperation;
        _singletonOperation = singletonOperation;
        _singletonInstanceOperation = singletonInstanceOperation;
    }

    public IActionResult Index()
    {
        // ViewBag contains controller-requested services.
        ViewBag.Transient = _transientOperation;
        ViewBag.Scoped = _scopedOperation;
        ViewBag.Singleton = _singletonOperation;
        ViewBag.SingletonInstance = _singletonInstanceOperation;

        // Operation service has its own requested services.
        ViewBag.Service = _operationService;

        return View();
    }
}

```

As duas saídas a seguir mostram os resultados das duas solicitações:

#### **Primeira solicitação:**

Operações do controlador:

Transitória: d233e165-f417-469b-a866-1cf1935d2518  
 Com escopo: 5d997e2d-55f5-4a64-8388-51c4e3a1ad19  
 Singleton: 01271bc1-9e31-48e7-8f7c-7261b040ded9  
 Instância: 00000000-0000-0000-0000-000000000000

`OperationService` operações:

Transitória: c6b049eb-1318-4e31-90f1-eb2dd849ff64  
 Com escopo: 5d997e2d-55f5-4a64-8388-51c4e3a1ad19  
 Singleton: 01271bc1-9e31-48e7-8f7c-7261b040ded9  
 Instância: 00000000-0000-0000-0000-000000000000

#### **Segunda solicitação:**

Operações do controlador:

Transitória: b63bd538-0a37-4ff1-90ba-081c5138dda0  
 Com escopo: 31e820c5-4834-4d22-83fc-a60118acb9f4  
 Singleton: 01271bc1-9e31-48e7-8f7c-7261b040ded9  
 Instância: 00000000-0000-0000-0000-000000000000

`OperationService` operações:

Transitória: c4cbacb8-36a2-436d-81c8-8c1b78808aaf  
Com escopo: 31e820c5-4834-4d22-83fc-a60118acb9f4  
Singleton: 01271bc1-9e31-48e7-8f7c-7261b040ded9  
Instância: 00000000-0000-0000-0000-000000000000

Observe qual dos valores de `OperationId` varia em uma solicitação, e entre as solicitações:

- Os objetos *transitórios* sempre são diferentes. Observe que o valor `OperationId` transitório da primeira e da segunda solicitações é diferente para as duas operações `OperationService` e em todas as solicitações. Uma nova instância é fornecida para cada solicitação e serviço.
- Os objetos *com escopo* são os mesmos em uma solicitação, mas diferentes entre solicitações.
- Os objetos *singleton* são os mesmos para cada objeto e solicitação, independentemente de uma instância `Operation` ser fornecida em `ConfigureServices`.

## Chamar os serviços desde o principal

Crie um `IServiceScope` com `IServiceScopeFactory.CreateScope` para resolver um serviço com escopo dentro do escopo do aplicativo. Essa abordagem é útil para acessar um serviço com escopo durante a inicialização para executar tarefas de inicialização. O exemplo a seguir mostra como obter um contexto para o `MyScopedService` em `Program.Main`:

```
public static void Main(string[] args)
{
    var host = CreateWebHostBuilder(args).Build();

    using (var serviceScope = host.Services.CreateScope())
    {
        var services = serviceScope.ServiceProvider;

        try
        {
            var serviceContext = services.GetRequiredService<MyScopedService>();
            // Use the context here
        }
        catch (Exception ex)
        {
            var logger = services.GetRequiredService<ILogger<Program>>();
            logger.LogError(ex, "An error occurred.");
        }
    }

    host.Run();
}
```

## Validação de escopo

Quando o aplicativo está em execução no ambiente de desenvolvimento, o provedor de serviço padrão executa verificações para saber se:

- Os serviços com escopo não são resolvidos direta ou indiretamente pelo provedor de serviço raiz.
- Os serviços com escopo não são injetados direta ou indiretamente em singletons.

O provedor de serviços raiz é criado quando `BuildServiceProvider` é chamado. O tempo de

vida do provedor de serviço raiz corresponde ao tempo de vida do aplicativo/servidor quando o provedor começa com o aplicativo e é descartado quando o aplicativo é desligado.

Os serviços com escopo são descartados pelo contêiner que os criou. Se um serviço com escopo é criado no contêiner raiz, o tempo de vida do serviço é promovido efetivamente para singleton, porque ele só é descartado pelo contêiner raiz quando o aplicativo/servidor é desligado. A validação dos escopos de serviço detecta essas situações quando `BuildServiceProvider` é chamado.

Para obter mais informações, consulte [Host da Web do ASP.NET Core](#).

## Serviços de solicitação

Os serviços disponíveis em uma solicitação do ASP.NET de `HttpContext` são expostos por meio da coleção `HttpContext.RequestServices`.

Os Serviços de Solicitação representam os serviços configurados e solicitados como parte do aplicativo. Quando os objetos especificam dependências, elas são atendidas pelos tipos encontrados em `RequestServices`, não em `ApplicationServices`.

Em geral, o aplicativo não deve usar essas propriedades diretamente. Em vez disso, solicite os tipos exigidos pelas classes por meio de construtores de classe e permita que a estrutura injete as dependências. Isso resulta em classes que são mais fáceis de testar.

### NOTE

Prefira solicitar dependências como parâmetros de construtor para acessar a coleção `RequestServices`.

## Projetar serviços para injeção de dependência

As melhores práticas são:

- Projetar serviços para usar a injeção de dependência a fim de obter suas dependências.
- Evite chamadas de método estático com estado (uma prática conhecida como [adesão estática](#)).
- Evite a instanciação direta das classes dependentes em serviços. A instanciação direta acopla o código a uma implementação específica.

Seguindo os [Princípios SOLID de Design Orientado a Objeto](#), as classes de aplicativo naturalmente tendem a ser pequenas, bem fatoradas e facilmente testadas.

Se parecer que uma classe tem muitas dependências injetadas, normalmente é um sinal de que a classe tem muitas responsabilidades e está violando o [Princípio da Responsabilidade Única \(SRP\)](#). Tente refatorar a classe movendo algumas de suas responsabilidades para uma nova classe. Lembre-se de que as classes de modelo de página de Razor Pages e as classes do controlador de MVC devem se concentrar em questões de interface do usuário. Os detalhes de implementação de acesso aos dados e as regras de negócios devem ser mantidos em classes apropriadas a esses [interesses separados](#).

### Descarte de serviços

O contêiner chama `Dispose` para os tipos `IDisposable` criados por ele. Se uma instância for adicionada ao contêiner pelo código do usuário, ele não será descartado automaticamente.

```

// Services that implement IDisposable:
public class Service1 : IDisposable {}
public class Service2 : IDisposable {}
public class Service3 : IDisposable {}

public interface ISomeService {}
public class SomeServiceImplementation : ISomeService, IDisposable {}

public void ConfigureServices(IServiceCollection services)
{
    // The container creates the following instances and disposes them automatically:
    services.AddScoped<Service1>();
    services.AddSingleton<Service2>();
    services.AddSingleton<ISomeService>(sp => new SomeServiceImplementation());

    // The container doesn't create the following instances, so it doesn't dispose of
    // the instances automatically:
    services.AddSingleton<Service3>(new Service3());
    services.AddSingleton(new Service3());
}

```

#### NOTE

No ASP.NET Core 1.0, as chamadas do contêiner descartam *todos* os objetos `IDisposable`, incluindo aqueles que ele não criou.

## Substituição do contêiner de serviço padrão

O contêiner de serviço interno deve atender às necessidades da estrutura e da maioria dos aplicativos do consumidor. É recomendado usar o contêiner interno, a menos que você precise de um recurso específico que não seja compatível com ele. Alguns dos recursos compatíveis com contêineres de terceiros não encontrados no contêiner interno:

- Injeção de propriedade
- Injeção com base no nome
- Contêineres filho
- Gerenciamento de tempo de vida personalizado
- Suporte de `Func<T>` para inicialização lenta

Confira o [arquivo readme.md de injeção de dependência](#) para obter uma lista de alguns dos contêineres que dão suporte a adaptadores.

O exemplo a seguir substitui o contêiner interno por [Autofac](#):

- Instale os pacotes de contêiner apropriados:
  - [Autofac](#)
  - [Autofac.Extensions.DependencyInjection](#)
- Configure o contêiner no `Startup.ConfigureServices` e retorne um `IServiceProvider`:

```

public IServiceProvider ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    // Add other framework services

    // Add Autofac
    var containerBuilder = new ContainerBuilder();
    containerBuilder.RegisterModule<DefaultModule>();
    containerBuilder.Populate(services);
    var container = containerBuilder.Build();
    return new AutofacServiceProvider(container);
}

```

Para usar um contêiner de terceiros, `Startup.ConfigureServices` deve retornar `IServiceProvider`.

- Configure o Autofac em `DefaultModule`:

```

public class DefaultModule : Module
{
    protected override void Load(ContainerBuilder builder)
    {
        builder.RegisterType<CharacterRepository>().As<ICharacterRepository>();
    }
}

```

Em tempo de execução, o Autofac é usado para resolver tipos e injetar dependências. Para saber mais sobre como usar o Autofac com o ASP.NET Core, consulte a [documentação do Autofac](#).

### Acesso thread-safe

Os serviços Singleton precisam ser thread-safe. Se um serviço singleton tiver uma dependência em um serviço transitório, o serviço transitório também precisará ser thread-safe, dependendo de como ele é usado pelo singleton.

O método de fábrica de um único serviço, como o segundo argumento para `AddSingleton<TService>(IServiceCollection, Func<IServiceProvider,TService>)`, não precisa ser thread-safe. Como um construtor do tipo (`static`), ele tem garantia de ser chamado uma vez por um único thread.

## Recomendações

- A resolução de serviço baseada em `async/await` e `Task` não é compatível. O C# não é compatível com os construtores assíncronos. Portanto, o padrão recomendado é usar os métodos assíncronos após a resolução síncrona do serviço.
- Evite armazenar dados e a configuração diretamente no contêiner do serviço. Por exemplo, o carrinho de compras de um usuário normalmente não deve ser adicionado ao contêiner do serviço. A configuração deve usar o [padrão de opções](#). Da mesma forma, evite objetos de "suporte de dados" que existem somente para permitir o acesso a outro objeto. É melhor solicitar o item real por meio da DI.
- Evite o acesso estático aos serviços (por exemplo, digitando estaticamente `IApplicationBuilder.ApplicationServices` para usar em outro lugar).
- Evite usar o *padrão do localizador de serviço*. Por exemplo, não invoque `GetService` para obter uma instância de serviço quando for possível usar a DI. Outra variação de

localizador de serviço a ser evitada é injetar um alocador que resolve as dependências em tempo de execução. Essas duas práticas misturam estratégias de [inversão de controle](#).

- Evite o acesso estático a `HttpContext` (por exemplo, `IHttpContextAccessor.HttpContext`).

Como todos os conjuntos de recomendações, talvez você encontre situações em que é necessário ignorar uma recomendação. As exceções são raras —principalmente casos especiais dentro da própria estrutura.

A DI é uma *alternativa* aos padrões de acesso a objeto estático/global. Talvez você não obtenha os benefícios da DI se combiná-lo com o acesso a objeto estático.

## Recursos adicionais

- [Injeção de dependência em exibições no ASP.NET Core](#)
- [Injeção de dependência em controladores no ASP.NET Core](#)
- [Injeção de dependência em manipuladores de requisito no ASP.NET Core](#)
- [Inicialização de aplicativo no ASP.NET Core](#)
- [Ativação de middleware baseada em alocador no ASP.NET Core](#)
- [Como escrever um código limpo no ASP.NET Core com injeção de dependência \(MSDN\)](#)
- [Design de aplicativo gerenciado por contêiner, prelúdio: a que local o contêiner pertence?](#)
- [Princípio de Dependências Explícitas](#)
- [Inversão de Contêineres de Controle e o padrão de Injeção de Dependência \(Martin Fowler\)](#)
- [New is Glue \("colando" o código em uma implementação específica\)](#)
- [Como registrar um serviço com várias interfaces na DI do ASP.NET Core](#)

# Roteamento no ASP.NET Core

30/01/2019 • 71 minutes to read • [Edit Online](#)

Por [Ryan Nowak](#), [Steve Smith](#) e [Rick Anderson](#)

Para obter a versão 1.1 deste tópico, baixe [Roteamento no ASP.NET Core \(versão 1.1, PDF\)](#).

O roteamento é responsável por mapear URLs de solicitação para seletores de ponto de extremidade e expedir solicitações de entrada para pontos de extremidade. As rotas são definidas no aplicativo e configuradas quando o aplicativo é iniciado. Uma rota pode opcionalmente extrair os valores da URL contida na solicitação e esses valores podem então ser usados para o processamento da solicitação. Usando as informações de rota do aplicativo, o roteamento também pode gerar URLs que são mapeadas para seletores de ponto de extremidade.

Para usar os últimos cenários de roteamento no ASP.NET Core 2.2, especifique a [versão de compatibilidade](#) com o registro de serviços MVC em `Startup.ConfigureServices`:

```
services.AddMvc()
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
```

A opção `EnableEndpointRouting` determina se o roteamento deve usar internamente a lógica baseada em ponto de extremidade ou a lógica baseada em [IRouter](#) do ASP.NET Core 2.1 ou anterior. Quando a versão de compatibilidade é definida como 2.2 ou posterior, o valor padrão é `true`. Defina o valor como `false` para usar a lógica de roteamento anterior:

```
// Use the routing logic of ASP.NET Core 2.1 or earlier:
services.AddMvc(options => options.EnableEndpointRouting = false)
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
```

Para obter mais informações sobre o roteamento baseado em [IRouter](#), confira a [versão do ASP.NET Core 2.1](#) deste tópico.

O roteamento é responsável por mapear URLs de solicitação para manipuladores de rotas e expedir as solicitações de entrada. As rotas são definidas no aplicativo e configuradas quando o aplicativo é iniciado. Uma rota pode opcionalmente extrair os valores da URL contida na solicitação e esses valores podem então ser usados para o processamento da solicitação. Usando rotas configuradas no aplicativo, o roteamento pode gerar URLs que são mapeadas para manipuladores de rotas.

Para usar os últimos cenários de roteamento no ASP.NET Core 2.1, especifique a [versão de compatibilidade](#) com o registro de serviços MVC em `Startup.ConfigureServices`:

```
services.AddMvc()
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
```

## IMPORTANT

Este documento aborda o roteamento de nível inferior do ASP.NET Core. Para obter informações sobre o roteamento do ASP.NET Core MVC, confira [Roteamento para ações do controlador no ASP.NET Core](#). Para obter informações sobre convenções de roteamento no Razor Pages, confira [Convenções de rota e aplicativo das Páginas do Razor no ASP.NET Core](#).

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Conceitos básicos sobre roteamento

A maioria dos aplicativos deve escolher um esquema de roteamento básico e descriptivo para que as URLs sejam legíveis e significativas. A rota convencional padrão `{controller=Home}/{action=Index}/{id?}` :

- Dá suporte a um esquema de roteamento básico e descriptivo.
- É um ponto de partida útil para aplicativos baseados em interface do usuário.

Os desenvolvedores geralmente adicionam outras rotas concisas às áreas de alto tráfego de um aplicativo em situações especiais (por exemplo, pontos de extremidade de blog e comércio eletrônico) usando o [roteamento de atributo](#) ou rotas convencionais dedicadas.

As APIs da Web devem usar o roteamento de atributo para modelar a funcionalidade do aplicativo como um conjunto de recursos em que as operações são representadas por verbos HTTP. Isso significa que muitas operações (por exemplo, GET, POST) no mesmo recurso lógico usarão a mesma URL. O roteamento de atributo fornece um nível de controle necessário para projetar cuidadosamente o layout de ponto de extremidade público de uma API.

Os aplicativos do Razor Pages usam o roteamento convencional padrão para fornecer recursos nomeados na pasta *Pages* de um aplicativo. Estão disponíveis convenções adicionais que permitem a personalização do comportamento de roteamento do Razor Pages. Para obter mais informações, consulte [Introdução a Páginas do Razor no ASP.NET Core](#) e [Convenções de rota e aplicativo das Páginas do Razor no ASP.NET Core](#).

O suporte à geração de URL permite que o aplicativo seja desenvolvido sem hard-coding das URLs para vincular o aplicativo. Esse suporte permite começar com uma configuração de roteamento básica e modificar as rotas, depois que o layout de recurso do aplicativo é determinado.

O roteamento usa *pontos de extremidade* (`Endpoint`) para representar os pontos de extremidade lógicos em um aplicativo.

Um ponto de extremidade define um delegado para processar solicitações e uma coleção de metadados arbitrários. Os metadados usados implementam interesses paralelos com base em políticas e na configuração anexada a cada ponto de extremidade.

O sistema de roteamento tem as seguintes características:

- A sintaxe do modelo de rota é usada para definir rotas com os parâmetros de rota com tokens criados.
- A configuração de ponto de extremidade de estilo convencional e de estilo de atributo é permitida.
- `IRouteConstraint` é usado para determinar se um parâmetro de URL contém um valor válido para determinada restrição de ponto de extremidade.
- Modelos de aplicativo, como MVC/Razor Pages, registram todos os seus pontos de extremidade, que têm uma implementação previsível de cenários de roteamento.
- A implementação de roteamento toma decisões de roteamento, sempre que desejado no pipeline de middleware.

- O Middleware que aparece após um Middleware de Roteamento pode inspecionar o resultado da decisão de ponto de extremidade do Middleware de Roteamento para determinado URI de solicitação.
- É possível enumerar todos os pontos de extremidade no aplicativo em qualquer lugar do pipeline de middleware.
- Um aplicativo pode usar o roteamento para gerar URLs (por exemplo, para redirecionamento ou links) com base nas informações de ponto de extremidade e evitar URLs embutidas em código, o que ajuda na facilidade de manutenção.
- A geração de URL baseia-se em endereços, que dá suporte à extensibilidade arbitrária:
  - A API de Gerador de Link (`LinkGenerator`) pode ser resolvida em qualquer lugar usando a [DI \(Injeção de Dependência\)](#) para gerar URLs.
  - Quando a API de Gerador de Link não está disponível por meio da DI, `IUrlHelper` oferece métodos para criar URLs.

#### NOTE

Com o lançamento do roteamento de ponto de extremidade no ASP.NET Core 2.2, a vinculação de ponto de extremidade fica limitada às ações e às páginas do MVC/Razor Pages. As expansões de funcionalidades de vinculação de ponto de extremidade estão planejadas para versões futuras.

O roteamento usa *rotas* (implementações de `IRouter`) para:

- Mapear solicitações de entrada para *manipuladores de rotas*.
- Gerar as URLs usadas nas respostas.

Por padrão, um aplicativo tem uma única coleção de rotas. Quando uma solicitação é recebida, as rotas na coleção são processadas na ordem em que existem na coleção. A estrutura tenta corresponder uma URL de solicitação de entrada a uma rota na coleção chamando o método `RouteAsync` em cada rota da coleção. Uma resposta pode usar o roteamento para gerar URLs (por exemplo, para redirecionamento ou links) com base nas informações de rotas e evitar URLs embutidas em código, o que ajuda na facilidade de manutenção.

O sistema de roteamento tem as seguintes características:

- A sintaxe do modelo de rota é usada para definir rotas com os parâmetros de rota com tokens criados.
- A configuração de ponto de extremidade de estilo convencional e de estilo de atributo é permitida.
- `IRouteConstraint` é usado para determinar se um parâmetro de URL contém um valor válido para determinada restrição de ponto de extremidade.
- Modelos de aplicativo, como MVC/Razor Pages, registram todas as suas rotas, que têm uma implementação previsível de cenários de roteamento.
- Uma resposta pode usar o roteamento para gerar URLs (por exemplo, para redirecionamento ou links) com base nas informações de rotas e evitar URLs embutidas em código, o que ajuda na facilidade de manutenção.
- A geração de URL baseia-se em rotas, que dá suporte à extensibilidade arbitrária. `IUrlHelper` oferece métodos para criar URLs.

O roteamento está conectado ao pipeline do [middleware](#) pela classe `RouterMiddleware`. O [ASP.NET Core MVC](#) adiciona o roteamento ao pipeline de middleware como parte de sua configuração e manipula o roteamento nos aplicativos do MVC e do Razor Pages. Para saber como usar o roteamento como um componente autônomo, confira a seção [Usar o Middleware de Roteamento](#).

## Correspondência de URL

A correspondência de URL é o processo pelo qual o roteamento expede uma solicitação de entrada para um *ponto de extremidade*. Esse processo se baseia nos dados do caminho da URL, mas pode ser estendido para considerar qualquer dado na solicitação. A capacidade de expedir solicitações para manipuladores separados é fundamental para dimensionar o tamanho e a complexidade de um aplicativo.

O sistema de roteamento no roteamento de ponto de extremidade é responsável por todas as decisões de expedição. Como o middleware aplica políticas com base no ponto de extremidade selecionado, é importante que qualquer decisão que possa afetar a expedição ou a aplicação de políticas de segurança seja feita dentro do sistema de roteamento.

Quando o delegado do ponto de extremidade é executado, as propriedades de `RouteContext.RouteData` são definidas com valores apropriados com base no processamento da solicitação executado até o momento.

Correspondência de URL é o processo pelo qual o roteamento expede uma solicitação de entrada para um *manipulador*. Esse processo se baseia nos dados do caminho da URL, mas pode ser estendido para considerar qualquer dado na solicitação. A capacidade de expedir solicitações para manipuladores separados é fundamental para dimensionar o tamanho e a complexidade de um aplicativo.

As solicitações de entrada entram no `RouterMiddleware`, que chama o método `RouteAsync` em cada rota na sequência. A instância `IRouter` escolhe se deseja *manipular* a solicitação definindo o `RouteContext.Handler` como um `RequestDelegate` não nulo. Se uma rota definir um manipulador para a solicitação, o processamento de rotas será interrompido e o manipulador será invocado para processar a solicitação. Se nenhum manipulador de rotas é encontrado para processar a solicitação, o middleware transmite a solicitação para o próximo middleware no pipeline de solicitação.

A entrada primária para `RouteAsync` é o `RouteContext.HttpContext` associado à solicitação atual. O `RouteContext.Handler` e o `RouteContext.RouteData` são as saídas definidas depois que é encontrada uma correspondência de uma rota.

Uma correspondência que chama `RouteAsync` também define as propriedades do `RouteContext.RouteData` com valores apropriados com base no processamento da solicitação executado até o momento.

`RouteData.Values` é um dicionário de *valores de rota* produzido por meio da rota. Esses valores geralmente são determinados pela criação de token da URL e podem ser usados para aceitar a entrada do usuário ou tomar outras decisões de expedição dentro do aplicativo.

`RouteData.DataTokens` é um recipiente de propriedades de dados adicionais relacionados à rota correspondente. `DataTokens` são fornecidos para dar suporte à associação de dados de estado com cada rota para que o aplicativo possa tomar decisões com base em qual rota teve uma correspondência. Esses valores são definidos pelo desenvolvedor e **não** afetam de forma alguma o comportamento do roteamento. Além disso, os valores armazenados em stash em `RouteData.DataTokens` podem ser de qualquer tipo, ao contrário de `RouteData.Values`, que precisa ser conversível bidirecionalmente em cadeias de caracteres.

`RouteData.Routers` é uma lista das rotas que participaram da correspondência bem-sucedida da solicitação. As rotas podem ser aninhadas uma dentro da outra. A propriedade `Routers` reflete o caminho pela árvore lógica de rotas que resultou em uma correspondência. Em geral, o primeiro item em `Routers` é a coleção de rotas e deve ser usado para a geração de URL. O último item em `Routers` é o manipulador de rotas que teve uma correspondência.

## Geração de URL

Geração de URL é o processo pelo qual o roteamento pode criar um caminho de URL de acordo com um conjunto de valores de rota. Isso permite uma separação lógica entre os pontos de extremidade e as URLs que os acessam.

O roteamento de ponto de extremidade inclui a API de Gerador de Link (`LinkGenerator`). `LinkGenerator` é um serviço singleton que pode ser recuperado por meio da DI. A API pode ser usada fora do contexto de

uma solicitação em execução. `IUrlHelper` do MVC e cenários que dependem de `IUrlHelper`, como [Auxiliares de Marcação](#), [Auxiliares de HTML](#) e [Resultados da Ação](#), usam o gerador de link para fornecer funcionalidades de geração de link.

O gerador de link é respaldado pelo conceito de um *endereço* e *esquemas de endereço*. Um esquema de endereço é uma maneira de determinar os pontos de extremidade que devem ser considerados para a geração de link. Por exemplo, os cenários de nome de rota e valores de rota com os quais muitos usuários estão familiarizados no MVC/Razor Pages são implementados como um esquema de endereço.

O gerador de link pode ser vinculado a ações e páginas do MVC/Razor Pages por meio dos seguintes métodos de extensão:

- `GetPathByAction`
- `GetUriByAction`
- `GetPathByPage`
- `GetUriByPage`

Uma sobrecarga desses métodos aceita argumentos que incluem o `HttpContext`. Esses métodos são funcionalmente equivalentes a `Url.Action` e `Url.Page`, mas oferecem mais flexibilidade e opções.

Os métodos `GetPath*` são mais semelhantes a `Url.Action` e `Url.Page`, pois geram um URI que contém um caminho absoluto. Os métodos `GetUri*` sempre geram um URI absoluto que contém um esquema e um host. Os métodos que aceitam um `HttpContext` geram um URI no contexto da solicitação em execução. Os valores de rota de ambiente, o caminho base da URL, o esquema e o host da solicitação em execução são usados, a menos que sejam substituídos.

`LinkGenerator` é chamado com um endereço. A geração de um URI ocorre em duas etapas:

1. Um endereço é associado a uma lista de pontos de extremidade que correspondem ao endereço.
2. O `RoutePattern` de cada ponto de extremidade é avaliado até que seja encontrado um padrão de rota correspondente aos valores fornecidos. A saída resultante é combinada com as outras partes de URI fornecidas ao gerador de link e é retornada.

Os métodos fornecidos pelo `LinkGenerator` dão suporte a funcionalidades de geração de link padrão para qualquer tipo de endereço. A maneira mais conveniente usar o gerador de link é por meio de métodos de extensão que executam operações para um tipo de endereço específico.

MÉTODO DE EXTENSÃO	DESCRIÇÃO
<code>GetPathByAddress</code>	Gera um URI com um caminho absoluto com base nos valores fornecidos.
<code>GetUriByAddress</code>	Gera um URI absoluto com base nos valores fornecidos.

## WARNING

Preste atenção às seguintes implicações da chamada de métodos `LinkGenerator` :

- Use métodos de extensão de `GetUri*` com cuidado em uma configuração de aplicativo que não valide o cabeçalho `Host` das solicitações de entrada. Se o cabeçalho `Host` das solicitações de entrada não é validado, uma entrada de solicitação não confiável pode ser enviada novamente ao cliente em URLs em uma exibição/página. Recomendamos que todos os aplicativos de produção configurem seu servidor para validar o cabeçalho `Host` com os valores válidos conhecidos.
- Use `LinkGenerator` com cuidado no middleware em combinação com `Map` ou `MapWhen`. `Map*` altera o caminho base da solicitação em execução, o que afeta a saída da geração de link. Todas as APIs de `LinkGenerator` permitem a especificação de um caminho base. Sempre especifique um caminho base vazio para desfazer o efeito de `Map*` na geração de link.

## Diferenças das versões anteriores de roteamento

Existem algumas diferenças entre o roteamento de ponto de extremidade no ASP.NET Core 2.2 ou posterior e nas versões anteriores de roteamento no ASP.NET Core:

- O sistema de roteamento do ponto de extremidade não dá suporte à extensibilidade baseada em `IRouter`, incluindo a herança de `Route`.
- O roteamento de ponto de extremidade não dá suporte a `WebApiCompatShim`. Use a [versão de compatibilidade 2.1](#) (`.SetCompatibilityVersion(CompatibilityVersion.Version_2_1)`) para continuar usando o shim de compatibilidade.
- O Roteamento de Ponto de Extremidade tem um comportamento diferente para o uso de maiúsculas dos URLs gerados ao usar rotas convencionais.

Considere o seguinte modelo de rota padrão:

```
app.UseMvc(routes =>
{
    routes.MapRoute("default", "{controller=Home}/{action=Index}/{id?}");
});
```

Suponha que você gere um link para uma ação usando a seguinte rota:

```
var link = Url.Action("ReadPost", "blog", new { id = 17, });
```

Com o roteamento baseado em `IRouter`, esse código gera um URI igual a `/blog/ReadPost/17`, que respeita o uso de maiúsculas do valor de rota fornecido. O roteamento de ponto de extremidade no ASP.NET Core 2.2 ou posterior produz `/Blog/ReadPost/17` ("Blog" está em letras maiúsculas). O roteamento de ponto de extremidade fornece a interface `IOutboundParameterTransformer` que pode ser usada para personalizar esse comportamento globalmente ou para aplicar diferentes convenções ao mapeamento de URLs.

Para obter mais informações, confira a seção [Referência de transformador de parâmetro](#).

- A Geração de Link usada pelo MVC/Razor Pages com rotas convencionais tem um comportamento diferente ao tentar estabelecer um vínculo com um controlador/uma ação ou uma página não existente.

Considere o seguinte modelo de rota padrão:

```
app.UseMvc(routes =>
{
    routes.MapRoute("default", "{controller=Home}/{action=Index}/{id?}");
});
```

Suponha que você gere um link para uma ação usando o modelo padrão com o seguinte:

```
var link = Url.Action("ReadPost", "Blog", new { id = 17, });
```

Com o roteamento baseado em `IRouter`, o resultado é sempre `/Blog/ReadPost/17`, mesmo se o `BlogController` não existe ou não tem um método de ação `ReadPost`. Conforme esperado, o roteamento de ponto de extremidade no ASP.NET Core 2.2 ou posterior produz `/Blog/ReadPost/17` se o método de ação existe. *No entanto, o roteamento de ponto de extremidade produz uma cadeia de caracteres vazia se a ação não existe.* Conceitualmente, o roteamento de ponto de extremidade não pressupõe a existência do ponto de extremidade, caso a ação não exista.

- O *algoritmo de invalidação de valor de ambiente* da geração de link tem um comportamento diferente quando usado com o roteamento de ponto de extremidade.

A *invalidação de valor de ambiente* é o algoritmo que decide quais valores de rota da solicitação em execução no momento (os valores de ambiente) podem ser usados em operações de geração de link. O roteamento convencional sempre invalidava valores de rota extras ao estabelecer o vínculo com uma ação diferente. O roteamento de atributo não tinha esse comportamento antes do lançamento do ASP.NET Core 2.2. Em versões anteriores do ASP.NET Core, os links para outra ação que usam os mesmos nomes de parâmetro de rota resultavam em erros de geração de link. No ASP.NET Core 2.2 ou posterior, as duas formas de roteamento invalidam os valores ao estabelecer o vínculo com outra ação.

Considere o exemplo a seguir no ASP.NET Core 2.1 ou anterior. Ao estabelecer o vínculo com outra ação (ou outra página), os valores de rota podem ser reutilizados de maneiras indesejadas.

Em `/Pages/Store/Product.cshtml`:

```
@page "{id}"
@Url.Page("/Login")
```

Em `/Pages/Login.cshtml`:

```
@page "{id?}"
```

Se o URI é `/Store/Product/18` no ASP.NET Core 2.1 ou anterior, o link gerado na página `Store/Info` por `@Url.Page("/Login")` é `/Login/18`. O valor de `id` igual a 18 é reutilizado, mesmo que o destino do link seja uma parte totalmente diferente do aplicativo. O valor de rota de `id` no contexto da página `/Login` provavelmente é um valor de ID de usuário, e não um valor de ID do produto (product ID) da loja.

No roteamento de ponto de extremidade com o ASP.NET Core 2.2 ou posterior, o resultado é `/Login`. Os valores de ambiente não são reutilizados quando o destino vinculado é uma ação ou uma página diferente.

- Sintaxe do parâmetro de rota de viagem de ida e volta: as barras "/" não são codificadas ao usar uma sintaxe do parâmetro catch-all de asterisco duplo (`**`).

Durante a geração de link, o sistema de roteamento codifica o valor capturado em um parâmetro

catch-all de asterisco duplo (`**`) (por exemplo, `{**myparametername}`), exceto as barras "/". O catch-all de asterisco duplo é compatível com o roteamento baseado em `IRouter` no ASP.NET Core 2.2 ou posterior.

A sintaxe do parâmetro catch-all de asterisco único em versões anteriores do ASP.NET Core (`{*myparametername}`) permanece com suporte e as barras "/" são codificadas.

ROTA	LINK GERADO COM
<code>/search/{*page}</code>	<code>/search/admin%2Fproducts</code> (a barra "/" é codificada)
<code>/search/{**page}</code>	<code>/search/admin/products</code>

## Exemplo de middleware

No exemplo a seguir, um middleware usa a API de `LinkGenerator` para criar um link para um método de ação que lista os produtos da loja. O uso do gerador de link com sua injeção em uma classe e uma chamada a `GenerateLink` está disponível para qualquer classe em um aplicativo.

```
using Microsoft.AspNetCore.Routing;

public class ProductsLinkMiddleware
{
    private readonly LinkGenerator _linkGenerator;

    public ProductsLinkMiddleware(RequestDelegate next, LinkGenerator linkGenerator)
    {
        _linkGenerator = linkGenerator;
    }

    public async Task InvokeAsync(HttpContext httpContext)
    {
        var url = _linkGenerator.GetPathByAction("ListProducts", "Store");

        httpContext.Response.ContentType = "text/plain";

        await httpContext.Response.WriteAsync($"Go to {url} to see our products.");
    }
}
```

Geração de URL é o processo pelo qual o roteamento pode criar um caminho de URL de acordo com um conjunto de valores de rota. Isso permite uma separação lógica entre os manipuladores de rotas e as URLs que os acessam.

A geração de URL segue um processo iterativo semelhante, mas começa com o código da estrutura ou do usuário chamando o método `GetVirtualPath` da coleção de rotas. Cada *rota* tem seu método `GetVirtualPath` chamado em sequência, até que um `VirtualPathData` não nulo seja retornado.

As entradas primárias para `GetVirtualPath` são:

- `VirtualPathContext.HttpContext`
- `VirtualPathContext.Values`
- `VirtualPathContext.AmbientValues`

As rotas usam principalmente os valores de rota fornecidos por `Values` e `AmbientValues` para decidir se é possível gerar uma URL e quais valores serão incluídos. Os `AmbientValues` são o conjunto de valores de rota produzidos pela correspondência da solicitação atual. Por outro lado, `Values` são os valores de rota que especificam como gerar a URL desejada para a operação atual. O `HttpContext` é fornecido para o caso de

uma rota precisar obter serviços ou dados adicionais associados ao contexto atual.

#### TIP

Considere `VirtualPathContext.Values` como um conjunto de substituições de `VirtualPathContext.AmbientValues`. A geração de URL tenta reutilizar os valores de rota da solicitação atual para gerar URLs para links usando a mesma rota ou os mesmos valores de rota.

A saída de `GetVirtualPath` é um `VirtualPathData`. `VirtualPathData` é um paralelo de `RouteData`.

`VirtualPathData` contém o `VirtualPath` da URL de saída e algumas propriedades adicionais que devem ser definidas pela rota.

A propriedade `VirtualPathData.VirtualPath` contém o *caminho virtual* produzido pela rota. Dependendo das suas necessidades, talvez você precise processar ainda mais o caminho. Se você desejar renderizar a URL gerada em HTML, preceda o caminho base do aplicativo.

O `VirtualPathData.Router` é uma referência à rota que gerou a URL com êxito.

As propriedades de `VirtualPathData.DataTokens` são um dicionário de dados adicionais relacionados à rota que gerou a URL. Isso é o paralelo de `RouteData.DataTokens`.

#### Criar rotas

O roteamento fornece a classe `Route` como a implementação padrão de `IRouter`. `Route` usa a sintaxe de *modelo de rota* para definir padrões que corresponderão ao caminho da URL quando `RouteAsync` for chamado. `Route` usa o mesmo modelo de rota para gerar uma URL quando `GetVirtualPath` é chamado.

A maioria dos aplicativos cria rotas chamando `MapRoute` ou um dos métodos de extensão semelhantes definidos em `IRouteBuilder`. Qualquer um dos métodos de extensão de `IRouteBuilder` cria uma instância de `Route` e a adicionam à coleção de rotas.

`MapRoute` não aceita um parâmetro de manipulador de rotas. `MapRoute` apenas adiciona rotas que são manipuladas pelo `DefaultHandler`. Para saber mais sobre o roteamento no MVC, confira [Roteamento para ações do controlador no ASP.NET Core](#).

`MapRoute` não aceita um parâmetro de manipulador de rotas. `MapRoute` apenas adiciona rotas que são manipuladas pelo `DefaultHandler`. O manipulador padrão é um `IRouter`, e o manipulador não pode manipular a solicitação. Por exemplo, o ASP.NET Core MVC normalmente é configurado como um manipulador padrão que só manipula as solicitações que correspondem a um controlador e a uma ação disponíveis. Para saber mais sobre o roteamento no MVC, confira [Roteamento para ações do controlador no ASP.NET Core](#).

O exemplo de código a seguir é um exemplo de uma chamada `MapRoute` usada por uma definição de rota típica do ASP.NET Core MVC:

```
routes.MapRoute(
    name: "default",
    template: "{controller=Home}/{action=Index}/{id?}");
```

Esse modelo corresponde a um caminho de URL e extrai os valores de rota. Por exemplo, o caminho `/Products/Details/17` gera os seguintes valores de rota:

```
{ controller = Products, action = Details, id = 17 }.
```

Os valores de rota são determinados pela divisão do caminho da URL em segmentos e pela correspondência de cada segmento com o nome do *parâmetro de rota* no modelo de rota. Os parâmetros de rota são nomeados. Os parâmetros são definidos com a colocação do nome do parâmetro em chaves `{ ... }`.

O modelo anterior também pode corresponder ao caminho da URL `/` e produzir os valores `{ controller = Home, action = Index }`. Isso ocorre porque os parâmetros de rota `{controller}` e `{action}` têm valores padrão e o parâmetro de rota `id` é opcional. Um sinal de igual (`=`) seguido de um valor após o nome do parâmetro de rota define um valor padrão para o parâmetro. Um ponto de interrogação (`?`) após o nome do parâmetro de rota define o parâmetro como opcional.

Os parâmetros de rota com um valor padrão *sempre* produzem um valor de rota quando a rota corresponde. Os parâmetros opcionais não produzem um valor de rota quando não há nenhum segmento de caminho de URL correspondente. Confira a seção [Referência de modelo de rota](#) para obter uma descrição completa dos recursos e da sintaxe de modelo de rota.

No seguinte exemplo, a definição do parâmetro de rota `{id:int}` define uma [restrição de rota](#) para o parâmetro de rota `id`:

```
routes.MapRoute(  
    name: "default",  
    template: "{controller=Home}/{action=Index}/{id:int}");
```

Esse modelo corresponde a um caminho de URL, como `/Products/Details/17`, mas não como `/Products/Details/Apples`. As restrições de rota implementam [IRouteConstraint](#) e inspecionam valores de rota para verificar-lhos. Neste exemplo, o valor de rota `id` precisa ser conversível em um inteiro. Confira [route-constraint-reference](#) para obter uma explicação das restrições de rota fornecidas pela estrutura.

Sobrecargas adicionais de `MapRoute` aceitam valores para `constraints`, `dataTokens` e `defaults`. O uso típico desses parâmetros é passar um objeto de tipo anônimo, no qual os nomes da propriedade do tipo anônimo correspondem aos nomes do parâmetro de rota.

Os seguintes exemplos de `MapRoute` criam rotas equivalentes:

```
routes.MapRoute(  
    name: "default_route",  
    template: "{controller}/{action}/{id?}",  
    defaults: new { controller = "Home", action = "Index" });  
  
routes.MapRoute(  
    name: "default_route",  
    template: "{controller=Home}/{action=Index}/{id?}");
```

#### TIP

A sintaxe embutida para a definição de restrições e padrões pode ser interessante para rotas simples. No entanto, há cenários, como tokens de dados, que não dão suporte à sintaxe embutida.

O seguinte exemplo demonstra mais alguns cenários:

```
routes.MapRoute(  
    name: "blog",  
    template: "Blog/{**article}",  
    defaults: new { controller = "Blog", action = "ReadArticle" });
```

O modelo anterior corresponde a um caminho de URL, como `/Blog/All-About-Routing/Introduction`, e extrai os valores `{ controller = Blog, action = ReadArticle, article = All-About-Routing/Introduction }`. Os valores de rota padrão para `controller` e `action` são produzidos pela rota, mesmo que não haja nenhum parâmetro de rota correspondente no modelo. Os valores padrão podem ser especificados no modelo de

rota. O parâmetro de rota `article` é definido como um *catch-all* pela aparência de um asterisco duplo (`**`) antes do nome do parâmetro de rota. Os parâmetros de rota *catch-all* capturam o restante do caminho de URL e também podem corresponder à cadeia de caracteres vazia.

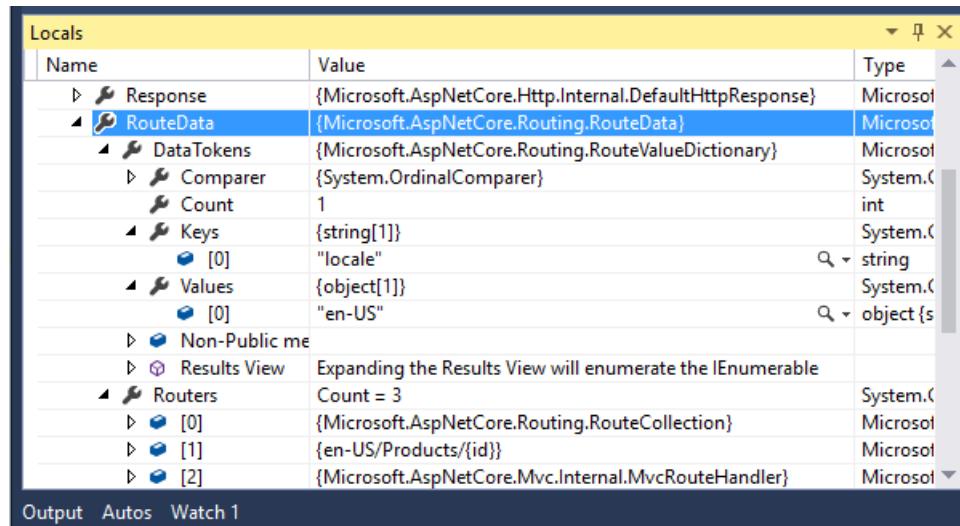
```
routes.MapRoute(
    name: "blog",
    template: "Blog/{*article}",
    defaults: new { controller = "Blog", action = "ReadArticle" });
```

O modelo anterior corresponde a um caminho de URL, como `/Blog/All-About-Routing/Introduction`, e extrai os valores `{ controller = Blog, action = ReadArticle, article = All-About-Routing/Introduction }`. Os valores de rota padrão para `controller` e `action` são produzidos pela rota, mesmo que não haja nenhum parâmetro de rota correspondente no modelo. Os valores padrão podem ser especificados no modelo de rota. O parâmetro de rota `article` é definido como um *catch-all* pela aparência de um asterisco (`*`) antes do nome do parâmetro de rota. Os parâmetros de rota *catch-all* capturam o restante do caminho de URL e também podem corresponder à cadeia de caracteres vazia.

O seguinte exemplo adiciona restrições de rota e tokens de dados:

```
routes.MapRoute(
    name: "us_english_products",
    template: "en-US/Products/{id}",
    defaults: new { controller = "Products", action = "Details" },
    constraints: new { id = new IntRouteConstraint() },
    dataTokens: new { locale = "en-US" });
```

O modelo anterior corresponde a um caminho de URL como `/en-US/Products/5` e extrai os valores de `{ controller = Products, action = Details, id = 5 }` e os tokens de dados `{ locale = en-US }`.



## Geração de URL da classe de rota

A classe `Route` também pode executar a geração de URL pela combinação de um conjunto de valores de rota com seu modelo de rota. Logicamente, este é o processo inverso de correspondência do caminho de URL.

### TIP

Para entender melhor a geração de URL, imagine qual URL você deseja gerar e, em seguida, pense em como um modelo de rota corresponderia a essa URL. Quais valores serão produzidos? Este é o equivalente aproximado de como funciona a geração de URL na classe `Route`.

O seguinte exemplo usa uma rota padrão geral do ASP.NET Core MVC:

```
routes.MapRoute(
    name: "default",
    template: "{controller=Home}/{action=Index}/{id?}");
```

Com os valores de rota `{ controller = Products, action = List }`, a URL `/Products/List` é gerada. Os valores de rota são substituídos pelos parâmetros de rota correspondentes para formar o caminho de URL. Como `id` é um parâmetro de rota opcional, a URL é gerada com êxito sem um valor para `id`.

Com os valores de rota `{ controller = Home, action = Index }`, a URL `/` é gerada. Os valores de rota fornecidos correspondem aos valores padrão, sendo que os segmentos correspondentes aos valores padrão são omitidos com segurança.

A viagem de ida e volta gerada pelas duas URLs com a definição de rota a seguir (`/Home/Index` e `/`) produz os mesmos valores de rota que foram usados para gerar a URL.

#### NOTE

Um aplicativo que usa o ASP.NET Core MVC deve usar [UrlHelper](#) para gerar URLs, em vez de chamar o roteamento diretamente.

Para obter mais informações sobre a geração de URL, confira a seção [Referência de geração de URL](#).

## Usar o middleware de roteamento

Referencie o [metapacote Microsoft.AspNetCore.App](#) no arquivo de projeto do aplicativo.

Adicione o roteamento ao contêiner de serviço em `Startup.ConfigureServices`:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddRouting();
}
```

As rotas precisam ser configuradas no método `Startup.Configure`. O aplicativo de exemplo usa as seguintes APIs:

- [RouteBuilder](#)
- [MapGet](#) – Corresponde apenas às solicitações HTTP GET.
- [UseRouter](#)

```

var trackPackageRouteHandler = new RouteHandler(context =>
{
    var routeValues = context.GetRouteData().Values;
    return context.Response.WriteAsync(
        $"Hello! Route values: {string.Join(", ", routeValues)}");
});

var routeBuilder = new RouteBuilder(app, trackPackageRouteHandler);

routeBuilder.MapRoute(
    "Track Package Route",
    "package/{operation:regex(^track|create|detonate$)}/{id:int}");

routeBuilder.MapGet("hello/{name}", context =>
{
    var name = context.GetRouteValue("name");
    // The route handler when HTTP GET "hello/<anything>" matches
    // To match HTTP GET "hello/<anything>/<anything>",
    // use routeBuilder.MapGet("hello/{*name}")
    return context.Response.WriteAsync($"Hi, {name}!");
});

var routes = routeBuilder.Build();
app.UseRouter(routes);

```

A tabela a seguir mostra as respostas com os URIs fornecidos.

URI	RESPOSTA
/package/create/3	Olá! Valores de rota: [operation, create], [id, 3]
/package/track/-3	Olá! Valores de rota: [operation, track], [id, -3]
/package/track/-3/	Olá! Valores de rota: [operation, track], [id, -3]
/package/track/	A solicitação é ignorada; sem correspondência.
GET /hello/Joe	Olá, Joe!
POST /hello/Joe	A solicitação é ignorada; corresponde apenas a HTTP GET.
GET /hello/Joe/Smith	A solicitação é ignorada; sem correspondência.

Se você estiver configurando uma única rota, chame [UseRouter](#) passando uma instância de [IRouter](#). Você não precisa usar [RouteBuilder](#).

A estrutura fornece um conjunto de métodos de extensão para a criação de rotas ([RequestDelegateRouteBuilderExtensions](#)):

- [MapDelete](#)
- [MapGet](#)
- [MapMiddlewareDelete](#)
- [MapMiddlewareGet](#)
- [MapMiddlewarePost](#)
- [MapMiddlewarePut](#)
- [MapMiddlewareRoute](#)

- `MapMiddlewareVerb`
- `MapPost`
- `MapPut`
- `MapRoute`
- `MapVerb`

Alguns dos métodos listados, como `MapGet`, exigem um `RequestDelegate`. O `RequestDelegate` é usado como o *manipulador de rotas* quando a rota corresponde. Outros métodos nesta família permitem configurar um pipeline de middleware a ser usado como o manipulador de rotas. Se o método `Map*` não aceitar um manipulador, como `MapRoute`, ele usará o `DefaultHandler`.

Os métodos `Map[Verb]` usam restrições para limitar a rota ao Verbo HTTP no nome do método. Por exemplo, veja `MapGet` e `MapVerb`.

## Referência de modelo de rota

Os tokens entre chaves (`{ ... }`) definem os *parâmetros de rota* que serão associados se a rota for correspondida. Você pode definir mais de um parâmetro de rota em um segmento de rota, mas eles precisam ser separados por um valor literal. Por exemplo, `{controller=Home}{action=Index}` não é uma rota válida, já que não há nenhum valor literal entre `{controller}` e `{action}`. Esses parâmetros de rota precisam ter um nome e podem ter atributos adicionais especificados.

Um texto literal diferente dos parâmetros de rota (por exemplo, `{id}`) e do separador de caminho `/` precisa corresponder ao texto na URL. A correspondência de texto não diferencia maiúsculas de minúsculas e se baseia na representação decodificada do caminho de URLs. Para encontrar a correspondência de um delimitador de parâmetro de rota literal (`{` ou `}`), faça o escape do delimitador repetindo o caractere (`{}{` ou `}{{`}).

Padrões de URL que tentam capturar um nome de arquivo com uma extensão de arquivo opcional apresentam considerações adicionais. Por exemplo, considere o modelo `files/{filename}.{ext?}`. Quando existem valores para `filename` e `ext`, ambos os valores são populados. Se apenas existir um valor para `filename` na URL, a rota encontrará uma correspondência, pois o ponto à direita (`.`) é opcional. As URLs a seguir correspondem a essa rota:

- `/files/myFile.txt`
- `/files/myFile`

Você pode usar um asterisco (`*`) ou um asterisco duplo (`**`) como um prefixo para um parâmetro de rota para associá-lo ao restante do URI. Eles são chamados de parâmetros *catch-all*. Por exemplo, `blog/**slug` corresponde a qualquer URI que comece com `/blog` e tem qualquer valor depois dele, que é atribuído ao valor de rota `slug`. Os parâmetros catch-all também podem corresponder à cadeia de caracteres vazia.

O parâmetro catch-all faz o escape dos caracteres corretos quando a rota é usada para gerar uma URL, incluindo os caracteres separadores de caminho (`/`). Por exemplo, a rota `foo/{*path}` com valores de rota `{ path = "my/path" }` gera `foo/my%2Fpath`. Observe o escape da barra invertida. Para fazer a viagem de ida e volta dos caracteres separadores de caminho, use o prefixo do parâmetro da rota `**`. A rota `foo/**path` com `{ path = "my/path" }` gera `foo/my/path`.

Você pode usar o asterisco (`*`) como um prefixo para um parâmetro de rota a ser associado ao restante do URI. Isso é chamado de parâmetro *catch-all*. Por exemplo, `blog/*slug` corresponde a qualquer URI que comece com `/blog` e tem qualquer valor depois dele, que é atribuído ao valor de rota `slug`. Os parâmetros catch-all também podem corresponder à cadeia de caracteres vazia.

O parâmetro catch-all faz o escape dos caracteres corretos quando a rota é usada para gerar uma URL,

incluindo os caracteres separadores de caminho (`/`). Por exemplo, a rota `foo/{*path}` com valores de rota `{ path = "my/path" }` gera `foo/my%2Fpath`. Observe o escape da barra invertida.

Os parâmetros de rota podem ter *valores padrão*, designados pela especificação do valor padrão após o nome do parâmetro separado por um sinal de igual (`=`). Por exemplo, `{controller=Home}` define `Home` como o valor padrão de `controller`. O valor padrão é usado se nenhum valor está presente na URL para o parâmetro. Os parâmetros de rota se tornam opcionais com o acréscimo de um ponto de interrogação (`?`) ao final do nome do parâmetro, como em `id?`. A diferença entre valores opcionais e parâmetros de rota padrão é que um parâmetro de rota com um valor padrão sempre produz um valor – um parâmetro opcional tem um valor somente quando um valor é fornecido pela URL de solicitação.

Os parâmetros de rota podem ter restrições que precisam corresponder ao valor de rota associado da URL. A adição de dois-pontos (`:`) e do nome da restrição após o nome do parâmetro de rota especifica uma *restrição embutida* em um parâmetro de rota. Se a restrição exigir argumentos, eles ficarão entre parênteses (`( ... )`) após o nome da restrição. Várias restrições embutidas podem ser especificadas por meio do acréscimo de outros dois-pontos (`:`) e do nome da restrição.

O nome da restrição e os argumentos são passados para o serviço [IInlineConstraintResolver](#) para criar uma instância de [IRouteConstraint](#) a ser usada no processamento de URL. Por exemplo, o modelo de rota `blog/{article:minlength(10)}` especifica uma restrição `minlength` com o argumento `10`. Para obter mais informações sobre as restrições de rota e uma lista das restrições fornecidas pela estrutura, confira a seção [Referência de restrição de rota](#).

Os parâmetros de rota também podem ter transformadores de parâmetro, que transformam o valor de um parâmetro ao gerar links e fazer a correspondência de ações e páginas com URLs. Assim como as restrições, os transformadores de parâmetro podem ser adicionados embutidos a um parâmetro de rota colocando dois-pontos (`:`) e o nome do transformador após o nome do parâmetro de rota. Por exemplo, o modelo de rota `blog/{article:slugify}` especifica um transformador `slugify`. Para obter mais informações sobre transformadores de parâmetro, confira a seção [Referência de transformador de parâmetro](#).

A tabela a seguir demonstra modelos de rota de exemplo e seu comportamento.

MODELO DE ROTA	URI DE CORRESPONDÊNCIA DE EXEMPLO	O URI DE SOLICITAÇÃO...
<code>hello</code>	<code>/hello</code>	Somente corresponde ao caminho único <code>/hello</code> .
<code>{Page=Home}</code>	<code>/</code>	Faz a correspondência e define <code>Page</code> como <code>Home</code> .
<code>{Page=Home}</code>	<code>/Contact</code>	Faz a correspondência e define <code>Page</code> como <code>Contact</code> .
<code>{controller}/{action}/{id?}</code>	<code>/Products&gt;List</code>	É mapeado para o controlador <code>Products</code> e a ação <code>List</code> .
<code>{controller}/{action}/{id?}</code>	<code>/Products/Details/123</code>	É mapeado para o controlador <code>Products</code> e a ação <code>Details</code> ( <code>id</code> definido como 123).
<code>{controller=Home}/{action=Index}/{id / }</code>		É mapeado para o controlador <code>Home</code> e o método <code>Index</code> ( <code>id</code> é ignorado).

Em geral, o uso de um modelo é a abordagem mais simples para o roteamento. Restrições e padrões

também podem ser especificados fora do modelo de rota.

#### TIP

Habilite o [Log](#) para ver como as implementações de roteamento internas, como `Route`, correspondem às solicitações.

## Nomes reservados de roteamento

As seguintes palavras-chave são nomes reservados e não podem ser usadas como nomes de rota ou parâmetros:

- `action`
- `area`
- `controller`
- `handler`
- `page`

## Referência de restrição de rota

As restrições de rota são executadas quando ocorre uma correspondência com a URL de entrada e é criado um token do caminho da URL em valores de rota. Em geral, as restrições da rota inspecionam o valor de rota associado por meio do modelo de rota e tomam uma decisão do tipo "sim/não" sobre se o valor é aceitável ou não. Algumas restrições da rota usam dados fora do valor de rota para considerar se a solicitação pode ser encaminhada. Por exemplo, a [HttpMethodRouteConstraint](#) pode aceitar ou rejeitar uma solicitação de acordo com o verbo HTTP. As restrições são usadas em solicitações de roteamento e na geração de link.

#### WARNING

Não use restrições para a **validação de entrada**. Se as restrições forem usadas para a **validação de entrada**, uma entrada inválida resultará em uma resposta `404 – Não Encontrado`, em vez de `400 – Solicitação Inválida` com uma mensagem de erro apropriada. As restrições de rota são usadas para **desfazer a ambiguidade** entre rotas semelhantes, não para validar as entradas de uma rota específica.

A tabela a seguir demonstra restrições de rota de exemplo e seu comportamento esperado.

RESTRIÇÃO	EXEMPLO	CORRESPONDÊNCIAS DE EXEMPLO	OBSERVAÇÕES
<code>int</code>	<code>{id:int}</code>	<code>123456789</code> , <code>-123456789</code>	Corresponde a qualquer inteiro
<code>bool</code>	<code>{active:bool}</code>	<code>true</code> , <code>FALSE</code>	Corresponde a <code>true</code> ou <code>false</code> (não diferencia maiúsculas de minúsculas)
<code>datetime</code>	<code>{dob:datetime}</code>	<code>2016-12-31</code> , <code>2016-12-31 7:32pm</code>	Corresponde a um valor <code>DateTime</code> válido (na cultura invariável – veja o aviso)

RESTRIÇÃO	EXEMPLO	CORRESPONDÊNCIAS DE EXEMPLO	OBSERVAÇÕES
<code>decimal</code>	<code>{price:decimal}</code>	<code>49.99</code> , <code>-1,000.01</code>	Corresponde a um valor <code>decimal</code> válido (na cultura invariável – veja o aviso)
<code>double</code>	<code>{weight:double}</code>	<code>1.234</code> , <code>-1,001.01e8</code>	Corresponde a um valor <code>double</code> válido (na cultura invariável – veja o aviso)
<code>float</code>	<code>{weight:float}</code>	<code>1.234</code> , <code>-1,001.01e8</code>	Corresponde a um valor <code>float</code> válido (na cultura invariável – veja o aviso)
<code>guid</code>	<code>{id:guid}</code>	<code>CD2C1638-1638-72D5-1638-DEADBEEF1638</code> , <code>{CD2C1638-1638-72D5-1638-DEADBEEF1638}</code>	Corresponde a um valor <code>Guid</code> válido
<code>long</code>	<code>{ticks:long}</code>	<code>123456789</code> , <code>-123456789</code>	Corresponde a um valor <code>long</code> válido
<code>minlength(value)</code>	<code>{username:minlength(4)}</code>	<code>Rick</code>	A cadeia de caracteres deve ter, no mínimo, 4 caracteres
<code>maxlength(value)</code>	<code>{filename:maxlength(8)}</code>	<code>Richard</code>	A cadeia de caracteres não pode ser maior que 8 caracteres
<code>length(length)</code>	<code>{filename:length(12)}</code>	<code>somefile.txt</code>	A cadeia de caracteres deve ter exatamente 12 caracteres
<code>length(min,max)</code>	<code>{filename:length(8,16)}</code>	<code>somefile.txt</code>	A cadeia de caracteres deve ter, pelo menos, 8 e não mais de 16 caracteres
<code>min(value)</code>	<code>{age:min(18)}</code>	<code>19</code>	O valor inteiro deve ser, pelo menos, 18
<code>max(value)</code>	<code>{age:max(120)}</code>	<code>91</code>	O valor inteiro não deve ser maior que 120
<code>range(min,max)</code>	<code>{age:range(18,120)}</code>	<code>91</code>	O valor inteiro deve ser, pelo menos, 18, mas não maior que 120
<code>alpha</code>	<code>{name:alpha}</code>	<code>Rick</code>	A cadeia de caracteres deve consistir em um ou mais caracteres alfabéticos ( <code>a - z</code> , não diferencia maiúsculas de minúsculas)

RESTRIÇÃO	EXEMPLO	CORRESPONDÊNCIAS DE EXEMPLO	OBSERVAÇÕES
<code>regex(expression)</code>	<code>{ssn:regex(`^\\d{3}-\\d{2}-\\d{4}`)}</code>	<code>123-45-6789</code>	A cadeia de caracteres deve corresponder à expressão regular (veja as dicas sobre como definir uma expressão regular)
<code>required</code>	<code>{name:required}</code>	<code>Rick</code>	Usado para impor que um valor não parâmetro está presente durante a geração de URL

Várias restrições delimitadas por vírgula podem ser aplicadas a um único parâmetro. Por exemplo, a restrição a seguir restringe um parâmetro para um valor inteiro de 1 ou maior:

```
[Route("users/{id:int:min(1)}")]
public User GetUserById(int id) { }
```

#### WARNING

As restrições de rota que verificam a URL e são convertidas em um tipo CLR (como `int` ou `DateTime`) sempre usam a cultura invariável. Essas restrições consideram que a URL não é localizável. As restrições de rota fornecidas pela estrutura não modificam os valores armazenados nos valores de rota. Todos os valores de rota analisados com base na URL são armazenados como cadeias de caracteres. Por exemplo, a restrição `float` tenta converter o valor de rota em um `float`, mas o valor convertido é usado somente para verificar se ele pode ser convertido em um `float`.

## Expressões regulares

A estrutura do ASP.NET Core adiciona

`RegexOptions.IgnoreCase` | `RegexOptions.Compiled` | `RegexOptions.CultureInvariant` ao construtor de expressão regular. Confira [RegexOptions](#) para obter uma descrição desses membros.

As expressões regulares usam delimitadores e tokens semelhantes aos usados pelo Roteamento e pela linguagem C#. Os tokens de expressão regular precisam ter escape. Para usar a expressão regular `^\d{3}-\d{2}-\d{4}$` no roteamento, a expressão precisa ter os caracteres `\`` (barra invertida) fornecidos na cadeia de caracteres como caracteres `\\`` (barra invertida dupla) no arquivo de origem C# para fazer o escape do caractere de escape da cadeia de caracteres `\`` (a menos que estejam sendo usados [literais de cadeia de caracteres textuais](#)). Para fazer o escape dos caracteres de delimitador de parâmetro de roteamento (`{`, `}`, `[`, `]`), duplique os caracteres na expressão (`{{`, `}}`, `[[`, `]]`). A tabela a seguir mostra uma expressão regular e a versão com escape.

EXPRESSÃO REGULAR	EXPRESSÃO REGULAR COM ESCAPE
<code>^\d{3}-\d{2}-\d{4}\$</code>	<code>^\\d{3}-\\d{2}-\\d{4}\$</code>
<code>^[a-z]{2}\$</code>	<code>^[[a-z]]{{2}}\$</code>

As expressões regulares usadas no roteamento geralmente começam com o caractere de acento circunflexo (`^`) e correspondem à posição inicial da cadeia de caracteres. As expressões geralmente terminam com o caractere de cifrão (`$`) e correspondem ao final da cadeia de caracteres. Os caracteres `^` e `$` garantem que a expressão regular corresponde a todo o valor do parâmetro de rota. Sem os caracteres `^` e `$`, a

expressão regular corresponde a qualquer subcadeia de caracteres na cadeia de caracteres, o que geralmente não é o desejado. A tabela a seguir fornece exemplos e explica por que eles encontram ou não uma correspondência.

EXPRESSÃO	CADEIA DE CARACTERES	CORRESPONDER A	COMENTÁRIO
<code>[a-z]{2}</code>	hello	Sim	A subcadeia de caracteres corresponde
<code>[a-z]{2}</code>	123abc456	Sim	A subcadeia de caracteres corresponde
<code>[a-z]{2}</code>	mz	Sim	Corresponde à expressão
<code>[a-z]{2}</code>	MZ	Sim	Não diferencia maiúsculas de minúsculas
<code>^[a-z]{2}\$</code>	hello	Não	Confira <code>^</code> e <code>\$</code> acima
<code>^[a-z]{2}\$</code>	123abc456	Não	Confira <code>^</code> e <code>\$</code> acima

Para saber mais sobre a sintaxe de expressões regulares, confira [Expressões regulares do .NET Framework](#).

Para restringir um parâmetro a um conjunto conhecido de valores possíveis, use uma expressão regular. Por exemplo, `{action:regex(^list|get|create)}` apenas corresponde o valor da rota `action` a `list`, `get` ou `create`. Se passada para o dicionário de restrições, a cadeia de caracteres `^(list|get|create)$` é equivalente. As restrições passadas para o dicionário de restrições (não embutidas em um modelo) que não correspondem a uma das restrições conhecidas também são tratadas como expressões regulares.

## Restrições de rota personalizadas

Além das restrições de rota internas, é possível criar restrições de rota personalizadas com a implementação da interface do `IRouteConstraint`. A interface do `IRouteConstraint` contém um único método, `Match`, que retorna `true` quando a restrição é satisfeita. Caso contrário, retorna `false`.

Para usar uma `IRouteConstraint` personalizada, o tipo de restrição de rota deve ser registrado com o `RouteOptions.ConstraintMap` do aplicativo, no contêiner de serviço do aplicativo. O `ConstraintMap` é um dicionário que mapeia as chaves de restrição de rota para implementações de `IRouteConstraint` que validam essas restrições. É possível atualizar o `RouteOptions.ConstraintMap` do aplicativo no `Startup.ConfigureServices` como parte de uma chamada `services.AddRouting` ou configurando `RouteOptions` diretamente com `services.Configure<RouteOptions>`. Por exemplo:

```
services.AddRouting(options =>
{
    options.ConstraintMap.Add("customName", typeof(MyCustomConstraint));
});
```

a restrição pode então ser aplicada às rotas da maneira usual, usando o nome especificado ao registrar o tipo de restrição. Por exemplo:

```
[HttpGet("{id:customName}")]
public ActionResult<string> Get(string id)
```

# Referência de parâmetro de transformador

Transformadores de parâmetro:

- Executar ao gerar um link para um `Route`.
- Implementar `Microsoft.AspNetCore.Routing.IOutboundParameterTransformer`.
- São configurados usando `ConstraintMap`.
- Usam o valor de rota do parâmetro e o transformam em um novo valor de cadeia de caracteres.
- Resultam no uso do valor transformado no link gerado.

Por exemplo, um transformador de parâmetro `slugify` personalizado em padrão de rota

```
blog\{article:slugify} com Url.Action(new { article = "MyTestArticle" }) gera blog\my-test-article.
```

Para usar um transformador de parâmetro em um padrão de rota, configure-o primeiro usando `ConstraintMap` em `Startup.ConfigureServices`:

```
services.AddRouting(options =>
{
    // Replace the type and the name used to refer to it with your own
    // IOutboundParameterTransformer implementation
    options.ConstraintMap["slugify"] = typeof(SlugifyParameterTransformer);
});
```

Os transformadores de parâmetro são usados pela estrutura para transformar o URI no qual um ponto de extremidade é resolvido. Por exemplo, o ASP.NET Core MVC usa os transformadores de parâmetro para transformar o valor de rota usado para corresponder a um `area`, `controller`, `action` e `page`.

```
routes.MapRoute(
    name: "default",
    template: "{controller:slugify=Home}/{action:slugify=Index}/{id?}");
```

Com a rota anterior, a ação `SubscriptionManagementController.GetAll()` é combinada com o URI `/subscription-management/get-all`. Um transformador de parâmetro não altera os valores de rota usados para gerar um link. Por exemplo, `Url.Action("GetAll", "SubscriptionManagement")` gera `/subscription-management/get-all`.

ASP.NET Core fornece convenções de API para usar transformadores de parâmetro com as rotas geradas:

- ASP.NET Core MVC tem a convenção de API `Microsoft.AspNetCore.Mvc.ApplicationModels.RouteTokenTransformerConvention`. Essa convenção aplica um transformador de parâmetro especificado a todas as rotas de atributo no aplicativo. O transformador de parâmetro transforma os tokens de rota do atributo conforme elas são substituídas. Para obter mais informações, confira [Usar um transformador de parâmetro para personalizar a substituição de token](#).
- O Razor Pages tem a convenção de API `Microsoft.AspNetCore.Mvc.ApplicationModels.PageRouteTransformerConvention`. Essa convenção aplica um transformador de parâmetro especificado a todas as Razor Pages descobertas automaticamente. O transformador de parâmetro transforma os segmentos de nome de arquivo e pasta de rotas do Razor Pages. Para obter mais informações, confira [Usar um transformador de parâmetros para personalizar rotas de página](#).

## Referência de geração de URL

O exemplo a seguir mostra como gerar um link para uma rota com base em um dicionário de valores de rota e em um `RouteCollection`.

```

app.Run(async (context) =>
{
    var dictionary = new RouteValueDictionary
    {
        { "operation", "create" },
        { "id", 123}
    };

    var vpc = new VirtualPathContext(context, null, dictionary,
        "Track Package Route");
    var path = routes.GetVirtualPath(vpc).VirtualPath;

    context.Response.ContentType = "text/html";
    await context.Response.WriteAsync("Menu<hr/>");
    await context.Response.WriteAsync(
        $"<a href='{path}'>Create Package 123</a><br/>");
});

```

O `VirtualPath` gerado no final do exemplo anterior é `/package/create/123`. O dicionário fornece os valores de rota `operation` e `id` do modelo "Rastrear rota do pacote", `package/{operation}/{id}`. Para obter detalhes, consulte o código de exemplo na seção [Usar o middleware de roteamento](#) ou no [aplicativo de exemplo](#).

O segundo parâmetro para o construtor `VirtualPathContext` é uma coleção de *valores de ambiente*. Os valores de ambiente são convenientes de serem usados porque limitam o número de valores que um desenvolvedor precisa especificar em um contexto de solicitação. Os valores de rota atuais da solicitação atual são considerados valores de ambiente para a geração de link. Na ação `About` de um aplicativo ASP.NET Core MVC do `HomeController`, não é necessário especificar o valor de rota do controlador a ser vinculado à ação `Index` – o valor de ambiente `Home` é usado.

Os valores de ambiente que não correspondem a um parâmetro são ignorados. Os valores de ambiente também são ignorados quando um valor fornecido explicitamente substitui o valor de ambiente. A correspondência ocorre da esquerda para a direita na URL.

Valores fornecidos explicitamente, mas que não correspondem a um segmento da rota, são adicionados à cadeia de consulta. A tabela a seguir mostra o resultado do uso do modelo de rota

`{controller}/{action}/{id?}` .

VALORES DE AMBIENTE	VALORES EXPLÍCITOS	RESULTADO
controlador = "Home"	ação = "About"	<code>/Home/About</code>
controlador = "Home"	controlador = "Order", ação = "About"	<code>/Order/About</code>
controlador = "Home", cor = "Red"	ação = "About"	<code>/Home/About</code>
controlador = "Home"	ação = "About", cor = "Red"	<code>/Home/About?color=Red</code>

Se uma rota tem um valor padrão que não corresponde a um parâmetro e esse valor é fornecido de forma explícita, ele precisa corresponder ao valor padrão:

```

routes.MapRoute("blog_route", "blog/{*slug}",
    defaults: new { controller = "Blog", action = "ReadPost" });

```

A geração de link somente gera um link para essa rota quando os valores correspondentes de `controller` e

`action` são fornecidos.

## Segmentos complexos

Segmentos complexos (por exemplo, `[Route("/x{token}y")]`) são processados por meio da combinação de literais da direita para a esquerda, de uma maneira diferente de Greedy. Confira [este código](#) para ver uma explicação detalhada de como os segmentos complexos são combinados. O [exemplo de código](#) não é usado pelo ASP.NET Core, mas fornece uma explicação adequada sobre segmentos complexos.

# Usar vários ambientes no ASP.NET Core

30/01/2019 • 15 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

O ASP.NET Core configura o comportamento do aplicativo com base no ambiente de tempo de execução usando uma variável de ambiente.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Ambientes

O ASP.NET Core lê a variável de ambiente `ASPNETCORE_ENVIRONMENT` na inicialização do aplicativo e armazena o valor em `IHostingEnvironment.EnvironmentName`. Você pode definir `ASPNETCORE_ENVIRONMENT` como qualquer valor, mas há suporte para [três valores](#) na estrutura: [Desenvolvimento](#), [Preparo](#) e [Produção](#). Se `ASPNETCORE_ENVIRONMENT` não estiver definido, o padrão será `Production`.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    if (env.IsProduction() || env.IsStaging() || env.IsEnvironment("Staging_2"))
    {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseCookiePolicy();
    app.UseMvc();
}
```

O código anterior:

- Chama `UseDeveloperExceptionPage` quando `ASPNETCORE_ENVIRONMENT` é definido como `Development`.
- Chama `UseExceptionHandler` quando o valor de `ASPNETCORE_ENVIRONMENT` é definido com um dos seguintes:
  - `Staging`
  - `Production`
  - `Staging_2`

O [Auxiliar de Marcação de Ambiente](#) usa o valor de `IHostingEnvironment.EnvironmentName` para incluir ou excluir a marcação no elemento:

```
<environment include="Development">
    <div>&lt;environment include="Development"&gt;</div>
</environment>
<environment exclude="Development">
    <div>&lt;environment exclude="Development"&gt;</div>
</environment>
<environment include="Staging,Development,Staging_2">
    <div>
        &lt;environment include="Staging,Development,Staging_2"&gt;
        </div>
    </environment>
```

No Windows e no macOS, valores e variáveis de ambiente não diferenciam maiúsculas de minúsculas.

Valores e variáveis de ambiente do Linux **diferenciam maiúsculas de minúsculas** por padrão.

## Desenvolvimento

O ambiente de desenvolvimento pode habilitar recursos que não devem ser expostos em produção. Por exemplo, os modelos do ASP.NET Core habilitam a [página de exceção do desenvolvedor](#) no ambiente de desenvolvimento.

O ambiente de desenvolvimento do computador local pode ser definido no arquivo *Properties\launchSettings.json* do projeto. Os valores de ambiente definidos em *launchSettings.json* substituem os valores definidos no ambiente do sistema.

O seguinte JSON mostra três perfis de um arquivo *launchSettings.json*:

```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:54339/",
      "sslPort": 0
    }
  },
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_My_Environment": "1",
        "ASPNETCORE_DETAILEDERRORS": "1",
        "ASPNETCORE_ENVIRONMENT": "Staging"
      }
    },
    "EnvironmentsSample": {
      "commandName": "Project",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Staging"
      },
      "applicationUrl": "http://localhost:54340/"
    },
    "Kestrel Staging": {
      "commandName": "Project",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_My_Environment": "1",
        "ASPNETCORE_DETAILEDERRORS": "1",
        "ASPNETCORE_ENVIRONMENT": "Staging"
      },
      "applicationUrl": "http://localhost:51997/"
    }
  }
}
```

#### NOTE

A propriedade `applicationUrl` no `launchSettings.json` pode especificar uma lista de URLs de servidores. Use um ponto e vírgula entre as URLs na lista:

```
"EnvironmentsSample": {
  "commandName": "Project",
  "launchBrowser": true,
  "applicationUrl": "https://localhost:5001;http://localhost:5000",
  "environmentVariables": {
    "ASPNETCORE_ENVIRONMENT": "Development"
  }
}
```

Quando o aplicativo é inicializado com `dotnet run`, o primeiro perfil com `"commandName": "Project"` é usado. O valor de `commandName` especifica o servidor Web a ser iniciado. `commandName` pode ser qualquer um dos seguintes:

- `IISExpress`
- `IIS`
- `Project` (que inicia o Kestrel)

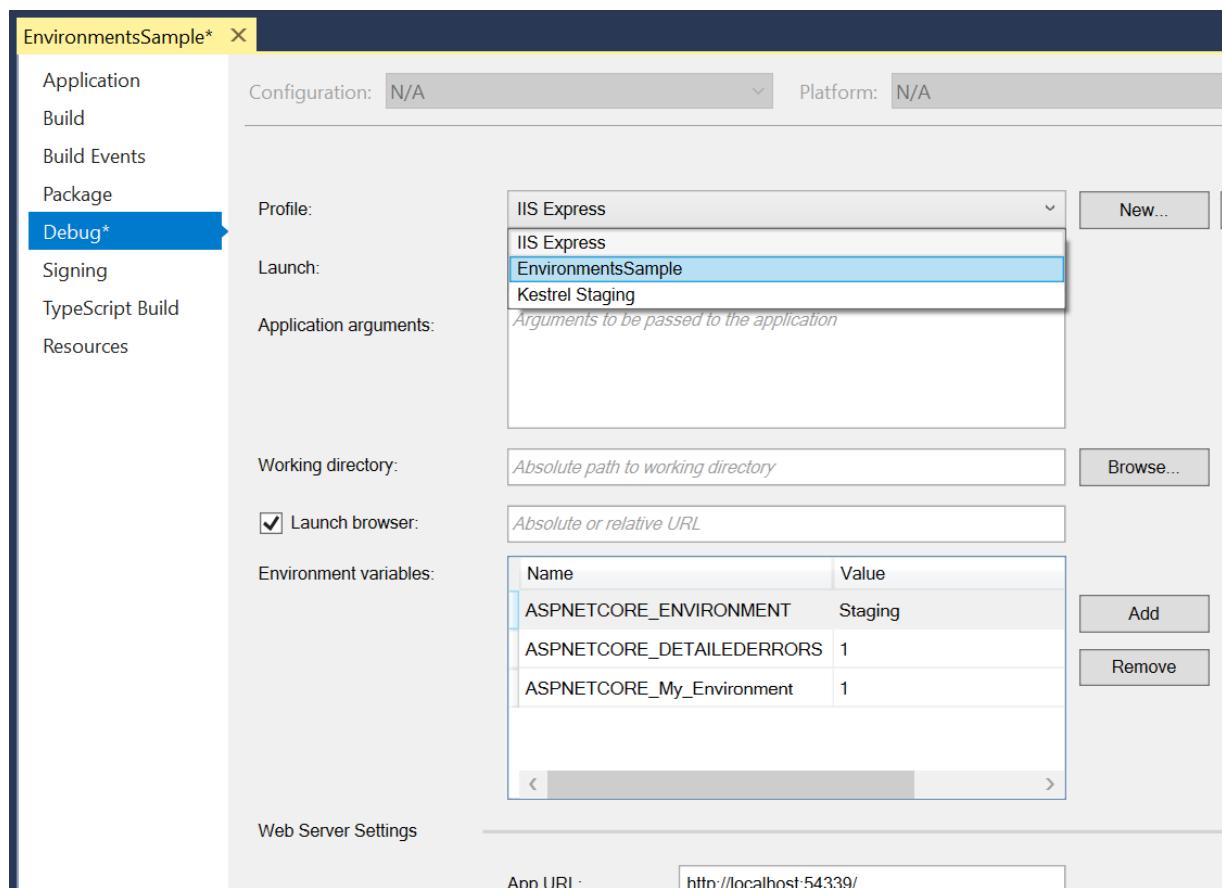
Quando um aplicativo for iniciado com `dotnet run`:

- `launchSettings.json` é lido, se está disponível. As configurações de `environmentVariables` em `launchSettings.json` substituem as variáveis de ambiente.
- O ambiente de hospedagem é exibido.

A saída a seguir mostra um aplicativo iniciado com `dotnet run`:

```
PS C:\Websites\EnvironmentsSample> dotnet run
Using launch settings from C:\Websites\EnvironmentsSample\Properties\launchSettings.json...
Hosting environment: Staging
Content root path: C:\Websites\EnvironmentsSample
Now listening on: http://localhost:54340
Application started. Press Ctrl+C to shut down.
```

A guia **Depurar** das propriedades do projeto do Visual Studio fornece uma GUI para editar o arquivo `launchSettings.json`:



As alterações feitas nos perfis do projeto poderão não ter efeito até que o servidor Web seja reiniciado. O Kestrel precisa ser reiniciado antes de detectar as alterações feitas ao seu ambiente.

#### **WARNING**

`launchSettings.json` não deve armazenar segredos. A [ferramenta Secret Manager](#) pode ser usado para armazenar segredos de desenvolvimento local.

Ao usar [Visual Studio Code](#), variáveis de ambiente podem ser definidas no arquivo `.vscode/launch.json`. O exemplo a seguir define o ambiente como `Development`:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": ".NET Core Launch (web)",
      ... additional VS Code configuration settings ...

      "env": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  ]
}
```

Um arquivo `.vscode/launch.json` do projeto não é lido ao iniciar o aplicativo com `dotnet run` da mesma maneira que `Properties/launchSettings.json`. Ao inicializar um aplicativo em desenvolvimento que não tem um arquivo `launchSettings.json`, defina o ambiente com uma variável de ambiente ou um argumento de linha de comando para o comando `dotnet run`.

## Produção

O ambiente de produção deve ser configurado para maximizar a segurança, o desempenho e a robustez do aplicativo. Algumas configurações comuns que são diferentes do desenvolvimento incluem:

- Cache.
- Recursos do lado do cliente são agrupados, minimizados e potencialmente atendidos por meio de uma CDN.
- Páginas de erro de diagnóstico desabilitadas.
- Páginas de erro amigáveis habilitadas.
- Log de produção e monitoramento habilitados. Por exemplo, [Application Insights](#).

## Definir o ambiente

Geralmente, é útil definir um ambiente específico para teste. Se o ambiente não for definido, ele usará `Production` como padrão, o que desabilitará a maioria dos recursos de depuração. O método para configurar o ambiente depende do sistema operacional.

### Serviço de Aplicativo do Azure

Para definir o ambiente no [Serviço de Aplicativo do Azure](#), execute as seguintes etapas:

1. Selecione o aplicativo na folha **Serviços de Aplicativos**.
2. No grupo **CONFIGURAÇÕES**, selecione a folha **Configurações do aplicativo**.
3. Na área **Configurações do aplicativo**, selecione **Adicionar nova configuração**.
4. Para **Inserir um nome**, forneça `ASPNETCORE_ENVIRONMENT`. Para **Inserir um valor**, fornecer o ambiente (por exemplo, `Staging`).
5. Marque a caixa de seleção **Configuração do Slot** se desejar que a configuração do ambiente permaneça no slot atual quando os slots de implantação forem trocados. Para obter mais informações, confira [Documentação do Azure: que configurações são trocadas?](#).
6. Selecione **Salvar** na parte superior da folha.

O Serviço de Aplicativo do Azure reinicia automaticamente o aplicativo após uma configuração de aplicativo (variável de ambiente) ser adicionada, alterada ou excluída no Portal do Azure.

### Windows

Para definir o `ASPNETCORE_ENVIRONMENT` para a sessão atual quando o aplicativo for iniciado usando `dotnet run`,

os comandos a seguir serão usados:

### Prompt de comando

```
set ASPNETCORE_ENVIRONMENT=Development
```

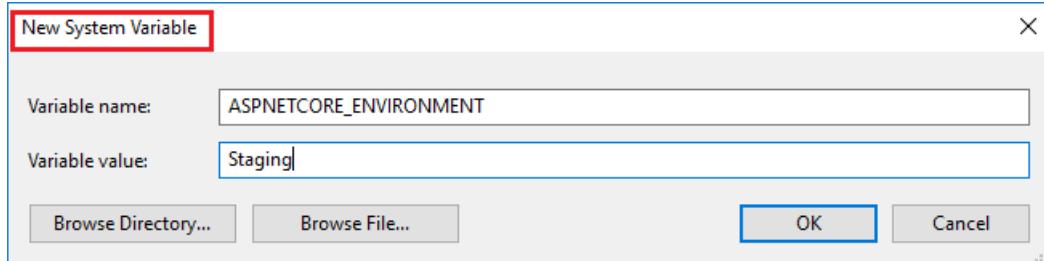
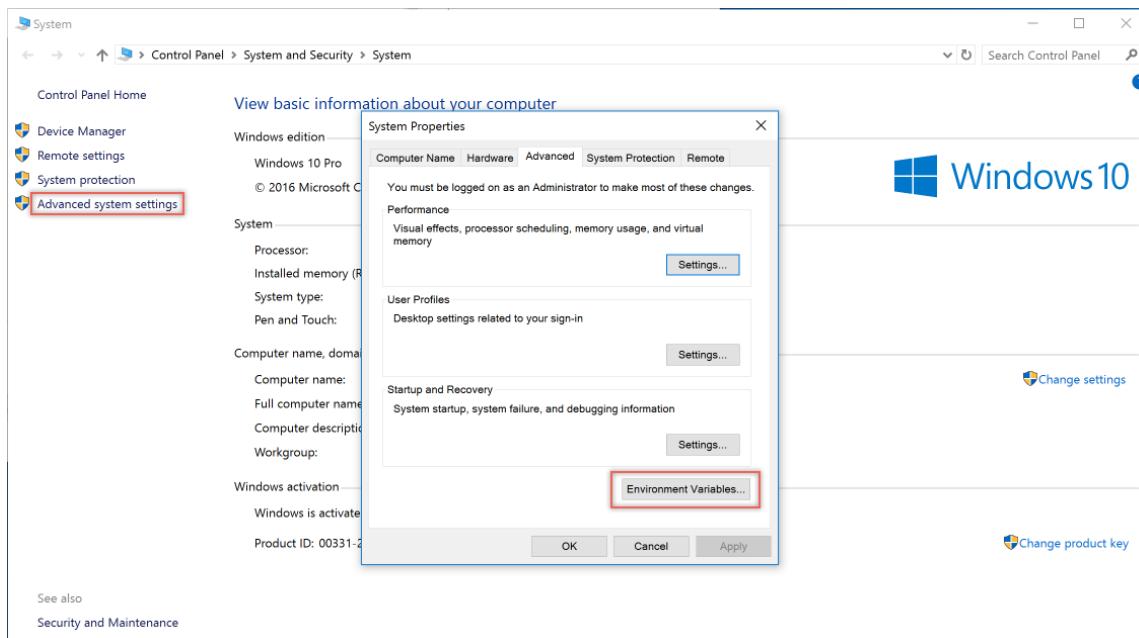
### PowerShell

```
$Env:ASPNETCORE_ENVIRONMENT = "Development"
```

Esses comandos somente têm efeito para a janela atual. Quando a janela é fechada, a configuração `ASPNETCORE_ENVIRONMENT` é revertida para a configuração padrão ou o valor de computador.

Para definir o valor globalmente no Windows, use uma das seguintes abordagens:

- Abra o **Painel de Controle > Sistema > Configurações avançadas do sistema** e adicione ou edite o valor `ASPNETCORE_ENVIRONMENT`:



- Abra um prompt de comando administrativo e use o comando `setx` ou abra um prompt de comando administrativo do PowerShell e use `[Environment]::SetEnvironmentVariable`:

### Prompt de comando

```
setx ASPNETCORE_ENVIRONMENT Development /M
```

O comutador `/M` indica para definir a variável de ambiente no nível do sistema. Se o comutador `/M` não for usado, a variável de ambiente será definida para a conta de usuário.

### PowerShell

```
[Environment]::SetEnvironmentVariable("ASPNETCORE_ENVIRONMENT", "Development", "Machine")
```

O valor da opção `Machine` indica para definir a variável de ambiente no nível do sistema. Se o valor da opção for alterado para `User`, a variável de ambiente será definida para a conta de usuário.

Quando a variável de ambiente `ASPNETCORE_ENVIRONMENT` é definida globalmente, ela entra em vigor para `dotnet run` em qualquer janela de comando aberta depois que o valor é definido.

## web.config

Para definir a variável de ambiente `ASPNETCORE_ENVIRONMENT` com `web.config`, consulte a seção *Definindo variáveis de ambiente* de [Módulo do ASP.NET Core](#). Quando a variável de ambiente `ASPNETCORE_ENVIRONMENT` é definida com `web.config`, seu valor substitui uma configuração no nível do sistema.

## Arquivo de projeto ou perfil de publicação

**Para implantações do Windows IIS:** Inclua a propriedade `<EnvironmentName>` no perfil de publicação (`.pubxml`) ou no arquivo de projeto. Esta abordagem define o ambiente no arquivo `web.config` quando o projeto é publicado:

```
<PropertyGroup>
  <EnvironmentName>Development</EnvironmentName>
</PropertyGroup>
```

## Por pool de aplicativos do IIS

Para definir a variável de ambiente `ASPNETCORE_ENVIRONMENT` para um aplicativo em execução em um Pool de aplicativos isolado (compatível com IIS 10.0 ou posterior), consulte a seção *Comando AppCmd.exe* do tópico [Variáveis de ambiente <environmentVariables>](#). Quando a variável de ambiente `ASPNETCORE_ENVIRONMENT` é definida para um pool de aplicativos, seu valor substitui uma configuração no nível do sistema.

### IMPORTANT

Ao hospedar um aplicativo no IIS e adicionar ou alterar a variável de ambiente `ASPNETCORE_ENVIRONMENT`, use qualquer uma das abordagens a seguir para que o novo valor seja escolhido por aplicativos:

- Execute `net stop was /y` seguido por `net start w3svc` em um prompt de comando.
- Reinicie o servidor.

## macOS

A configuração do ambiente atual para macOS pode ser feita em linha ao executar o aplicativo:

```
ASPNETCORE_ENVIRONMENT=Development dotnet run
```

Como alternativa, defina o ambiente com `export` antes de executar o aplicativo:

```
export ASPNETCORE_ENVIRONMENT=Development
```

As variáveis de ambiente no nível do computador são definidas no arquivo `.bashrc` ou `.bash_profile`. Edite o arquivo usando qualquer editor de texto. Adicione a seguinte instrução:

```
export ASPNETCORE_ENVIRONMENT=Development
```

## Linux

Para distribuições Linux, use o comando `export` no prompt de comando para as configurações de variável baseadas na sessão e o arquivo `bash_profile` para as configurações de ambiente no nível do computador.

## Configuração por ambiente

Para carregar a configuração por ambiente, recomendamos:

- Arquivos `appsettings` (\*`appsettings.<>.json`). Confira [Configuração: provedor de configuração do arquivo](#).
- Variáveis de ambiente (definidas em cada sistema em que o aplicativo está hospedado). Confira [Configuração: provedor de configuração do arquivo](#) e [Armazenamento seguro de segredos do aplicativo em desenvolvimento: variáveis de ambiente](#).
- Gerenciador de Segredo (somente no ambiente de desenvolvimento). Consulte [Armazenamento seguro dos segredos do aplicativo em desenvolvimento no ASP.NET Core](#).

## Métodos e classe Startup baseados no ambiente

### Convenções da classe Startup

Quando um aplicativo ASP.NET Core é iniciado, a `classe Startup` inicia o aplicativo. O aplicativo pode definir classes `Startup` separadas para ambientes diferentes (por exemplo, `StartupDevelopment`), e a classe `Startup` adequada é selecionada em tempo de execução. A classe cujo sufixo do nome corresponde ao ambiente atual é priorizada. Se uma classe `Startup{EnvironmentName}` correspondente não for encontrada, a classe `Startup` será usada.

Para implementar classes `Startup` com base em ambiente, crie uma classe `Startup{EnvironmentName}` para cada ambiente no uso e classe `Startup` de fallback:

```

// Startup class to use in the Development environment
public class StartupDevelopment
{
    public void ConfigureServices(IServiceCollection services)
    {
        ...
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        ...
    }
}

// Startup class to use in the Production environment
public class StartupProduction
{
    public void ConfigureServices(IServiceCollection services)
    {
        ...
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        ...
    }
}

// Fallback Startup class
// Selected if the environment doesn't match a Startup{EnvironmentName} class
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        ...
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        ...
    }
}

```

Use a sobrecarga [UseStartup \(IWebHostBuilder, String\)](#) que aceita um nome de assembly:

```

public static void Main(string[] args)
{
    CreateWebHostBuilder(args).Build().Run();
}

public static IWebHostBuilder CreateWebHostBuilder(string[] args)
{
    var assemblyName = typeof(Startup).GetTypeInfo().Assembly.FullName;

    return WebHost.CreateDefaultBuilder(args)
        .UseStartup(assemblyName);
}

```

```
public static void Main(string[] args)
{
    CreateWebHost(args).Run();
}

public static IWebHost CreateWebHost(string[] args)
{
    var assemblyName = typeof(Startup).GetTypeInfo().Assembly.FullName;

    return WebHost.CreateDefaultBuilder(args)
        .UseStartup(assemblyName)
        .Build();
}
```

```
public class Program
{
    public static void Main(string[] args)
    {
        var assemblyName = typeof(Startup).GetTypeInfo().Assembly.FullName;

        var host = new WebHostBuilder()
            .UseStartup(assemblyName)
            .Build();

        host.Run();
    }
}
```

## Convenções do método Startup

[Configure](#) e [ConfigureServices](#) são compatíveis com versões específicas do ambiente dos formatos

`Configure<EnvironmentName>` e `Configure<EnvironmentName>Services` :

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        StartupConfigureServices(services);
    }

    public void ConfigureStagingServices(IServiceCollection services)
    {
        StartupConfigureServices(services);
    }

    private void StartupConfigureServices(IServiceCollection services)
    {
        services.Configure<CookiePolicyOptions>(options =>
        {
            options.CheckConsentNeeded = context => true;
            options.MinimumSameSitePolicy = SameSiteMode.None;
        });

        services.AddMvc()
            .SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        if (env.IsProduction() || env.IsStaging() || env.IsEnvironment("Staging_2"))
        {
            app.UseExceptionHandler("/Error");
            app.UseHsts();
        }

        app.UseHttpsRedirection();
        app.UseStaticFiles();
        app.UseCookiePolicy();
        app.UseMvc();
    }

    public void ConfigureStaging(IApplicationBuilder app, IHostingEnvironment env)
    {
        if (!env.IsStaging())
        {
            throw new Exception("Not staging.");
        }

        app.UseExceptionHandler("/Error");
        app.UseHsts();
        app.UseHttpsRedirection();
        app.UseStaticFiles();
        app.UseCookiePolicy();
        app.UseMvc();
    }
}
```

## Recursos adicionais

- Inicialização de aplicativo no ASP.NET Core
- Configuração no ASP.NET Core
- `IHostingEnvironment.EnvironmentName`

# Configuração no ASP.NET Core

28/01/2019 • 66 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

A configuração de aplicativos no ASP.NET Core se baseia em pares chave-valor estabelecidos por *provedores de configuração*. Os provedores de configuração leem os dados de configuração em pares chave-valor de várias fontes de configuração:

- Azure Key Vault
  - Argumentos de linha de comando
  - Provedores personalizados (instalados ou criados)
  - Arquivos de diretório
  - Variáveis de ambiente
  - Objetos do .NET na memória
  - Arquivos de configurações
- 
- Azure Key Vault
  - Argumentos de linha de comando
  - Provedores personalizados (instalados ou criados)
  - Variáveis de ambiente
  - Objetos do .NET na memória
  - Arquivos de configurações
- 
- Argumentos de linha de comando
  - Provedores personalizados (instalados ou criados)
  - Variáveis de ambiente
  - Objetos do .NET na memória
  - Arquivos de configurações

O *padrão de opções* é uma extensão dos conceitos de configuração descritos neste tópico. As opções usam classes para representar grupos de configurações relacionadas. Para saber mais sobre como usar o padrão de opções, confira [Padrão de opções no ASP.NET Core](#).

[Exibir ou baixar código de exemplo \(como baixar\)](#)

Esses três pacotes estão incluídos no [metapacote Microsoft.AspNetCore.App](#).

Esses três pacotes estão incluídos no [metapacote Microsoft.AspNetCore.All](#).

## Configuração do host versus aplicativo

Antes do aplicativo ser configurado e iniciado, um *host* é configurado e iniciado. O host é responsável pelo gerenciamento de tempo de vida e pela inicialização do aplicativo. O aplicativo e o host são configurados usando os provedores de configuração descritos neste tópico. Os pares chave-valor de configuração do host se tornam parte da configuração global do aplicativo. Para saber mais sobre como os provedores de configuração são usados quando o host for compilado, e como as fontes de configuração afetam o host e a configuração, confira [Host da Web e Host Genérico no ASP.NET Core](#).

## Configuração padrão

Aplicativos Web baseados em modelos `dotnet new` do ASP.NET Core chamam `CreateDefaultBuilder` ao criar um host. `CreateDefaultBuilder` fornece a configuração padrão para o aplicativo.

- A configuração do host é fornecida de:
  - Variáveis de ambiente prefixadas com `ASPNETCORE_` (por exemplo, `ASPNETCORE_ENVIRONMENT`) usando o [Provedor de Configuração de Variáveis de Ambiente](#).
  - Argumentos de linha de comando usando o [Provedor de Configuração de Linha de Comando](#).
- A configuração do aplicativo é fornecida de (na seguinte ordem):
  - `appsettings.json` usando o [Provedor de Configuração do Arquivo](#).
  - `appsettings.{Environment}.json` usando o [Provedor de Configuração do Arquivo](#).
  - [Gerenciador de Segredo](#) quando o aplicativo é executado no ambiente `Development` usando o assembly de entrada.
  - Variáveis de ambiente usando o [Provedor de Configuração de Variáveis de Ambiente](#).
  - Argumentos de linha de comando usando o [Provedor de Configuração de Linha de Comando](#).

Os provedores de configuração são explicados posteriormente neste tópico. Para obter mais informações sobre o host e `CreateDefaultBuilder`, consulte [Host da Web do ASP.NET Core](#).

## Segurança

Adote as melhores práticas a seguir:

- Nunca armazene senhas ou outros dados confidenciais no código do provedor de configuração ou nos arquivos de configuração de texto sem formatação.
- Não use segredos de produção em ambientes de teste ou de desenvolvimento.
- Especifique segredos fora do projeto para que eles não sejam accidentalmente comprometidos com um repositório de código-fonte.

Saiba mais sobre [como usar vários ambientes](#) e gerenciar o [armazenamento seguro de segredos de aplicativo em desenvolvimento](#) com o [Gerenciador de Segredo](#) (inclui recomendações sobre como usar variáveis de ambiente para armazenar dados confidenciais). O Gerenciador de Segredo usa o Provedor de Configuração de Arquivo para armazenar segredos do usuário em um arquivo JSON no sistema local. O Provedor de Configuração de Arquivo será descrito mais adiante neste tópico.

O [Azure Key Vault](#) é uma opção para o armazenamento seguro de segredos do aplicativo. Para obter mais informações, consulte [Provedor de configuração do Cofre de chaves do Azure no ASP.NET Core](#).

## Dados de configuração hierárquica

A API de Configuração é capaz de manter dados de configuração hierárquica nivelando os dados hierárquicos com o uso de um delimitador nas chaves de configuração.

No seguinte arquivo JSON, há quatro chaves em uma hierarquia estruturada com duas seções:

```
{  
    "section0": {  
        "key0": "value",  
        "key1": "value"  
    },  
    "section1": {  
        "key0": "value",  
        "key1": "value"  
    }  
}
```

Quando o arquivo é lido na configuração, ocorre a criação de chaves exclusivas para manter a estrutura hierárquica de dados original da fonte de configuração. As seções e as chaves são niveladas usando dois-pontos ( `:` ) para manter a estrutura original:

- `section0:key0`
- `section0:key1`
- `section1:key0`
- `section1:key1`

Os métodos `GetSection` e `GetChildren` estão disponíveis para isolar as seções e os filhos de uma seção nos dados de configuração. Esses métodos serão descritos posteriormente em [GetSection](#), [GetChildren](#) e [Exists](#). `GetSection` está no pacote [Microsoft.Extensions.Configuration](#), que está no metapacote [Microsoft.AspNetCore.App](#).

## Convenções

Na inicialização do aplicativo, as fontes de configuração são lidas na ordem especificada pelos provedores de configuração.

Os Provedores de Configuração de Arquivo têm a capacidade de recarregar a configuração quando um arquivo de configurações subjacente é alterado após a inicialização do aplicativo. O Provedor de Configuração de Arquivo será descrito mais adiante neste tópico.

`IConfiguration` está disponível no contêiner [DI \(injeção de dependência\)](#) do aplicativo. Os provedores de configuração não podem utilizar a DI, pois ela não é disponibilizada quando eles são configurados pelo host.

As chaves de configuração adotam as convenções a seguir:

- As chaves não diferenciam maiúsculas de minúsculas. Por exemplo, `ConnectionString` e `connectionstring` são tratados como chaves equivalentes.
- Se um valor para a mesma chave for definido pelos mesmos provedores de configuração, ou por outros, o último valor definido na chave será o valor usado.
- Chaves hierárquicas
  - Ao interagir com a API de configuração, um separador de dois-pontos ( `:` ) funciona em todas as plataformas.
  - Nas variáveis de ambiente, talvez um separador de dois-pontos não funcione em todas as plataformas. Um sublinhado duplo ( `__` ) é compatível com todas as plataformas e é convertido em dois-pontos.
  - No Azure Key Vault, as chaves hierárquicas usam `--` (dois traços) como separador. Você deve fornecer o código para substituir os traços por dois-pontos quando os segredos forem carregados na configuração do aplicativo.
- O `ConfigurationBinder` dá suporte a matrizes de associação para objetos usando os índices em chaves de configuração. A associação de matriz está descrita na seção [Associar uma matriz a uma](#)

classe.

Os valores de configuração adotam as convenções a seguir:

- Os valores são cadeias de caracteres.
- Não é possível armazenar valores nulos na configuração ou associá-los a objetos.

## Provedores

A tabela a seguir mostra os provedores de configuração disponíveis para aplicativos ASP.NET Core.

PROVIDER	FORNECE A CONFIGURAÇÃO DE ...
<a href="#">Provedor de Configuração do Azure Key Vault (tópicos de Segurança)</a>	Azure Key Vault
<a href="#">Provedor de Configuração de Linha de Comando</a>	Parâmetros de linha de comando
<a href="#">Provedor de Configuração personalizado</a>	Fonte personalizada
<a href="#">Provedor de Configuração de Variáveis de Ambiente</a>	Variáveis de ambiente
<a href="#">Provedor de Configuração de Arquivo</a>	Arquivos (INI, JSON, XML)
<a href="#">Provedor de Configuração de Chave por Arquivo</a>	Arquivos de diretório
<a href="#">Provedor de Configuração de Memória</a>	Coleções na memória
<a href="#">Segredos do usuário (Gerenciador de Segredo) (tópicos de Segurança)</a>	Arquivo no diretório de perfil do usuário
PROVIDER	FORNECE A CONFIGURAÇÃO DE ...
<a href="#">Provedor de Configuração do Azure Key Vault (tópicos de Segurança)</a>	Azure Key Vault
<a href="#">Provedor de Configuração de Linha de Comando</a>	Parâmetros de linha de comando
<a href="#">Provedor de Configuração personalizado</a>	Fonte personalizada
<a href="#">Provedor de Configuração de Variáveis de Ambiente</a>	Variáveis de ambiente
<a href="#">Provedor de Configuração de Arquivo</a>	Arquivos (INI, JSON, XML)
<a href="#">Provedor de Configuração de Memória</a>	Coleções na memória
<a href="#">Segredos do usuário (Gerenciador de Segredo) (tópicos de Segurança)</a>	Arquivo no diretório de perfil do usuário
PROVIDER	FORNECE A CONFIGURAÇÃO DE ...
<a href="#">Provedor de Configuração de Linha de Comando</a>	Parâmetros de linha de comando

PROVIDER	FORNECE A CONFIGURAÇÃO DE ...
Provedor de Configuração personalizado	Fonte personalizada
Provedor de Configuração de Variáveis de Ambiente	Variáveis de ambiente
Provedor de Configuração de Arquivo	Arquivos (INI, JSON, XML)
Provedor de Configuração de Memória	Coleções na memória
Segredos do usuário (Gerenciador de Segredo) (tópicos de Segurança)	Arquivo no diretório de perfil do usuário

Na inicialização, as fontes de configuração são lidas na ordem especificada pelos provedores de configuração. Os provedores de configuração descritos neste tópico estão descritos em ordem alfabética, não na ordem na qual seu código pode organizá-los. Organize os provedores de configuração em seu código para atender às suas prioridades para as fontes de configuração subjacentes.

Uma sequência comum de provedores de configuração é:

1. Arquivos (`appsettings.json`, `appsettings.{Environment}.json`, em que `{Environment}` é o ambiente de hospedagem atual do aplicativo)
2. [Azure Key Vault](#)
3. [Segredos do usuário \(Gerenciador de Segredo\)](#) (apenas no ambiente de desenvolvimento)
4. Variáveis de ambiente
5. Argumentos de linha de comando

É comum posicionar o Provedor de Configuração de Linha de Comando por último em uma série de provedores, a fim de permitir que os argumentos de linha de comando substituam a configuração definida por outros provedores.

Essa sequência de provedores é aplicada quando você inicializa um novo [WebHostBuilder](#) com [CreateDefaultBuilder](#). Para obter mais informações, confira [Host da Web: Configurar um host](#).

Essa sequência de provedores pode ser criada para o aplicativo (não o host) com um [ConfigurationBuilder](#) e uma chamada para seu método `Build` em `Startup`:

```
public Startup(IHostingEnvironment env)
{
    var builder = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
        .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true,
            reloadOnChange: true);

    var appAssembly = Assembly.Load(new AssemblyName(env.ApplicationName));

    if (appAssembly != null)
    {
        builder.AddUserSecrets(appAssembly, optional: true);
    }

    builder.AddEnvironmentVariables();

    Configuration = builder.Build();
}

public IConfiguration Configuration { get; }

public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton< IConfiguration>(Configuration);
}
```

No exemplo anterior, o nome do ambiente (`env.EnvironmentName`) e o nome do assembly de aplicativo (`env.ApplicationName`) são fornecidos pelo `IHostingEnvironment`. Para obter mais informações, consulte [Usar vários ambientes no ASP.NET Core](#). O caminho base é definido com `SetBasePath`. `SetBasePath` está no pacote [Microsoft.Extensions.Configuration.FileExtensions](#), que está no metapacote [Microsoft.AspNetCore.App](#).

## ConfigureAppConfiguration

Chame `ConfigureAppConfiguration` ao criar o host para especificar os provedores de configuração do aplicativo, além daqueles adicionados automaticamente por `CreateDefaultBuilder`:

```

public class Program
{
    public static Dictionary<string, string> arrayDict = new Dictionary<string, string>
    {
        {"array:entries:0", "value0"},
        {"array:entries:1", "value1"},
        {"array:entries:2", "value2"},
        {"array:entries:4", "value4"},
        {"array:entries:5", "value5"}
    };

    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .ConfigureAppConfiguration((hostingContext, config) =>
            {
                config.SetBasePath(Directory.GetCurrentDirectory());
                config.AddInMemoryCollection(arrayDict);
                config.AddJsonFile("json_array.json", optional: false, reloadOnChange: false);
                config.AddJsonFile("starship.json", optional: false, reloadOnChange: false);
                config.AddXmlFile("tvshow.xml", optional: false, reloadOnChange: false);
                config.AddEFConfiguration(options => options.UseInMemoryDatabase("InMemoryDb"));
                config.AddCommandLine(args);
            })
            .UseStartup<Startup>();
    }
}

```

## Provedor de Configuração de Linha de Comando

O [CommandLineConfigurationProvider](#) carrega a configuração dos pares chave-valor do argumento de linha de comando em tempo de execução.

Para ativar a configuração de linha de comando, o método de extensão [AddCommandLine](#) é chamado em uma instância do [ConfigurationBuilder](#).

`AddCommandLine` é chamado automaticamente quando você inicializa um novo [WebHostBuilder](#) com [CreateDefaultBuilder](#). Para obter mais informações, confira [Host da Web: Configurar um host](#).

`CreateDefaultBuilder` também carrega:

- Configuração opcional de `appsettings.json` e `appsettings.{Environment}.json`.
- [Segredos do usuário \(Gerenciador de Segredo\)](#) (no ambiente de desenvolvimento).
- Variáveis de ambiente.

`CreateDefaultBuilder` adiciona o Provedor de Configuração de Linha de Comando por último. Os argumentos de linha de comando passados em tempo de execução substituem a configuração definida por outros provedores.

`CreateDefaultBuilder` age quando o host é construído. Portanto, a configuração de linha de comando ativada por `CreateDefaultBuilder` pode afetar como o host é configurado.

Chame [ConfigureAppConfiguration](#) ao criar o host para especificar a configuração do aplicativo.

`AddCommandLine` já foi chamado por `CreateDefaultBuilder`. Se precisar oferecer configuração de aplicativo e ainda poder substituir essa configuração por argumentos de linha de comando, chame os provedores adicionais do aplicativo em [ConfigureAppConfiguration](#) e chame `AddCommandLine` por fim.

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .ConfigureAppConfiguration((hostingContext, config) =>
            {
                // Call other providers here and call AddCommandLine last.
                config.AddCommandLine(args);
            })
            .UseStartup<Startup>();
}
```

Ao criar um [WebHostBuilder](#) diretamente, chame [UseConfiguration](#) com a configuração:

Aplice a configuração ao [WebHostBuilder](#) com o método [UseConfiguration](#).

`AddCommandLine` já foi chamado por `CreateDefaultBuilder` quando `UseConfiguration` é chamado. Se precisar oferecer configuração de aplicativo e ainda poder substituir essa configuração por argumentos de linha de comando, chame os provedores adicionais do aplicativo em um `ConfigurationBuilder` e chame `AddCommandLine` por fim.

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args)
    {
        var config = new ConfigurationBuilder()
            // Call other providers here and call AddCommandLine last.
            .AddCommandLine(args)
            .Build();

        return WebHost.CreateDefaultBuilder(args)
            .UseConfiguration(config)
            .UseStartup<Startup>();
    }
}
```

Ao criar um [WebHostBuilder](#) diretamente, chame [UseConfiguration](#) com a configuração:

Para ativar a configuração de linha de comando, chame o método de extensão [AddCommandLine](#) em uma instância do [ConfigurationBuilder](#).

Chamar o provedor por último permite que os argumentos de linha de comando passados em tempo de execução substituam a configuração definida por outros provedores de configuração.

Aplice a configuração ao [WebHostBuilder](#) com o método [UseConfiguration](#):

```

var config = new ConfigurationBuilder()
    // Call additional providers here as needed.
    // Call AddCommandLine last to allow arguments to override other configuration.
    .AddCommandLine(args)
    .Build();

var host = new WebHostBuilder()
    .UseConfiguration(config)
    .UseKestrel()
    .UseStartup<Startup>();

```

## Exemplo

O aplicativo de exemplo 2.x aproveita a vantagem do método de conveniência estático `CreateDefaultBuilder` para criar o host, que inclui uma chamada para `AddCommandLine`.

O aplicativo de exemplo 1.x chama `AddCommandLine` em um `ConfigurationBuilder`.

1. Abra um prompt de comando no diretório do projeto.
2. Forneça um argumento de linha de comando para o comando `dotnet run`,  
`dotnet run CommandLineKey=CommandLineValue`.
3. Quando o aplicativo estiver em execução, abra um navegador para o aplicativo em  
`http://localhost:5000`.
4. Observe que a saída contém o par chave-valor do argumento de linha de comando de configuração fornecido para `dotnet run`.

## Arguments

O valor deve vir após um sinal de igual (`=`), ou a chave deve ter um prefixo (`--` ou `/`) quando o valor vier após um espaço. O valor pode ser nulo se um sinal de igual for usado (por exemplo, `CommandLineKey=`).

PREFIXO DA CHAVE	EXEMPLO
Nenhum prefixo	<code>CommandLineKey1=value1</code>
Dois traços ( <code>--</code> )	<code>--CommandLineKey2=value2</code> , <code>--CommandLineKey2 value2</code>
Barra ( <code>/</code> )	<code>/CommandLineKey3=value3</code> , <code>/CommandLineKey3 value3</code>

No mesmo comando, não combine pares chave-valor do argumento de linha de comando que usam um sinal de igual com pares chave-valor que usam um espaço.

Exemplo de comandos:

```

dotnet run CommandLineKey1=value1 --CommandLineKey2=value2 /CommandLineKey3=value3
dotnet run --CommandLineKey1 value1 /CommandLineKey2 value2
dotnet run CommandLineKey1= CommandLineKey2=value2

```

## Mapeamentos de comutador

Os mapeamentos de comutador permitem fornecer a lógica de substituição do nome da chave. Quando você cria manualmente a configuração com um `ConfigurationBuilder`, pode fornecer um dicionário de substituições de opções para o método `AddCommandLine`.

Ao ser usado, o dicionário de mapeamentos de comutador é verificado para oferecer uma chave que corresponda à chave fornecida por um argumento de linha de comando. Se a chave de linha de comando for encontrada no dicionário, o valor do dicionário (a substituição da chave) será passado de volta para definir o par chave-valor na configuração do aplicativo. Um mapeamento de comutador é necessário para qualquer chave de linha de comando prefixada com um traço único (-).

Regras de chave do dicionário de mapeamentos de comutador:

- Os comutadores devem começar com um traço (-) ou traço duplo (--).
- O dicionário de mapeamentos de comutador chave não deve conter chaves duplicadas.

Chame [ConfigureAppConfiguration](#) ao criar o host para especificar a configuração do aplicativo:

```
public class Program
{
    public static readonly Dictionary<string, string> _switchMappings =
        new Dictionary<string, string>
    {
        { "-CLKey1", "CommandLineKey1" },
        { "-CLKey2", "CommandLineKey2" }
    };

    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    // Do not pass the args to CreateDefaultBuilder
    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder()
            .ConfigureAppConfiguration((hostingContext, config) =>
            {
                config.AddCommandLine(args, _switchMappings);
            })
            .UseStartup<Startup>();
}
```

Conforme mostrado no exemplo anterior, a chamada para `CreateDefaultBuilder` não deve passar argumentos ao usar mapeamentos de opção. A chamada `AddCommandLine` do método `CreateDefaultBuilder` não inclui opções mapeadas, e não é possível passar o dicionário de mapeamento de opções para `CreateDefaultBuilder`. Se os argumentos incluírem uma opção mapeada e forem passados para `CreateDefaultBuilder`, o provedor `AddCommandLine` não inicializará com um `FormatException`. A solução não é passar os argumentos para `CreateDefaultBuilder`, mas, em vez disso, permitir que o método `AddCommandLine` do método `ConfigurationBuilder` processe os dois argumentos e o dicionário de mapeamento de opções.

```

public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args)
    {
        var switchMappings = new Dictionary<string, string>
        {
            { "-CLKey1", "CommandLineKey1" },
            { "-CLKey2", "CommandLineKey2" }
        };

        var config = new ConfigurationBuilder()
            .AddCommandLine(args, switchMappings)
            .Build();

        // Do not pass the args to CreateDefaultBuilder
        return WebHost.CreateDefaultBuilder()
            .UseConfiguration(config)
            .UseStartup<Startup>();
    }
}

```

Conforme mostrado no exemplo anterior, a chamada para `CreateDefaultBuilder` não deve passar argumentos ao usar mapeamentos de opção. A chamada `AddCommandLine` do método `CreateDefaultBuilder` não inclui opções mapeadas, e não é possível passar o dicionário de mapeamento de opções para `CreateDefaultBuilder`. Se os argumentos incluírem uma opção mapeada e forem passados para `CreateDefaultBuilder`, o provedor `AddCommandLine` não inicializará com um `FormatException`. A solução não é passar os argumentos para `CreateDefaultBuilder`, mas, em vez disso, permitir que o método `AddCommandLine` do método `ConfigurationBuilder` processe os dois argumentos e o dicionário de mapeamento de opções.

```

public static void Main(string[] args)
{
    var switchMappings = new Dictionary<string, string>
    {
        { "-CLKey1", "CommandLineKey1" },
        { "-CLKey2", "CommandLineKey2" }
    };

    var config = new ConfigurationBuilder()
        .AddCommandLine(args, switchMappings)
        .Build();

    var host = new WebHostBuilder()
        .UseConfiguration(config)
        .UseKestrel()
        .UseStartup<Startup>()
        .Start();

    using (host)
    {
        Console.ReadLine();
    }
}

```

Depois que o dicionário de mapeamentos de comutador for criado, ele conterá os dados mostrados na tabela a seguir.

CHAVE	VALOR
-CLKey1	CommandLineKey1
-CLKey2	CommandLineKey2

Se as chaves mapeadas para opção forem usadas ao iniciar o aplicativo, a configuração receberá o valor de configuração na chave fornecida pelo dicionário:

```
dotnet run -CLKey1=value1 -CLKey2=value2
```

Após a execução do comando anterior, a configuração conterá os valores mostrados na tabela a seguir.

CHAVE	VALOR
CommandLineKey1	value1
CommandLineKey2	value2

## Provedor de Configuração de Variáveis de Ambiente

O [EnvironmentVariablesConfigurationProvider](#) carrega a configuração dos pares chave-valor da variável de ambiente em tempo de execução.

Para ativar a configuração das variáveis de ambiente, chame o método de extensão [AddEnvironmentVariables](#) em uma instância do [ConfigurationBuilder](#).

Ao trabalhar com chaves hierárquicas nas variáveis de ambiente, talvez um separador de dois-pontos (`:`) não funcione em todas as plataformas. Um sublinhado duplo (`__`) é compatível com todas as plataformas e é substituído por dois-pontos.

O [Serviço de Aplicativo do Azure](#) permite que você defina variáveis de ambiente no portal do Azure que podem substituir a configuração do aplicativo usando o Provedor de Configuração de Variáveis de Ambiente. Para obter mais informações, confira [Aplicativos Azure: Substituir a configuração do aplicativo no Portal do Azure](#).

`AddEnvironmentVariables` é chamado automaticamente para as variáveis de ambiente prefixadas com `ASPNETCORE_` quando você inicializa um novo [WebHostBuilder](#). Para obter mais informações, confira [Host da Web: Configurar um host](#).

`CreateDefaultBuilder` também carrega:

- Configuração de aplicativo em variáveis de ambiente sem prefixos chamando `AddEnvironmentVariables` sem um prefixo.
- Configuração opcional de `appsettings.json` e `appsettings.{Environment}.json`.
- [Segredos do usuário \(Gerenciador de Segredo\)](#) (no ambiente de desenvolvimento).
- Argumentos de linha de comando.

O Provedor de Configuração de Variáveis de Ambiente é chamado depois que a configuração é estabelecida por meio dos segredos do usuário e dos arquivos `appsettings`. Chamar o provedor nessa posição permite que as variáveis de ambiente sejam lidas em tempo de execução para substituir a configuração definida por segredos do usuário e arquivos `appsettings`.

Chame [ConfigureAppConfiguration](#) ao criar o host para especificar a configuração do aplicativo.

Se precisar fornecer configuração do aplicativo com base em variáveis de ambiente adicionais, chame os provedores adicionais do aplicativo no [ConfigureAppConfiguration](#) e chame `AddEnvironmentVariables` com o prefixo.

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .ConfigureAppConfiguration((hostingContext, config) =>
            {
                // Call additional providers here as needed.
                // Call AddEnvironmentVariables last if you need to allow environment
                // variables to override values from other providers.
                config.AddEnvironmentVariables(prefix: "PREFIX_");
            })
            .UseStartup<Startup>();
}
```

Ao criar um [WebHostBuilder](#) diretamente, chame [UseConfiguration](#) com a configuração:

Chame o método de extensão `AddEnvironmentVariables` em uma instância de [ConfigurationBuilder](#). Aplique a configuração ao [WebHostBuilder](#) com o método [UseConfiguration](#).

Se precisar fornecer configuração do aplicativo com base em variáveis de ambiente adicionais, chame os provedores adicionais do aplicativo no [ConfigureAppConfiguration](#) e chame `AddEnvironmentVariables` com o prefixo.

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args)
    {
        var config = new ConfigurationBuilder()
            // Call additional providers here as needed.
            // Call AddEnvironmentVariables last if you need to allow environment
            // variables to override values from other providers.
            .AddEnvironmentVariables(prefix: "PREFIX_")
            .Build();

        return WebHost.CreateDefaultBuilder(args)
            .UseConfiguration(config)
            .UseStartup<Startup>();
    }
}
```

Ao criar um [WebHostBuilder](#) diretamente, chame [UseConfiguration](#) com a configuração:

Aplice a configuração ao [WebHostBuilder](#) com o método [UseConfiguration](#):

```
var config = new ConfigurationBuilder()
    .AddEnvironmentVariables()
    .Build();

var host = new WebHostBuilder()
    .UseConfiguration(config)
    .UseKestrel()
    .UseStartup<Startup>();
```

## Exemplo

O aplicativo de exemplo 2.x aproveita a vantagem do método de conveniência estático `CreateDefaultBuilder` para criar o host, que inclui uma chamada para `AddEnvironmentVariables`.

O aplicativo de exemplo 1.x chama `AddEnvironmentVariables` em um `ConfigurationBuilder`.

1. Execute o aplicativo de exemplo. Abra um navegador para o aplicativo em <http://localhost:5000>.
2. Observe que a saída contém o par chave-valor da variável de ambiente `ENVIRONMENT`. O valor reflete o ambiente no qual o aplicativo está em execução, normalmente `Development` ao executar localmente.

Para encurtar a lista de variáveis de ambiente renderizadas pelo aplicativo, o aplicativo filtra as variáveis de ambiente para mostrar as que começam com o seguinte:

- `ASPNETCORE_`
- `urls`
- Registrando em log
- `AMBIENTE`
- `contentRoot`
- `AllowedHosts`
- `applicationName`
- `CommandLine`

Se você quiser expor todas as variáveis de ambiente disponíveis para o aplicativo, altere o `FilteredConfiguration` em `Pages/Index.cshtml.cs` para o seguinte:

Se você quiser expor todas as variáveis de ambiente disponíveis para o aplicativo, altere o `FilteredConfiguration` em `Controllers/HomeController.cs` para o seguinte:

```
FilteredConfiguration = _config.AsEnumerable();
```

## Prefixos

Variáveis de ambiente carregadas na configuração do aplicativo são filtradas quando você fornece um prefixo para o método `AddEnvironmentVariables`. Por exemplo, para filtrar as variáveis de ambiente no prefixo `CUSTOM_`, forneça o prefixo para o provedor de configuração:

```
var config = new ConfigurationBuilder()
    .AddEnvironmentVariables("CUSTOM_")
    .Build();
```

O prefixo é removido na criação dos pares chave-valor de configuração.

O método de conveniência estático `CreateDefaultBuilder` cria um `WebHostBuilder` para estabelecer o host do aplicativo. Quando `WebHostBuilder` é criado, ele encontra a configuração de seu host nas

variáveis de ambiente prefixadas com `ASPNETCORE_`.

### Prefixos de cadeia de conexão

A API de configuração tem regras de processamento especiais para quatro variáveis de ambiente de cadeia de conexão envolvidas na configuração de cadeias de conexão do Azure para o ambiente de aplicativo. As variáveis de ambiente com os prefixos mostrados na tabela são carregadas no aplicativo se nenhum prefixo for fornecido para `AddEnvironmentVariables`.

PREFIXO DA CADEIA DE CONEXÃO	PROVIDER
<code>CUSTOMCONNSTR_</code>	Provedor personalizado
<code>MYSQLCONNSTR_</code>	MySQL
<code>SQLAZURECONNSTR_</code>	Banco de Dados SQL do Azure
<code>SQLCONNSTR_</code>	SQL Server

Quando uma variável de ambiente for descoberta e carregada na configuração com qualquer um dos quatro prefixos mostrados na tabela:

- A chave de configuração é criada removendo o prefixo da variável de ambiente e adicionando uma seção de chave de configuração (`ConnectionStrings`).
- Um novo par chave-valor de configuração é criado para representar o provedor de conexão de banco de dados (exceto para `CUSTOMCONNSTR_`, que não tem um provedor indicado).

CHAVE DE VARIÁVEL DE AMBIENTE	CHAVE DE CONFIGURAÇÃO CONVERTIDA	ENTRADA DE CONFIGURAÇÃO DO PROVEDOR
<code>CUSTOMCONNSTR_&lt;KEY&gt;</code>	<code>ConnectionStrings:&lt;KEY&gt;</code>	Entrada de configuração não criada.
<code>MYSQLCONNSTR_&lt;KEY&gt;</code>	<code>ConnectionStrings:&lt;KEY&gt;</code>	Chave: <code>ConnectionStrings:&lt;KEY&gt;_ProviderName</code> : Valor: <code>MySql.Data.MySqlClient</code>
<code>SQLAZURECONNSTR_&lt;KEY&gt;</code>	<code>ConnectionStrings:&lt;KEY&gt;</code>	Chave: <code>ConnectionStrings:&lt;KEY&gt;_ProviderName</code> : Valor: <code>System.Data.SqlClient</code>
<code>SQLCONNSTR_&lt;KEY&gt;</code>	<code>ConnectionStrings:&lt;KEY&gt;</code>	Chave: <code>ConnectionStrings:&lt;KEY&gt;_ProviderName</code> : Valor: <code>System.Data.SqlClient</code>

## Provedor de Configuração de Arquivo

[FileConfigurationProvider](#) é a classe base para carregamento da configuração do sistema de arquivos. Os provedores de configuração a seguir são dedicados a tipos específicos de arquivos:

- [Provedor de Configuração INI](#)

- [Provedor de Configuração JSON](#)
- [Provedor de Configuração XML](#)

### Provedor de Configuração INI

O [IniConfigurationProvider](#) carrega a configuração de pares chave-valor do arquivo INI em tempo de execução.

Para ativar a configuração de arquivo INI, chame o método de extensão [AddIniFile](#) em uma instância do [ConfigurationBuilder](#).

Os dois-pontos podem ser usados como um delimitador de seção na configuração do arquivo INI.

As sobrecargas permitem especificar:

- Se o arquivo é opcional.
- Se a configuração será recarregada caso o arquivo seja alterado.
- O [IFileProvider](#) usado para acessar o arquivo.

Chame [ConfigureAppConfiguration](#) ao criar o host para especificar a configuração do aplicativo:

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .ConfigureAppConfiguration((hostingContext, config) =>
            {
                config.SetBasePath(Directory.GetCurrentDirectory());
                config.AddIniFile("config.ini", optional: true, reloadOnChange: true);
            })
            .UseStartup<Startup>();
}
```

O caminho base é definido com [SetBasePath](#). [SetBasePath](#) está no pacote [Microsoft.Extensions.Configuration.FileExtensions](#), que está no metapacote [Microsoft.AspNetCore.App](#).

Ao criar um [WebHostBuilder](#) diretamente, chame [UseConfiguration](#) com a configuração:

Ao chamar [CreateDefaultBuilder](#), chame [UseConfiguration](#) com a configuração:

```

public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args)
    {
        var config = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddIniFile("config.ini", optional: true, reloadOnChange: true)
            .Build();

        return WebHost.CreateDefaultBuilder(args)
            .UseConfiguration(config)
            .UseStartup<Startup>();
    }
}

```

O caminho base é definido com [SetBasePath](#). `SetBasePath` está no pacote [Microsoft.Extensions.Configuration.FileExtensions](#), que está no metapacote [Microsoft.AspNetCore.App](#).

Ao criar um [WebHostBuilder](#) diretamente, chame [UseConfiguration](#) com a configuração:

Aplique a configuração ao [WebHostBuilder](#) com o método [UseConfiguration](#):

```

var config = new ConfigurationBuilder()
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddIniFile("config.ini", optional: true, reloadOnChange: true)
    .Build();

var host = new WebHostBuilder()
    .UseConfiguration(config)
    .UseKestrel()
    .UseStartup<Startup>();

```

O caminho base é definido com [SetBasePath](#). `SetBasePath` está no pacote [Microsoft.Extensions.Configuration.FileExtensions](#), que está no metapacote [Microsoft.AspNetCore.App](#).

Um exemplo genérico de um arquivo de configuração INI:

```

[section0]
key0=value
key1=value

[section1]
subsection:key=value

[section2:subsection0]
key=value

[section2:subsection1]
key=value

```

O arquivo de configuração anterior carrega as seguintes chaves com `value`:

- section0:key0
- section0:key1

- section1:subsection:key
- section2:subsection0:key
- section2:subsection1:key

## Provedor de Configuração JSON

O [JsonConfigurationProvider](#) carrega a configuração de pares chave-valor do arquivo JSON em tempo de execução.

Para ativar a configuração de arquivo JSON, chame o método de extensão [AddJsonFile](#) em uma instância do [ConfigurationBuilder](#).

As sobrecargas permitem especificar:

- Se o arquivo é opcional.
- Se a configuração será recarregada caso o arquivo seja alterado.
- O [IFileProvider](#) usado para acessar o arquivo.

`AddJsonFile` é chamado automaticamente duas vezes quando você inicializa um novo [WebHostBuilder](#) com [CreateDefaultBuilder](#). O método é chamado para carregar a configuração de:

- *appsettings.json* – Esse arquivo é lido primeiro. A versão do ambiente do arquivo pode substituir os valores fornecidos pelo arquivo *appsettings.json*.
- *appsettings.{Environment}.json* – A versão de ambiente do arquivo é carregada com base em [IHostingEnvironment.EnvironmentName](#).

Para obter mais informações, confira [Host da Web: Configurar um host](#).

`CreateDefaultBuilder` também carrega:

- Variáveis de ambiente.
- [Segredos do usuário \(Gerenciador de Segredo\)](#) (no ambiente de desenvolvimento).
- Argumentos de linha de comando.

O Provedor de Configuração JSON é estabelecido primeiro. Portanto, os segredos do usuário, as variáveis de ambiente e os argumentos de linha de comando substituem a configuração definida pelos arquivos *appsettings*.

Chame [ConfigureAppConfiguration](#) ao criar o host para especificar a configuração do aplicativo para arquivos que não sejam *appsettings.json* e *appsettings.{Environment}.json*:

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .ConfigureAppConfiguration((hostingContext, config) =>
            {
                config.SetBasePath(Directory.GetCurrentDirectory());
                config.AddJsonFile("config.json", optional: true, reloadOnChange: true);
            })
            .UseStartup<Startup>();
    }
}
```

O caminho base é definido com [SetBasePath](#). `SetBasePath` está no pacote [Microsoft.Extensions.Configuration.FileExtensions](#), que está no metapacote

## Microsoft.AspNetCore.App.

Ao criar um `WebHostBuilder` diretamente, chame `UseConfiguration` com a configuração:

Você pode chamar diretamente o método de extensão `AddJsonFile` em uma instância do `ConfigurationBuilder`.

Ao chamar `CreateDefaultBuilder`, chame `UseConfiguration` com a configuração:

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args)
    {
        var config = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("config.json", optional: true, reloadOnChange: true)
            .Build();

        return WebHost.CreateDefaultBuilder(args)
            .UseConfiguration(config)
            .UseStartup<Startup>();
    }
}
```

O caminho base é definido com `SetBasePath`. `SetBasePath` está no pacote `Microsoft.Extensions.Configuration.FileExtensions`, que está no metapacote `Microsoft.AspNetCore.App`.

Ao criar um `WebHostBuilder` diretamente, chame `UseConfiguration` com a configuração:

Aplice a configuração ao `WebHostBuilder` com o método `UseConfiguration`:

```
var config = new ConfigurationBuilder()
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddJsonFile("config.json", optional: true, reloadOnChange: true)
    .Build();

var host = new WebHostBuilder()
    .UseConfiguration(config)
    .UseKestrel()
    .UseStartup<Startup>();
```

O caminho base é definido com `SetBasePath`. `SetBasePath` está no pacote `Microsoft.Extensions.Configuration.FileExtensions`, que está no metapacote `Microsoft.AspNetCore.App`.

## Exemplo

O aplicativo de exemplo 2.x aproveita a vantagem do método de conveniência estático `CreateDefaultBuilder` para criar o host, que inclui duas chamadas para `AddJsonFile`. A configuração é carregada de `appsettings.json` e de `appsettings.{Environment}.json`.

O aplicativo de exemplo 1.x chama `AddJsonFile` duas vezes em um `ConfigurationBuilder`. A configuração é carregada de `appsettings.json` e de `appsettings.{Environment}.json`.

1. Execute o aplicativo de exemplo. Abra um navegador para o aplicativo em `http://localhost:5000`.

2. Observe que a saída contém pares chave-valor para a configuração mostrada na tabela de acordo com o ambiente. Chaves de configuração de registro usam os dois-pontos (:) como um separador hierárquico.

CHAVE	VALOR DE DESENVOLVIMENTO	VALOR DE PRODUÇÃO
Logging:LogLevel:System	Informações	Informações
Logging:LogLevel:Microsoft	Informações	Informações
Logging:LogLevel:Default	Depurar	Erro
AllowedHosts	*	*

## provedor de Configuração XML

O [XmlConfigurationProvider](#) carrega a configuração de pares chave-valor do arquivo XML em tempo de execução.

Para ativar a configuração de arquivo XML, chame o método de extensão [AddXmlFile](#) em uma instância do [ConfigurationBuilder](#).

As sobrecargas permitem especificar:

- Se o arquivo é opcional.
- Se a configuração será recarregada caso o arquivo seja alterado.
- O [IFileProvider](#) usado para acessar o arquivo.

O nó raiz do arquivo de configuração é ignorado quando os pares chave-valor da configuração são criados. Não especifique um DTD (definição de tipo de documento) ou um namespace no arquivo.

Chame [ConfigureAppConfiguration](#) ao criar o host para especificar a configuração do aplicativo:

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .ConfigureAppConfiguration((hostingContext, config) =>
            {
                config.SetBasePath(Directory.GetCurrentDirectory());
                config.AddXmlFile("config.xml", optional: true, reloadOnChange: true);
            })
            .UseStartup<Startup>();
}
```

O caminho base é definido com [SetBasePath](#). [SetBasePath](#) está no pacote [Microsoft.Extensions.Configuration.FileExtensions](#), que está no metapacote [Microsoft.AspNetCore.App](#).

Ao criar um [WebHostBuilder](#) diretamente, chame [UseConfiguration](#) com a configuração:

Ao chamar [CreateDefaultBuilder](#), chame [UseConfiguration](#) com a configuração:

```

public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args)
    {
        var config = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddXmlFile("config.xml", optional: true, reloadOnChange: true)
            .Build();

        return WebHost.CreateDefaultBuilder(args)
            .UseConfiguration(config)
            .UseStartup<Startup>();
    }
}

```

O caminho base é definido com [SetBasePath](#). `SetBasePath` está no pacote [Microsoft.Extensions.Configuration.FileExtensions](#), que está no [metapacote Microsoft.AspNetCore.App](#).

Ao criar um [WebHostBuilder](#) diretamente, chame [UseConfiguration](#) com a configuração:

Aplique a configuração ao [WebHostBuilder](#) com o método [UseConfiguration](#):

```

var config = new ConfigurationBuilder()
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddXmlFile("config.xml", optional: true, reloadOnChange: true)
    .Build();

var host = new WebHostBuilder()
    .UseConfiguration(config)
    .UseKestrel()
    .UseStartup<Startup>();

```

O caminho base é definido com [SetBasePath](#). `SetBasePath` está no pacote [Microsoft.Extensions.Configuration.FileExtensions](#), que está no [metapacote Microsoft.AspNetCore.App](#).

Os arquivos de configuração XML podem usar nomes de elemento distintos para seções repetidas:

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <section0>
        <key0>value</key0>
        <key1>value</key1>
    </section0>
    <section1>
        <key0>value</key0>
        <key1>value</key1>
    </section1>
</configuration>

```

O arquivo de configuração anterior carrega as seguintes chaves com `value`:

- section0:key0
- section0:key1

- section1:key0
- section1:key1

Elementos repetidos que usam o mesmo nome de elemento funcionarão se o atributo `name` for usado para diferenciar os elementos:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <section name="section0">
    <key name="key0">value</key>
    <key name="key1">value</key>
  </section>
  <section name="section1">
    <key name="key0">value</key>
    <key name="key1">value</key>
  </section>
</configuration>
```

O arquivo de configuração anterior carrega as seguintes chaves com `value`:

- section:section0:key:key0
- section:section0:key:key1
- section:section1:key:key0
- section:section1:key:key1

É possível usar atributos para fornecer valores:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <key attribute="value" />
  <section>
    <key attribute="value" />
  </section>
</configuration>
```

O arquivo de configuração anterior carrega as seguintes chaves com `value`:

- key:attribute
- section:key:attribute

## Provedor de Configuração de Chave por Arquivo

O [KeyPerFileConfigurationProvider](#) usa arquivos do diretório como pares chave-valor de configuração. A chave é o nome do arquivo. O valor contém o conteúdo do arquivo. O Provedor de Configuração de Chave por arquivo é usado em cenários de hospedagem do Docker.

Para ativar a configuração de chave por arquivo, chame o método de extensão [AddKeyPerFile](#) em uma instância do [ConfigurationBuilder](#). O `directoryPath` para os arquivos deve ser um caminho absoluto.

As sobrecargas permitem especificar:

- Um delegado `Action<KeyPerFileConfigurationSource>` que configura a origem.
- Se o diretório é opcional, e o caminho para o diretório.

O sublinhado duplo (`__`) é usado como um delimitador de chave de configuração em nomes de arquivo. Por exemplo, o nome do arquivo `Logging__LogLevel__System` produz a chave de configuração `Logging:LogLevel:System`.

Chame [ConfigureAppConfiguration](#) ao criar o host para especificar a configuração do aplicativo:

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .ConfigureAppConfiguration((hostingContext, config) =>
            {
                config.SetBasePath(Directory.GetCurrentDirectory());
                var path = Path.Combine(Directory.GetCurrentDirectory(), "path/to/files");
                config.AddKeyPerFile(directoryPath: path, optional: true);
            })
            .UseStartup<Startup>();
}
```

O caminho base é definido com [SetBasePath](#). `SetBasePath` está no pacote [Microsoft.Extensions.Configuration.FileExtensions](#), que está no metapacote [Microsoft.AspNetCore.App](#).

Ao criar um [WebHostBuilder](#) diretamente, chame [UseConfiguration](#) com a configuração:

```
var path = Path.Combine(Directory.GetCurrentDirectory(), "path/to/files");
var config = new ConfigurationBuilder()
    .AddKeyPerFile(directoryPath: path, optional: true)
    .Build();

var host = new WebHostBuilder()
    .UseConfiguration(config)
    .UseKestrel()
    .UseStartup<Startup>();
```

## Provedor de Configuração de Memória

O [MemoryConfigurationProvider](#) usa uma coleção na memória como pares chave-valor de configuração.

Para ativar a configuração de coleção na memória, chame o método de extensão [AddInMemoryCollection](#) em uma instância do [ConfigurationBuilder](#).

O provedor de configuração pode ser inicializado com um `IEnumerable<KeyValuePair<String, String>>`.

Chame [ConfigureAppConfiguration](#) ao criar o host para especificar a configuração do aplicativo:

```

public class Program
{
    public static readonly Dictionary<string, string> _dict =
        new Dictionary<string, string>
    {
        {"MemoryCollectionKey1", "value1"},
        {"MemoryCollectionKey2", "value2"}
    };

    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .ConfigureAppConfiguration((hostingContext, config) =>
            {
                config.AddInMemoryCollection(_dict);
            })
            .UseStartup<Startup>();
}

```

Ao criar um [WebHostBuilder](#) diretamente, chame [UseConfiguration](#) com a configuração:

Ao chamar [CreateDefaultBuilder](#), chame [UseConfiguration](#) com a configuração:

```

public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args)
    {
        var dict = new Dictionary<string, string>
        {
            {"MemoryCollectionKey1", "value1"},
            {"MemoryCollectionKey2", "value2"}
        };

        var config = new ConfigurationBuilder()
            .AddInMemoryCollection(dict)
            .Build();

        return WebHost.CreateDefaultBuilder(args)
            .UseConfiguration(config)
            .UseStartup<Startup>();
    }
}

```

Ao criar um [WebHostBuilder](#) diretamente, chame [UseConfiguration](#) com a configuração:

Aplique a configuração ao [WebHostBuilder](#) com o método [UseConfiguration](#):

```

var dict = new Dictionary<string, string>
{
    {"MemoryCollectionKey1", "value1"},
    {"MemoryCollectionKey2", "value2"}
};

var config = new ConfigurationBuilder()
    .AddInMemoryCollection(dict)
    .Build();

var host = new WebHostBuilder()
    .UseConfiguration(config)
    .UseKestrel()
    .UseStartup<Startup>();

```

## GetValue

`ConfigurationBinder.GetValue<T>` extrai um valor de configuração com uma chave especificada e o converte para o tipo especificado. Uma sobrecarga permite que você forneça um valor padrão se a chave não for encontrada.

O exemplo a seguir extrai o valor de cadeia de caracteres da configuração com a chave `NumberKey`, digita o valor como um `int` e armazena o valor na variável `intValue`. Se `NumberKey` não for encontrado em chaves de configuração, `intValue` recebe o valor padrão de `99`:

```
var intValue = config.GetValue<int>("NumberKey", 99);
```

## GetSection, GetChildren e Exists

Para os exemplos a seguir, considere o seguinte arquivo JSON. Há quatro chaves em duas seções, cada uma delas inclui um par de subseções:

```
{
    "section0": {
        "key0": "value",
        "key1": "value"
    },
    "section1": {
        "key0": "value",
        "key1": "value"
    },
    "section2": {
        "subsection0" : {
            "key0": "value",
            "key1": "value"
        },
        "subsection1" : {
            "key0": "value",
            "key1": "value"
        }
    }
}
```

Quando o arquivo é lido na configuração, as seguintes chaves hierárquicas exclusivas são criadas para conter os valores de configuração:

- section0:key0
- section0:key1

- section1:key0
- section1:key1
- section2:subsection0:key0
- section2:subsection0:key1
- section2:subsection1:key0
- section2:subsection1:key1

## GetSection

`IConfiguration.GetSection` extrai uma subseção de configuração com a chave de subseção especificada. `GetSection` está no pacote [Microsoft.Extensions.Configuration](#), que está no [metapacote Microsoft.AspNetCore.App](#).

Para retornar um `IConfigurationSection` que contenha apenas os pares chave-valor em `section1`, chame `GetSection` e forneça o nome da seção:

```
var configSection = _config.GetSection("section1");
```

`configSection` não tem um valor, apenas uma chave e um caminho.

Da mesma forma, para obter os valores de chaves em `section2:subsection0`, chame `GetSection` e forneça o caminho de seção:

```
var configSection = _config.GetSection("section2:subsection0");
```

`GetSection` nunca retorna `null`. Se uma seção correspondente não for encontrada, um `IConfigurationSection` vazio será retornado.

Quando `GetSection` retorna uma seção correspondente, `Value` não é preenchido. `Key` e `Path` são retornados quando a seção existe.

## GetChildren

Uma chamada para `IConfiguration.GetChildren` em `section2` obtém um `IEnumerable< IConfigurationSection >` que inclui:

- `subsection0`
- `subsection1`

```
var configSection = _config.GetSection("section2");

var children = configSection.GetChildren();
```

## Exists

Use `ConfigurationExtensions.Exists` para determinar se há uma seção de configuração:

```
var sectionExists = _config.GetSection("section2:subsection2").Exists();
```

Considerando os dados de exemplo, `sectionExists` é `false`, pois não existe uma seção `section2:subsection2` nos dados de configuração.

## Associar a uma classe

A configuração pode ser associada a classes que representam grupos de configurações relacionadas

usando o *padrão de opções*. Para obter mais informações, consulte [Padrão de opções no ASP.NET Core](#).

Os valores de configuração retornam como cadeias de caracteres, mas chamar `Bind` permite a construção de objetos POCO. `Bind` está no pacote `Microsoft.Extensions.Configuration.Binder`, que está no metapacote `Microsoft.AspNetCore.App`.

O aplicativo de exemplo contém um modelo `Starship` (*Models/Starship.cs*):

```
public class Starship
{
    public string Name { get; set; }
    public string Registry { get; set; }
    public string Class { get; set; }
    public decimal Length { get; set; }
    public bool Commissioned { get; set; }
}
```

```
public class Starship
{
    public string Name { get; set; }
    public string Registry { get; set; }
    public string Class { get; set; }
    public decimal Length { get; set; }
    public bool Commissioned { get; set; }
}
```

A seção `starship` do arquivo `starship.json` cria a configuração quando o aplicativo de exemplo usa o Provedor de Configuração JSON para carregar a configuração:

```
{
  "starship": {
    "name": "USS Enterprise",
    "registry": "NCC-1701",
    "class": "Constitution",
    "length": 304.8,
    "commissioned": false
  },
  "trademark": "Paramount Pictures Corp. http://www.paramount.com"
}
```

```
{
  "starship": {
    "name": "USS Enterprise",
    "registry": "NCC-1701",
    "class": "Constitution",
    "length": 304.8,
    "commissioned": false
  },
  "trademark": "Paramount Pictures Corp. http://www.paramount.com"
}
```

Os seguintes pares chave-valor de configuração são criados:

CHAVE	VALOR
starship:name	USS Enterprise

CHAVE	VALOR
starship:registry	NCC-1701
starship:class	Constituição
starship:length	304,8
starship:commissioned	False
marca	Paramount Pictures Corp. <a href="http://www.paramount.com">http://www.paramount.com</a>

O aplicativo de exemplo chama `GetSection` com a chave `starship`. Os pares chave-valor `starship` são isolados. O método `Bind` é chamado na subseção passando uma instância da classe `Starship`. Depois de associar os valores de instância, a instância é atribuída a uma propriedade para renderização:

```
var starship = new Starship();
_config.GetSection("starship").Bind(starship);
Starship = starship;
```

```
var starship = new Starship();
_config.GetSection("starship").Bind(starship);
viewModel.Starship = starship;
```

`GetSection` está no pacote [Microsoft.Extensions.Configuration](#), que está no metapacote [Microsoft.AspNetCore.App](#).

## Associar a um gráfico de objeto

`Bind` é capaz de associar um grafo de objeto POCO inteiro. `Bind` está no pacote [Microsoft.Extensions.Configuration.Binder](#), que está no metapacote [Microsoft.AspNetCore.App](#).

O exemplo contém um modelo `TvShow` cujo grafo do objeto inclui as classes `Metadata` e `Actors` (`Models/TvShow.cs`):

```
public class TvShow
{
    public Metadata Metadata { get; set; }
    public Actors Actors { get; set; }
    public string Legal { get; set; }
}

public class Metadata
{
    public string Series { get; set; }
    public string Title { get; set; }
    public DateTime AirDate { get; set; }
    public int Episodes { get; set; }
}

public class Actors
{
    public string Names { get; set; }
}
```

```

public class TvShow
{
    public Metadata Metadata { get; set; }
    public Actors Actors { get; set; }
    public string Legal { get; set; }
}

public class Metadata
{
    public string Series { get; set; }
    public string Title { get; set; }
    public DateTime AirDate { get; set; }
    public int Episodes { get; set; }
}

public class Actors
{
    public string Names { get; set; }
}

```

O aplicativo de exemplo tem um arquivo *tvshow.xml* que contém os dados de configuração:

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <tvshow>
        <metadata>
            <series>Dr. Who</series>
            <title>The Sun Makers</title>
            <airdate>11/26/1977</airdate>
            <episodes>4</episodes>
        </metadata>
        <actors>
            <names>Tom Baker, Louise Jameson, John Leeson</names>
        </actors>
        <legal>(c)1977 BBC https://www.bbc.co.uk/programmes/b006q2x0</legal>
    </tvshow>
</configuration>

```

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <tvshow>
        <metadata>
            <series>Dr. Who</series>
            <title>The Sun Makers</title>
            <airdate>11/26/1977</airdate>
            <episodes>4</episodes>
        </metadata>
        <actors>
            <names>Tom Baker, Louise Jameson, John Leeson</names>
        </actors>
        <legal>(c)1977 BBC https://www.bbc.co.uk/programmes/b006q2x0</legal>
    </tvshow>
</configuration>

```

A configuração está associada ao método do grafo de objeto `TvShow` inteiro com o método `Bind`. A instância associada é atribuída a uma propriedade para renderização:

```

var tvShow = new TvShow();
_config.GetSection("tvshow").Bind(tvShow);
TvShow = tvShow;

```

```
var tvShow = new TvShow();
_config.GetSection("tvshow").Bind(tvShow);
viewModel.TvShow = tvShow;
```

`ConfigurationBinder.Get<T>` associa e retorna o tipo especificado. O `Get<T>` é mais conveniente do que usar `Bind`. O código a seguir mostra como usar `Get<T>` com o exemplo anterior, que permite que a instância associada seja diretamente atribuída à propriedade usada para renderização:

```
TvShow = _config.GetSection("tvshow").Get<TvShow>();
```

```
viewModel.TvShow = _config.GetSection("tvshow").Get<TvShow>();
```

`Get` está no pacote [Microsoft.Extensions.Configuration.Binder](#), que está no metapacote [Microsoft.AspNetCore.App](#). `Get<T>` está disponível no ASP.NET Core 1.1 ou posterior. `GetSection` está no pacote [Microsoft.Extensions.Configuration](#), que está no metapacote [Microsoft.AspNetCore.App](#).

## Associar uma matriz a uma classe

O aplicativo de exemplo demonstra os conceitos explicados nesta seção.

O `Bind` dá suporte a matrizes de associação para objetos usando os índices em chaves de configuração. Qualquer formato de matriz que exponha um segmento de chave numérica (`:0:`, `:1:`, ... `:{n}:`) é capaz de associar matrizes a uma matriz de classe POCO. `Bind` está no pacote [Microsoft.Extensions.Configuration.Binder](#), que está no metapacote [Microsoft.AspNetCore.App](#).

### NOTE

A associação é fornecida por convenção. Provedores de configuração personalizados não são necessários para implementar a associação de matriz.

## Processamento de matriz na memória

Considere as chaves de configuração e os valores mostrados na tabela a seguir.

CHAVE	VALOR
array:entries:0	value0
array:entries:1	value1
array:entries:2	value2
array:entries:4	value4
array:entries:5	value5

Essas chaves e valores são carregados no aplicativo de exemplo usando o Provedor de Configuração de Memória:

```

public class Program
{
    public static Dictionary<string, string> arrayDict = new Dictionary<string, string>
    {
        {"array:entries:0", "value0"},
        {"array:entries:1", "value1"},
        {"array:entries:2", "value2"},
        {"array:entries:4", "value4"},
        {"array:entries:5", "value5"}
    };

    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .ConfigureAppConfiguration((hostingContext, config) =>
            {
                config.SetBasePath(Directory.GetCurrentDirectory());
                config.AddInMemoryCollection(arrayDict);
                config.AddJsonFile("json_array.json", optional: false, reloadOnChange: false);
                config.AddJsonFile("starship.json", optional: false, reloadOnChange: false);
                config.AddXmlFile("tvshow.xml", optional: false, reloadOnChange: false);
                config.AddEFConfiguration(options => options.UseInMemoryDatabase("InMemoryDb"));
                config.AddCommandLine(args);
            })
            .UseStartup<Startup>();
    }
}

```

```

public class Startup
{
    public Startup(IHostingEnvironment env)
    {
        var arrayDict = new Dictionary<string, string>
        {
            {"array:entries:0", "value0"},
            {"array:entries:1", "value1"},
            {"array:entries:2", "value2"},
            {"array:entries:4", "value4"},
            {"array:entries:5", "value5"}
        };

        var builder = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddInMemoryCollection(arrayDict)
            .AddJsonFile("json_array.json", optional: false, reloadOnChange: false)
            .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
            .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true,
                reloadOnChange: true)
            .AddEnvironmentVariables()
            .AddJsonFile("starship.json", optional: false, reloadOnChange: false)
            .AddXmlFile("tvshow.xml", optional: false, reloadOnChange: false)
            .AddEFConfiguration(options => options.UseInMemoryDatabase("InMemoryDb"))
            .AddCommandLine(Program.Args);

        Configuration = builder.Build();
    }
}

```

A matriz ignora um valor para o índice #3. O associador de configuração não é capaz de associar valores nulos ou criar entradas nulas em objetos associados, o que fica claro em um momento quando o resultado da associação dessa matriz a um objeto é demonstrado.

No aplicativo de exemplo, uma classe POCO está disponível para armazenar os dados de configuração associados:

```
public class ArrayExample
{
    public string[] Entries { get; set; }
}
```

```
public class ArrayExample
{
    public string[] Entries { get; set; }
}
```

Os dados de configuração estão associados ao objeto:

```
var arrayExample = new ArrayExample();
_config.GetSection("array").Bind(arrayExample);
```

`GetSection` está no pacote [Microsoft.Extensions.Configuration](#), que está no [metapacote Microsoft.AspNetCore.App](#).

A sintaxe [ConfigurationBinder.Get<T>](#) também pode ser usada, o que resulta em um código mais compacto:

```
ArrayExample = _config.GetSection("array").Get<ArrayExample>();
```

```
viewModel.ArrayExample = _config.GetSection("array").Get<ArrayExample>();
```

O objeto associado, uma instância de `ArrayExample`, recebe os dados da matriz de configuração.

ÍNDICE	VALOR
0	value0
1	value1
2	value2
3	value4
4	value5

O índice #3 no objeto associado contém os dados de configuração para a chave de configuração `array:4` e seu valor de `value4`. Quando os dados de configuração que contêm uma matriz são associados, os índices da matriz nas chaves de configuração são simplesmente usados para iterar os dados de configuração ao criar o objeto. Um valor nulo não pode ser mantido nos dados de configuração, e uma entrada de valor nulo não será criada em um objeto de associação quando uma matriz nas chaves de configuração ignorar um ou mais índices.

O item de configuração ausente para o índice #3 pode ser fornecido antes da associação à instância `ArrayExample` por qualquer provedor de configuração que produza o par chave-valor correto na configuração. Se o exemplo incluir um Provedor de Configuração JSON adicional com o par chave-

valor ausente, o `ArrayExample.Entries` coincidirá com a matriz de configuração completa:

*missing\_value.json*:

```
{  
    "array:entries:3": "value3"  
}
```

No `ConfigureAppConfiguration`:

```
config.AddJsonFile("missing_value.json", optional: false, reloadOnChange: false);
```

No construtor `Startup`:

```
.AddJsonFile("missing_value.json", optional: false, reloadOnChange: false);
```

O par chave-valor mostrado na tabela é carregado na configuração.

CHAVE	VALOR
array:entries:3	value3

Se a instância da classe `ArrayExample` for associada após o Provedor de Configuração JSON incluir a entrada para o índice #3, a matriz `ArrayExample.Entries` incluirá o valor.

ÍNDICE <code>ARRAYEXAMPLE.ENTRIES</code>	VALOR <code>ARRAYEXAMPLE.ENTRIES</code>
0	value0
1	value1
2	value2
3	value3
4	value4
5	value5

## Processamento de matriz JSON

Se um arquivo JSON contiver uma matriz, as chaves de configuração serão criadas para os elementos da matriz com um índice de seção de base zero. No arquivo de configuração a seguir, `subsection` é uma matriz:

```
{
  "json_array": {
    "key": "valueA",
    "subsection": [
      "valueB",
      "valueC",
      "valueD"
    ]
  }
}
```

```
{
  "json_array": {
    "key": "valueA",
    "subsection": [
      "valueB",
      "valueC",
      "valueD"
    ]
  }
}
```

O Provedor de Configuração JSON lê os dados de configuração para os seguintes pares chave-valor:

CHAVE	VALOR
json_array:key	valueA
json_array:subsection:0	valueB
json_array:subsection:1	valueC
json_array:subsection:2	valueD

No aplicativo de exemplo, a seguinte classe POCO está disponível para associar os pares chave-valor de configuração:

```
public class JsonArrayExample
{
    public string Key { get; set; }
    public string[] Subsection { get; set; }
}
```

```
public class JsonArrayExample
{
    public string Key { get; set; }
    public string[] Subsection { get; set; }
}
```

Após a associação, `JsonArrayExample.Key` contém o valor `valueA`. Os valores de subseção são armazenados na propriedade matriz POCO, `Subsection`.

ÍNDICE	VALOR
0	valueB
1	valueC
2	valueD

## Provedor de Configuração personalizado

O aplicativo de exemplo demonstra como criar um provedor de configuração básico que lê os pares chave-valor da configuração de um banco de dados usando [Entity Framework \(EF\)](#).

O provedor tem as seguintes características:

- O banco de dados EF na memória é usado para fins de demonstração. Para usar um banco de dados que exija uma cadeia de conexão, implemente um `ConfigurationBuilder` secundário para fornecer a cadeia de conexão de outro provedor de configuração.
- O provedor lê uma tabela de banco de dados na configuração na inicialização. O provedor não consulta o banco de dados em uma base por chave.
- O recarregamento na alteração não está implementado, portanto, a atualização do banco de dados após a inicialização do aplicativo não tem efeito sobre a configuração do aplicativo.

Defina uma entidade `EFConfigurationValue` para armazenar valores de configuração no banco de dados.

*Models/EFConfigurationValue.cs:*

```
public class EFConfigurationValue
{
    public string Id { get; set; }
    public string Value { get; set; }
}
```

```
public class EFConfigurationValue
{
    public string Id { get; set; }
    public string Value { get; set; }
}
```

Adicione um `EFConfigurationContext` para armazenar e acessar os valores configurados.

*EFConfigurationProvider/EFConfigurationContext.cs:*

```
public class EFConfigurationContext : DbContext
{
    public EFConfigurationContext(DbContextOptions options) : base(options)
    {
    }

    public DbSet<EFConfigurationValue> Values { get; set; }
}
```

```
public class EFConfigurationContext : DbContext
{
    public EFConfigurationContext(DbContextOptions options) : base(options)
    {
    }

    public DbSet<EFConfigurationValue> Values { get; set; }
}
```

Crie uma classe que implementa [IConfigurationSource](#).

*EFConfigurationProvider/EFConfigurationSource.cs:*

```
public class EFConfigurationSource : IConfigurationSource
{
    private readonly Action<DbContextOptionsBuilder> _optionsAction;

    public EFConfigurationSource(Action<DbContextOptionsBuilder> optionsAction)
    {
        _optionsAction = optionsAction;
    }

    public IConfigurationProvider Build(IConfigurationBuilder builder)
    {
        return new EFConfigurationProvider(_optionsAction);
    }
}
```

```
public class EFConfigurationSource : IConfigurationSource
{
    private readonly Action<DbContextOptionsBuilder> _optionsAction;

    public EFConfigurationSource(Action<DbContextOptionsBuilder> optionsAction)
    {
        _optionsAction = optionsAction;
    }

    public IConfigurationProvider Build(IConfigurationBuilder builder)
    {
        return new EFConfigurationProvider(_optionsAction);
    }
}
```

Crie o provedor de configuração personalizado através da herança de [ConfigurationProvider](#). O provedor de configuração inicializa o banco de dados quando ele está vazio.

*EFConfigurationProvider/EFConfigurationProvider.cs:*

```

public class EFConfigurationProvider : ConfigurationProvider
{
    public EFConfigurationProvider(Action<DbContextOptionsBuilder> optionsAction)
    {
        OptionsAction = optionsAction;
    }

    Action<DbContextOptionsBuilder> OptionsAction { get; }

    // Load config data from EF DB.
    public override void Load()
    {
        var builder = new DbContextOptionsBuilder<EFConfigurationContext>();

        OptionsAction(builder);

        using (var dbContext = new EFConfigurationContext(builder.Options))
        {
            dbContext.Database.EnsureCreated();

            Data = !dbContext.Values.Any()
                ? CreateAndSaveDefaultValues(dbContext)
                : dbContext.Values.ToDictionary(c => c.Id, c => c.Value);
        }
    }

    private static IDictionary<string, string> CreateAndSaveDefaultValues(
        EFConfigurationContext dbContext)
    {
        // Quotes (c)2005 Universal Pictures: Serenity
        // https://www.uphe.com/movies/serenity
        var configValues = new Dictionary<string, string>
        {
            { "quote1", "I aim to misbehave." },
            { "quote2", "I swallowed a bug." },
            { "quote3", "You can't stop the signal, Mal." }
        };

        dbContext.Values.AddRange(configValues
            .Select(kvp => new EFConfigurationValue
            {
                Id = kvp.Key,
                Value = kvp.Value
            })
            .ToArray());

        dbContext.SaveChanges();

        return configValues;
    }
}

```

```

public class EFConfigurationProvider : ConfigurationProvider
{
    public EFConfigurationProvider(Action<DbContextOptionsBuilder> optionsAction)
    {
        OptionsAction = optionsAction;
    }

    Action<DbContextOptionsBuilder> OptionsAction { get; }

    // Load config data from EF DB.
    public override void Load()
    {
        var builder = new DbContextOptionsBuilder<EFConfigurationContext>();

        OptionsAction(builder);

        using (var dbContext = new EFConfigurationContext(builder.Options))
        {
            dbContext.Database.EnsureCreated();

            Data = !dbContext.Values.Any()
                ? CreateAndSaveDefaultValues(dbContext)
                : dbContext.Values.ToDictionary(c => c.Id, c => c.Value);
        }
    }

    private static IDictionary<string, string> CreateAndSaveDefaultValues(
        EFConfigurationContext dbContext)
    {
        // Quotes (c)2005 Universal Pictures: Serenity
        // https://www.uphe.com/movies/serenity
        var configValues = new Dictionary<string, string>
        {
            { "quote1", "I aim to misbehave." },
            { "quote2", "I swallowed a bug." },
            { "quote3", "You can't stop the signal, Mal." }
        };

        dbContext.Values.AddRange(configValues
            .Select(kvp => new EFConfigurationValue
            {
                Id = kvp.Key,
                Value = kvp.Value
            })
            .ToArray());
    }

    dbContext.SaveChanges();

    return configValues;
}
}

```

Um método de extensão `AddEFConfiguration` permite adicionar a fonte de configuração a um `ConfigurationBuilder`.

*Extensions/EntityFrameworkExtensions.cs:*

```

public static class EntityFrameworkExtensions
{
    public static IConfigurationBuilder AddEFConfiguration(
        this IConfigurationBuilder builder,
        Action<DbContextOptionsBuilder> optionsAction)
    {
        return builder.Add(new EFConfigurationSource(optionsAction));
    }
}

```

```

public static class EntityFrameworkExtensions
{
    public static IConfigurationBuilder AddEFConfiguration(
        this IConfigurationBuilder builder,
        Action<DbContextOptionsBuilder> optionsAction)
    {
        return builder.Add(new EFConfigurationSource(optionsAction));
    }
}

```

O código a seguir mostra como usar o `EFConfigurationProvider` personalizado em `Program.cs`:

```

public class Program
{
    public static Dictionary<string, string> arrayDict = new Dictionary<string, string>
    {
        {"array:entries:0", "value0"},
        {"array:entries:1", "value1"},
        {"array:entries:2", "value2"},
        {"array:entries:4", "value4"},
        {"array:entries:5", "value5"}
    };

    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .ConfigureAppConfiguration((hostingContext, config) =>
            {
                config.SetBasePath(Directory.GetCurrentDirectory());
                config.AddInMemoryCollection(arrayDict);
                config.AddJsonFile("json_array.json", optional: false, reloadOnChange: false);
                config.AddJsonFile("starship.json", optional: false, reloadOnChange: false);
                config.AddXmlFile("tvshow.xml", optional: false, reloadOnChange: false);
                config.AddEFConfiguration(options => options.UseInMemoryDatabase("InMemoryDb"));
                config.AddCommandLine(args);
            })
            .UseStartup<Startup>();
    }
}

```

```

public class Startup
{
    public Startup(IHostingEnvironment env)
    {
        var arrayDict = new Dictionary<string, string>
        {
            {"array:entries:0", "value0"},
            {"array:entries:1", "value1"},
            {"array:entries:2", "value2"},
            {"array:entries:4", "value4"},
            {"array:entries:5", "value5"}
        };

        var builder = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddInMemoryCollection(arrayDict)
            .AddJsonFile("json_array.json", optional: false, reloadOnChange: false)
            .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
            .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true,
                reloadOnChange: true)
            .AddEnvironmentVariables()
            .AddJsonFile("starship.json", optional: false, reloadOnChange: false)
            .AddXmlFile("tvshow.xml", optional: false, reloadOnChange: false)
            .AddEfConfiguration(options => options.UseInMemoryDatabase("InMemoryDb"))
            .AddCommandLine(Program.Args);

        Configuration = builder.Build();
    }
}

```

## Acessar a configuração durante a inicialização

Injecte `IConfiguration` no construtor `Startup` para acessar os valores de configuração em `Startup.ConfigureServices`. Para acessar a configuração em `Startup.Configure`, injete `IConfiguration` diretamente no método ou use a instância do construtor:

```

public class Startup
{
    private readonly IConfiguration _config;

    public Startup(IConfiguration config)
    {
        _config = config;
    }

    public void ConfigureServices(IServiceCollection services)
    {
        var value = _config["key"];
    }

    public void Configure(IApplicationBuilder app, IConfiguration config)
    {
        var value = config["key"];
    }
}

```

Para obter um exemplo de como acessar a configuração usando os métodos de conveniência de inicialização, confira [Inicialização do aplicativo: Métodos de conveniência](#).

## Acessar a configuração em uma página do Razor Pages ou exibição do MVC

Para acessar definições de configuração em uma página do Razor Pages ou uma exibição do MVC, adicione [usando diretiva \(referência de C#: usando diretiva\)](#) para o [namespace Microsoft.Extensions.Configuration](#) e injete [IConfiguration](#) na página ou na exibição.

Em uma página do Razor:

```
@page
@model IndexModel
@using Microsoft.Extensions.Configuration
@inject IConfiguration Configuration

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Index Page</title>
</head>
<body>
    <h1>Access configuration in a Razor Pages page</h1>
    <p>Configuration value for 'key': @Configuration["key"]</p>
</body>
</html>
```

Em uma exibição do MVC:

```
@using Microsoft.Extensions.Configuration
@inject IConfiguration Configuration

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Index View</title>
</head>
<body>
    <h1>Access configuration in an MVC view</h1>
    <p>Configuration value for 'key': @Configuration["key"]</p>
</body>
</html>
```

## Adicionar configuração de um assembly externo

Uma implementação [IHostingStartup](#) permite adicionar melhorias a um aplicativo durante a inicialização de um assembly externo fora da classe `Startup` do aplicativo. Para obter mais informações, consulte [Usar assemblies de inicialização de hospedagem no ASP.NET Core](#).

## Recursos adicionais

- [Padrão de opções no ASP.NET Core](#)
- [Detalhes da configuração da Microsoft](#)

# Padrão de opções no ASP.NET Core

16/01/2019 • 21 minutes to read • [Edit Online](#)

Por Luke Latham

Para obter a versão 1.1 deste tópico, baixe o [Padrão de opções no ASP.NET Core \(versão 1.1, PDF\)](#).

O padrão de opções usa classes para representar grupos de configurações relacionadas. Quando as [definições de configuração](#) são isoladas por cenário em classes separadas, o aplicativo segue dois princípios importantes de engenharia de software:

- O [ISP \(Princípio de Segregação da Interface\) ou Encapsulamento](#) – os cenários (classes) que dependem das definições de configuração dependem apenas das definições de configuração usadas por eles.
- [Separação de Interesses](#) – As configurações para diferentes partes do aplicativo não são dependentes nem acopladas entre si.

As opções também fornecem um mecanismo para validar os dados da configuração. Para obter mais configurações, consulte a seção [Validação de opções](#).

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Pré-requisitos

Referencie o [metapacote Microsoft.AspNetCore.App](#) ou adicione uma referência de pacote ao pacote [Microsoft.Extensions.Options.ConfigurationExtensions](#).

## Interfaces de opções

O [IOptionsMonitor<TOptions>](#) é usado para recuperar as opções e gerenciar notificações de opções para instâncias de `TOptions`. O [IOptionsMonitor<TOptions>](#) dá suporte aos seguintes cenários:

- Notificações de alteração
- [Opções nomeadas](#)
- [Configuração recarregável](#)
- Invalidação seletiva de opções ([IOptionsMonitorCache<TOptions>](#))

Os cenários de [Pós-configuração](#) permitem que você defina ou altere as opções depois de todas as configurações do [IConfigureOptions<TOptions>](#) serem feitas.

O [IOptionsFactory<TOptions>](#) é responsável por criar novas instâncias de opções. Ele tem um único método `Create`. A implementação padrão usa todos os [IConfigureOptions<TOptions>](#) e [IPostConfigureOptions<TOptions>](#) registrados e executa todas as configurações primeiro, seguidas da pós-configuração. Ela faz distinção entre [IConfigureNamedOptions<TOptions>](#) e [IConfigureOptions<TOptions>](#) e chama apenas a interface apropriada.

O [IOptionsMonitorCache<TOptions>](#) é usado pelo [IOptionsMonitor<TOptions>](#) para armazenar em cache as instâncias do `TOptions`. O [IOptionsMonitorCache<TOptions>](#) invalida as instâncias de opções no monitor, de modo que o valor seja recalculado ([TryRemove](#)). Os valores podem ser manualmente inseridos com [TryAdd](#). O método `Clear` é usado quando todas as instâncias nomeadas devem ser recriadas sob demanda.

O [IOptionsSnapshot<TOptions>](#) é útil em cenários em que as opções devam ser novamente computadas em cada solicitação. Para saber mais, consulte a seção [Recarregar dados de configuração com IOptionsSnapshot](#).

O `IOptions<TOptions>` pode ser usado para dar suporte a opções. No entanto, o `IOptions<TOptions>` não dá suporte a cenários anteriores do `IOptionsMonitor<TOptions>`. Você pode continuar a usar o `IOptions<TOptions>` em estruturas e bibliotecas existentes que já usam a interface do `IOptions<TOptions>` e não exigem os cenários fornecidos pelo `IOptionsMonitor<TOptions>`.

## Configuração de opções gerais

A configuração de opções gerais é demonstrada no Exemplo #1 no aplicativo de exemplo.

Uma classe de opções deve ser não abstrata e com um construtor público sem parâmetros. A classe a seguir, `MyOptions`, tem duas propriedades, `Option1` e `Option2`. A configuração de valores padrão é opcional, mas o construtor de classe no exemplo a seguir define o valor padrão de `Option1`. `Option2` tem um valor padrão definido com a inicialização da propriedade diretamente (`Models/MyOptions.cs`):

```
public class MyOptions
{
    public MyOptions()
    {
        // Set default value.
        Option1 = "value1_from_ctor";
    }

    public string Option1 { get; set; }
    public int Option2 { get; set; } = 5;
}
```

A classe `MyOptions` é adicionada ao contêiner de serviço com `Configure` e associada à configuração:

```
// Example #1: General configuration
// Register the Configuration instance which MyOptions binds against.
services.Configure<MyOptions>(Configuration);
```

O seguinte modelo de página usa a [injeção de dependência de construtor](#) com `IOptionsMonitor<TOptions>` para acessar as configurações (`Pages/Index.cshtml.cs`):

```
private readonly MyOptions _options;
```

```
public IndexModel(
    IOptionsMonitor<MyOptions> optionsAccessor,
    IOptionsMonitor<MyOptionsWithDelegateConfig> optionsAccessorWithDelegateConfig,
    IOptionsMonitor<MySubOptions> subOptionsAccessor,
    IOptionsSnapshot<MyOptions> snapshotOptionsAccessor,
    IOptionsSnapshot<MyOptions> namedOptionsAccessor)
{
    _options = optionsAccessor.CurrentValue;
    _optionsWithDelegateConfig = optionsAccessorWithDelegateConfig.CurrentValue;
    _subOptions = subOptionsAccessor.CurrentValue;
    _snapshotOptions = snapshotOptionsAccessor.Value;
    _named_options_1 = namedOptionsAccessor.Get("named_options_1");
    _named_options_2 = namedOptionsAccessor.Get("named_options_2");
}
```

```
// Example #1: Simple options
var option1 = _options.Option1;
var option2 = _options.Option2;
SimpleOptions = $"option1 = {option1}, option2 = {option2}";
```

O arquivo `appsettings.json` de exemplo especifica valores para `option1` e `option2`:

```
{  
    "option1": "value1_from_json",  
    "option2": -1,  
    "subsection": {  
        "suboption1": "subvalue1_from_json",  
        "suboption2": 200  
    },  
    "Logging": {  
        "LogLevel": {  
            "Default": "Warning"  
        }  
    },  
    "AllowedHosts": "*"  
}
```

Quando o aplicativo é executado, o método `OnGet` do modelo de página retorna uma cadeia de caracteres que mostra os valores da classe de opção:

```
option1 = value1_from_json, option2 = -1
```

#### NOTE

Ao usar um [ConfigurationBuilder](#) personalizado para carregar opções de configuração de um arquivo de configurações, confirme se o caminho de base está corretamente configurado:

```
var configBuilder = new ConfigurationBuilder()  
    .SetBasePath(Directory.GetCurrentDirectory())  
    .AddJsonFile("appsettings.json", optional: true);  
var config = configBuilder.Build();  
  
services.Configure<MyOptions>(config);
```

Não é necessário configurar o caminho de base explicitamente ao carregar opções de configuração do arquivo de configurações por meio do [CreateDefaultBuilder](#).

## Configurar opções simples com um delegado

A configuração de opções simples com um delegado é demonstrada no Exemplo #2 no aplicativo de exemplo.

Use um delegado para definir valores de opções. O aplicativo de exemplo usa a classe

`MyOptionsWithDelegateConfig` (`Models/MyOptionsWithDelegateConfig.cs`):

```
public class MyOptionsWithDelegateConfig  
{  
    public MyOptionsWithDelegateConfig()  
    {  
        // Set default value.  
        Option1 = "value1_from_ctor";  
    }  
  
    public string Option1 { get; set; }  
    public int Option2 { get; set; } = 5;  
}
```

No código a seguir, um segundo serviço `IConfigureOptions<TOptions>` é adicionado ao contêiner de serviço.

Ele usa um delegado para configurar a associação com `MyOptionsWithDelegateConfig`:

```
// Example #2: Options bound and configured by a delegate
services.Configure<MyOptionsWithDelegateConfig>(myOptions =>
{
    myOptions.Option1 = "value1_configured_by_delegate";
    myOptions.Option2 = 500;
});
```

*Index.cshtml.cs:*

```
private readonly MyOptionsWithDelegateConfig _optionsWithDelegateConfig;
```

```
public IndexModel(
    IOptionsMonitor<MyOptions> optionsAccessor,
    IOptionsMonitor<MyOptionsWithDelegateConfig> optionsAccessorWithDelegateConfig,
    IOptionsMonitor<MySubOptions> subOptionsAccessor,
    IOptionsSnapshot<MyOptions> snapshotOptionsAccessor,
    IOptionsSnapshot<MyOptions> namedOptionsAccessor)
{
    _options = optionsAccessor.CurrentValue;
    _optionsWithDelegateConfig = optionsAccessorWithDelegateConfig.CurrentValue;
    _subOptions = subOptionsAccessor.CurrentValue;
    _snapshotOptions = snapshotOptionsAccessor.Value;
    _named_options_1 = namedOptionsAccessor.Get("named_options_1");
    _named_options_2 = namedOptionsAccessor.Get("named_options_2");
}
```

```
// Example #2: Options configured by delegate
var delegate_config_option1 = _optionsWithDelegateConfig.Option1;
var delegate_config_option2 = _optionsWithDelegateConfig.Option2;
SimpleOptionsWithDelegateConfig =
    $"delegate_option1 = {delegate_config_option1}, " +
    $"delegate_option2 = {delegate_config_option2}";
```

Adicione vários provedores de configuração. Os provedores de configuração estão disponíveis nos pacotes do NuGet e são aplicados na ordem em que são registrados. Para obter mais informações, consulte [Configuração no ASP.NET Core](#).

Cada chamada à `Configure` adiciona um serviço `IConfigureOptions<TOptions>` ao contêiner de serviço. No exemplo anterior, os valores `Option1` e `Option2` são especificados em `appsettings.json`, mas os valores `Option1` e `Option2` são substituídos pelo delegado configurado.

Quando mais de um serviço de configuração é habilitado, a última fonte de configuração especificada *vence* e define o valor de configuração. Quando o aplicativo é executado, o método `OnGet` do modelo de página retorna uma cadeia de caracteres que mostra os valores da classe de opção:

```
delegate_option1 = value1_configured_by_delegate, delegate_option2 = 500
```

## Configuração de subopções

A configuração de subopções é demonstrada no Exemplo #3 no aplicativo de exemplo.

Os aplicativos devem criar classes de opções que pertencem a grupos de cenários específicos (classes) no aplicativo. Partes do aplicativo que exigem valores de configuração devem ter acesso apenas aos valores de configuração usados por elas.

Ao associar opções à configuração, cada propriedade no tipo de opções é associada a uma chave de configuração do formato `property[:sub-property:]`. Por exemplo, a propriedade `MyOptions.Option1` é associada à chave `Option1`, que é lida da propriedade `option1` em `appsettings.json`.

No código a seguir, um terceiro serviço `IConfigureOptions<TOptions>` é adicionado ao contêiner de serviço. Ele associa `MySubOptions` à seção `subsection` do arquivo `appsettings.json`:

```
// Example #3: Suboptions
// Bind options using a sub-section of the appsettings.json file.
services.Configure<MySubOptions>(Configuration.GetSection("subsection"));
```

O método de extensão `GetSection` exige o pacote NuGet [Microsoft.Extensions.Options.ConfigurationExtensions](#). Se o aplicativo usa o [metapacote Microsoft.AspNetCore.App](#) (ASP.NET Core 2.1 ou posterior), o pacote é automaticamente incluído.

O arquivo `appsettings.json` de exemplo define um membro `subsection` com chaves para `suboption1` e `suboption2`:

```
{
  "option1": "value1_from_json",
  "option2": -1,
  "subsection": {
    "suboption1": "subvalue1_from_json",
    "suboption2": 200
  },
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

A classe `MySubOptions` define as propriedades `SubOption1` e `SubOption2`, para armazenar os valores de opções (`Models/MySubOptions.cs`):

```
public class MySubOptions
{
    public MySubOptions()
    {
        // Set default values.
        SubOption1 = "value1_from_ctor";
        SubOption2 = 5;
    }

    public string SubOption1 { get; set; }
    public int SubOption2 { get; set; }
}
```

O método `OnGet` do modelo da página retorna uma cadeia de caracteres com os valores de opções (`Pages/Index.cshtml.cs`):

```
private readonly MySubOptions _subOptions;
```

```

public IndexModel(
    IOptionsMonitor<MyOptions> optionsAccessor,
    IOptionsMonitor<MyOptionsWithDelegateConfig> optionsAccessorWithDelegateConfig,
    IOptionsMonitor<MySubOptions> subOptionsAccessor,
    IOptionsSnapshot<MyOptions> snapshotOptionsAccessor,
    IOptionsSnapshot<MyOptions> namedOptionsAccessor)
{
    _options = optionsAccessor.CurrentValue;
    _optionsWithDelegateConfig = optionsAccessorWithDelegateConfig.CurrentValue;
    _subOptions = subOptionsAccessor.CurrentValue;
    _snapshotOptions = snapshotOptionsAccessor.Value;
    _named_options_1 = namedOptionsAccessor.Get("named_options_1");
    _named_options_2 = namedOptionsAccessor.Get("named_options_2");
}

```

```

// Example #3: Suboptions
var subOption1 = _subOptions.SubOption1;
var subOption2 = _subOptions.SubOption2;
SubOptions = $"subOption1 = {subOption1}, subOption2 = {subOption2}";

```

Quando o aplicativo é executado, o método `OnGet` retorna uma cadeia de caracteres que mostra os valores da classe de subopção:

```
subOption1 = subvalue1_from_json, subOption2 = 200
```

## Opções fornecidas por um modelo de exibição ou com a injeção de exibição direta

As opções fornecidas por um modelo de exibição ou com a injeção de exibição direta são demonstradas no Exemplo #4 no aplicativo de exemplo.

As opções podem ser fornecidas em um modelo de exibição ou pela injeção de `IOptionsMonitor<TOptions>` diretamente em uma exibição (`Pages/Index.cshtml.cs`):

```
private readonly MyOptions _options;
```

```

public IndexModel(
    IOptionsMonitor<MyOptions> optionsAccessor,
    IOptionsMonitor<MyOptionsWithDelegateConfig> optionsAccessorWithDelegateConfig,
    IOptionsMonitor<MySubOptions> subOptionsAccessor,
    IOptionsSnapshot<MyOptions> snapshotOptionsAccessor,
    IOptionsSnapshot<MyOptions> namedOptionsAccessor)
{
    _options = optionsAccessor.CurrentValue;
    _optionsWithDelegateConfig = optionsAccessorWithDelegateConfig.CurrentValue;
    _subOptions = subOptionsAccessor.CurrentValue;
    _snapshotOptions = snapshotOptionsAccessor.Value;
    _named_options_1 = namedOptionsAccessor.Get("named_options_1");
    _named_options_2 = namedOptionsAccessor.Get("named_options_2");
}

```

```

// Example #4: Bind options directly to the page
MyOptions = _options;

```

O aplicativo de exemplo mostra como injetar `IOptionsMonitor<MyOptions>` com uma diretiva `@inject`:

```
@page
@model IndexModel
@using Microsoft.Extensions.Options
@inject IOptionsMonitor<MyOptions> OptionsAccessor
 @{
    ViewData["Title"] = "Options Sample";
}

<h1>@ViewData["Title"]</h1>
```

Quando o aplicativo é executado, os valores de opções são mostrados na página renderizada:

#### Example #4: Model and injected options

##### Options provided by the model

Options provided by the model: `@Model.MyOptions.Option1` and `@Model.MyOptions.Option2`

**Option1:** value1\_from\_json

**Option2:** -1

##### Options injected into the page

Options injected into the page: `@inject IOptions<MyOptions> OptionsAccessor` with `@OptionsAccessor.Value.Option1` and `@OptionsAccessor.Value.Option2`

**Option1:** value1\_from\_json

**Option2:** -1

## Recarregar dados de configuração com `IOptionsSnapshot`

O recarregamento de dados de configuração com `IOptionsSnapshot<TOptions>` é demonstrado no Exemplo #5 no aplicativo de exemplo.

O `IOptionsSnapshot<TOptions>` dá suporte a opções de recarregamento com sobrecarga mínima de processamento.

As opções são calculadas uma vez por solicitação, quando acessadas e armazenadas em cache durante o tempo de vida da solicitação.

O exemplo a seguir demonstra como um novo `IOptionsSnapshot<TOptions>` é criado após a alteração de `appsettings.json` (`Pages/Index.cshtml.cs`). Várias solicitações ao servidor retornam valores de constante fornecidos pelo arquivo `appsettings.json`, até que o arquivo seja alterado e a configuração seja recarregada.

```
private readonly MyOptions _snapshotOptions;
```

```
public IndexModel(
    IOptionsMonitor<MyOptions> optionsAccessor,
    IOptionsMonitor<MyOptionsWithDelegateConfig> optionsAccessorWithDelegateConfig,
    IOptionsMonitor<MySubOptions> subOptionsAccessor,
    IOptionsSnapshot<MyOptions> snapshotOptionsAccessor,
    IOptionsSnapshot<MyOptions> namedOptionsAccessor)
{
    _options = optionsAccessor.CurrentValue;
    _optionsWithDelegateConfig = optionsAccessorWithDelegateConfig.CurrentValue;
    _subOptions = subOptionsAccessor.CurrentValue;
    _snapshotOptions = snapshotOptionsAccessor.Value;
    _named_options_1 = namedOptionsAccessor.Get("named_options_1");
    _named_options_2 = namedOptionsAccessor.Get("named_options_2");
}
```

```
// Example #5: Snapshot options
var snapshotOption1 = _snapshotOptions.Option1;
var snapshotOption2 = _snapshotOptions.Option2;
SnapshotOptions =
    $"snapshot option1 = {snapshotOption1}, " +
    $"snapshot option2 = {snapshotOption2}";
```

A seguinte imagem mostra os valores `option1` e `option2` iniciais carregados do arquivo `appsettings.json`:

```
snapshot option1 = value1_from_json, snapshot option2 = -1
```

Altere os valores no arquivo `appsettings.json` para `value1_from_json UPDATED` e `200`. Salve o arquivo `appsettings.json`. Atualize o navegador para ver se os valores de opções foram atualizados:

```
snapshot option1 = value1_from_json UPDATED, snapshot option2 = 200
```

## Suporte de opções nomeadas com `IConfigureNamedOptions`

O suporte de opções nomeadas com `IConfigureNamedOptions<TOptions>` é demonstrado como no Exemplo #6 no aplicativo de exemplo.

O suporte de *opções nomeadas* permite que o aplicativo faça a distinção entre as configurações de opções nomeadas. No aplicativo de exemplo, as opções nomeadas são declaradas com `OptionsServiceCollectionExtensions.Configure` que chama o método de extensão `ConfigureNamedOptions<TOptions>.Configure`:

```
// Example #6: Named options (named_options_1)
// Register the ConfigurationBuilder instance which MyOptions binds against.
// Specify that the options loaded from configuration are named
// "named_options_1".
services.Configure<MyOptions>("named_options_1", Configuration);

// Example #6: Named options (named_options_2)
// Specify that the options loaded from the MyOptions class are named
// "named_options_2".
// Use a delegate to configure option values.
services.Configure<MyOptions>("named_options_2", myOptions =>
{
    myOptions.Option1 = "named_options_2_value1_from_action";
});
```

O aplicativo de exemplo acessa as opções nomeadas com [Get](#) (`Pages/Index.cshtml.cs`):

```
private readonly MyOptions _named_options_1;
private readonly MyOptions _named_options_2;

public IndexModel(
    IOptionsMonitor<MyOptions> optionsAccessor,
    IOptionsMonitor<MyOptionsWithDelegateConfig> optionsAccessorWithDelegateConfig,
    IOptionsMonitor<MySubOptions> subOptionsAccessor,
    IOptionsSnapshot<MyOptions> snapshotOptionsAccessor,
    IOptionsSnapshot<MyOptions> namedOptionsAccessor)
{
    _options = optionsAccessor.CurrentValue;
    _optionsWithDelegateConfig = optionsAccessorWithDelegateConfig.CurrentValue;
    _subOptions = subOptionsAccessor.CurrentValue;
    _snapshotOptions = snapshotOptionsAccessor.Value;
    _named_options_1 = namedOptionsAccessor.Get("named_options_1");
    _named_options_2 = namedOptionsAccessor.Get("named_options_2");
}
```

```
// Example #6: Named options
var named_options_1 =
    $"named_options_1: option1 = {_named_options_1.Option1}, " +
    $"option2 = {_named_options_1.Option2}";
var named_options_2 =
    $"named_options_2: option1 = {_named_options_2.Option1}, " +
    $"option2 = {_named_options_2.Option2}";
NamedOptions = $"{named_options_1} {named_options_2}";
```

Executando o aplicativo de exemplo, as opções nomeadas são retornadas:

```
named_options_1: option1 = value1_from_json, option2 = -1
named_options_2: option1 = named_options_2_value1_from_action, option2 = 5
```

Os valores `named_options_1` são fornecidos pela configuração, que são carregados do arquivo `appsettings.json`.

Os valores `named_options_2` são fornecidos pelo:

- Delegado `named_options_2` em `ConfigureServices` para `Option1`.
- Valor padrão para `option2` fornecido pela classe `MyOptions`.

## Configurar todas as opções com o método `ConfigureAll`

Configure todas as instâncias de opções com o método [ConfigureAll](#). O código a seguir configura `Option1` para todas as instâncias de configuração com um valor comum. Adicione o seguinte código manualmente ao método `Startup.ConfigureServices`:

```
services.ConfigureAll<MyOptions>(myOptions =>
{
    myOptions.Option1 = "ConfigureAll replacement value";
});
```

A execução do aplicativo de exemplo após a adição do código produz o seguinte resultado:

```
named_options_1: option1 = ConfigureAll replacement value, option2 = -1
named_options_2: option1 = ConfigureAll replacement value, option2 = 5
```

#### NOTE

Todas as opções são instâncias nomeadas. As instâncias `IConfigureOptions<TOptions>` existentes são tratadas como sendo direcionadas à instância `Options.DefaultName`, que é `string.Empty`. `IConfigureNamedOptions<TOptions>` também implementa `IConfigureOptions<TOptions>`. A implementação padrão de `IOptionsFactory<TOptions>` tem lógica para usar cada um de forma adequada. A opção nomeada `null` é usada para direcionar todas as instâncias nomeadas, em vez de uma instância nomeada específica (`ConfigureAll` e `PostConfigureAll` usam essa convenção).

## API OptionsBuilder

`OptionsBuilder<TOptions>` é usada para configurar instâncias `TOptions`. `OptionsBuilder` simplifica a criação de opções nomeadas, pois é apenas um único parâmetro para a chamada `AddOptions<TOptions>(string optionsName)` inicial, em vez de aparecer em todas as chamadas subsequentes. A validação de opções e as sobrecargas `ConfigureOptions` que aceitam dependências de serviço só estão disponíveis por meio de `OptionsBuilder`.

```
// Options.DefaultName = "" is used.  
services.AddOptions<MyOptions>().Configure(o => o.Property = "default");  
  
services.AddOptions<MyOptions>("optionalName")  
.Configure(o => o.Property = "named");
```

## Configure<TOptions, TDep1, ... TDep4> método

O uso de serviços da DI para configurar opções com a implementação de `IConfigure[Named]Options` de forma padrão é detalhado. Sobrecargas de `ConfigureOptions` em `OptionsBuilder<TOptions>` permitem que você use até cinco serviços para configurar opções:

```
services.AddOptions<MyOptions>("optionalName")  
.Configure<Service1, Service2, Service3, Service4, Service5>(  
(o, s, s2, s3, s4, s5) =>  
    o.Property = DoSomethingWith(s, s2, s3, s4, s5));
```

A sobrecarga registra um genérico transitório `IConfigureNamedOptions<TOptions>`, que tem um construtor que aceita os tipos de serviços genéricos especificados.

## Validação de opções

A validação de opções permite validar as opções depois de configurá-las. Chame `Validate` com um método de validação que retorna `true` quando as opções são válidas, e `false` quando não são:

```

// Registration
services.AddOptions<MyOptions>("optionalOptionsName")
    .Configure(o => { }) // Configure the options
    .Validate(o => YourValidationShouldReturnTrueIfValid(o),
        "custom error");

// Consumption
var monitor = services.BuildServiceProvider()
    .GetService<IOptionsMonitor<MyOptions>>();

try
{
    var options = monitor.Get("optionalOptionsName");
}
catch (OptionsValidationException e)
{
    // e.OptionsName returns "optionalOptionsName"
    // e.OptionsType returns typeof(MyOptions)
    // e.Failures returns a list of errors, which would contain
    //     "custom error"
}

```

O exemplo anterior define a instância de opções nomeadas como `optionalOptionsName`. A instância de opções padrão é `Options.DefaultName`.

A validação é executada após a criação da instância de opções. A instância de opções tem garantia de passar pela validação, na primeira vez em que for acessada.

#### IMPORTANT

A validação não protege contra modificações de opções, depois que as opções são inicialmente configuradas e validadas.

O método `Validate` aceita um `Func<TOptions, bool>`. Para personalizar totalmente a validação, implemente `IValidateOptions<TOptions>`, que permite:

- A validação de vários tipos de opções:  
`class ValidateTwo : IValidateOptions<Option1>, IValidateOptions<Option2>`
- A validação que depende de outro tipo de opção:  
`public DependsOnAnotherOptionValidator(IOptionsMonitor<AnotherOption> options)`

`IValidateOptions` valida:

- Uma instância específica de opções nomeadas.
- Todas as opções quando `name` for `null`.

Retornar um `ValidateOptionsResult` da implementação da interface:

```

public interface IValidateOptions<TOptions> where TOptions : class
{
    ValidateOptionsResult Validate(string name, TOptions options);
}

```

A validação de anotação de dados está disponível no pacote [Microsoft.Extensions.Options.DataAnnotations](#) por meio da chamada do método `ValidateDataAnnotations` em `OptionsBuilder<TOptions>`. O `Microsoft.Extensions.Options.DataAnnotations` está incluído no [metapacote Microsoft.AspNetCore.App](#) (ASP.NET Core 2.2 ou posterior).

```

private class AnnotatedOptions
{
    [Required]
    public string Required { get; set; }

    [StringLength(5, ErrorMessage = "Too long.")]
    public string StringLength { get; set; }

    [Range(-5, 5, ErrorMessage = "Out of range.")]
    public int IntRange { get; set; }
}

[Fact]
public void CanValidateDataAnnotations()
{
    var services = new ServiceCollection();
    services.AddOptions<AnnotatedOptions>()
        .Configure(o =>
    {
        o.StringLength = "111111";
        o.IntRange = 10;
        o.Custom = "nowhere";
    })
        .ValidateDataAnnotations();

    var sp = services.BuildServiceProvider();

    var error = Assert.Throws<OptionsValidationException>(() =>
        sp.GetRequiredService<IOptionsMonitor<AnnotatedOptions>>().Value);
    ValidateFailure<AnnotatedOptions>(error, Options.DefaultName, 1,
        "DataAnnotation validation failed for members Required " +
        "with the error 'The Required field is required.'",
        "DataAnnotation validation failed for members StringLength " +
        "with the error 'Too long.'",
        "DataAnnotation validation failed for members IntRange " +
        "with the error 'Out of range.'");
}

```

A validação adiantada (fail fast na inicialização) está sendo considerada para uma versão futura.

## Pós-configuração de opções

Defina a pós-configuração com [IPostConfigureOptions<TOptions>](#). A pós-configuração é executada depois que toda a configuração de [IConfigureOptions<TOptions>](#) é feita:

```

services.PostConfigure<MyOptions>(myOptions =>
{
    myOptions.Option1 = "post_configured_option1_value";
});

```

O [PostConfigure](#) está disponível para pós-configurar opções nomeadas:

```

services.PostConfigure<MyOptions>("named_options_1", myOptions =>
{
    myOptions.Option1 = "post_configured_option1_value";
});

```

Use [PostConfigureAll](#) para pós-configurar todas as instâncias de configuração:

```
services.PostConfigureAll<MyOptions>(myOptions =>
{
    myOptions.Option1 = "post_configured_option1_value";
});
```

## Acessando opções durante a inicialização

`IOptions<TOptions>` e `IOptionsMonitor<TOptions>` podem ser usados em `Startup.Configure`, pois os serviços são criados antes da execução do método `Configure`.

```
public void Configure(IApplicationBuilder app, IOptionsMonitor<MyOptions> optionsAccessor)
{
    var option1 = optionsAccessor.CurrentValue.Option1;
}
```

Não use `IOptions<TOptions>` ou `IOptionsMonitor<TOptions>` em `Startup.ConfigureServices`. Pode haver um estado inconsistente de opções devido à ordenação dos registros de serviço.

## Recursos adicionais

- [Configuração no ASP.NET Core](#)

# Registro em log no ASP.NET Core

04/02/2019 • 47 minutes to read • [Edit Online](#)

Por [Steve Smith](#) e [Tom Dykstra](#)

O ASP.NET Core oferece suporte a uma API de registro em log que funciona com uma variedade de provedores de logs internos e terceirizados. Este artigo mostra como usar a API de registro em log com provedores internos.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Adicionar provedores

Um provedor de log exibe ou armazena logs. Por exemplo, o provedor de Console exibe os logs no console, e o provedor do Azure Application Insights armazena-os no Azure Application Insights. Os logs podem ser enviados para vários destinos por meio da adição de vários provedores.

Para adicionar um provedor, chame o método de extensão `Add{provider name}` do provedor no *Program.cs*:

```
public static void Main(string[] args)
{
    var webHost = new WebHostBuilder()
        .UseKestrel()
        .UseContentRoot(Directory.GetCurrentDirectory())
        .ConfigureAppConfiguration((hostingContext, config) =>
    {
        var env = hostingContext.HostingEnvironment;
        config.AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
            .AddJsonFile($"appsettings.{env.EnvironmentName}.json",
                optional: true, reloadOnChange: true);
        config.AddEnvironmentVariables();
    })
        .ConfigureLogging((hostingContext, logging) =>
    {
        logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));
        logging.AddConsole();
        logging.AddDebug();
        logging.AddEventSourceLogger();
    })
        .UseStartup<Startup>()
        .Build();

    webHost.Run();
}
```

O modelo de projeto padrão chama o método de extensão `CreateDefaultBuilder`, que adiciona os seguintes provedores de log:

- Console
- Depurar
- EventSource (a partir do ASP.NET Core 2.2)

```

public static void Main(string[] args)
{
    BuildWebHost(args).Run();
}

public static IWebHost BuildWebHost(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .Build();

```

Se você usar `CreateDefaultBuilder`, poderá substituir os provedores padrão por aqueles que preferir. Chame `ClearProviders` e adicione os provedores desejados.

```

public static void Main(string[] args)
{
    var host = BuildWebHost(args);

    var todoRepository = host.Services.GetRequiredService<ITodoRepository>();
    todoRepository.Add(new Core.Model.TodoItem() { Name = "Feed the dog" });
    todoRepository.Add(new Core.Model.TodoItem() { Name = "Walk the dog" });

    var logger = host.Services.GetRequiredService<ILogger<Program>>();
    logger.LogInformation("Seeded the database.");

    host.Run();
}

public static IWebHost BuildWebHost(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .ConfigureLogging(logging =>
    {
        logging.ClearProviders();
        logging.AddConsole();
    })
        .Build();

```

Para usar um provedor, instale o pacote NuGet e chame o método de extensão do provedor em uma instância de `ILoggerFactory`:

```

public void Configure(IApplicationBuilder app,
    IHostingEnvironment env,
    ILoggerFactory loggerFactory)
{
    loggerFactory
        .AddConsole()
        .AddDebug();
}

```

A [DI \(injeção de dependência\)](#) do ASP.NET Core fornece a instância de `ILoggerFactory`. Os métodos de extensão `AddConsole` e `AddDebug` são definidos nos pacotes [Microsoft.Extensions.Logging.Console](#) e [Microsoft.Extensions.Logging.Debug](#). Cada método de extensão chama o método `ILoggerFactory.AddProvider`, passando uma instância do provedor.

#### NOTE

O [aplicativo de exemplo](#) adiciona provedores de log no método `Startup.Configure`. Para obter saída de log do código que é executado anteriormente, adicione provedores de log no construtor da classe `Startup`.

Saiba mais sobre [provedores de log internos](#) e [provedores de log de terceiros](#) mais adiante no artigo.

## Criar logs

Obtenha um objeto `ILogger<TCategoryName>` da DI.

O exemplo de controlador a seguir cria os logs `Information` e `Warning`. A *categoria* é `TodoApiSample.Controllers.TodoController` (o nome de classe totalmente qualificado do `TodoController` no aplicativo de exemplo):

```
public class TodoController : Controller
{
    private readonly ITodoRepository _todoRepository;
    private readonly ILogger _logger;

    public TodoController(ITodoRepository todoRepository,
        ILogger<TodoController> logger)
    {
        _todoRepository = todoRepository;
        _logger = logger;
    }
}
```

```
public IActionResult GetById(string id)
{
    _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
    var item = _todoRepository.Find(id);
    if (item == null)
    {
        _logger.LogWarning(LoggingEvents.GetItemNotFound, "GetById({ID}) NOT FOUND", id);
        return NotFound();
    }
    return new ObjectResult(item);
}
```

O exemplo do Razor Pages a seguir cria logs com `Information` como o *nível* e `TodoApiSample.Pages.AboutModel` como a *categoria*:

```
public class AboutModel : PageModel
{
    private readonly ILogger _logger;

    public AboutModel(ILogger<AboutModel> logger)
    {
        _logger = logger;
    }
}
```

```
public void OnGet()
{
    Message = $"About page visited at {DateTime.UtcNow.ToString('yyyy-MM-ddTHH:mm:ss')}";
    _logger.LogInformation("Message displayed: {Message}", Message);
}
```

```
public class TodoController : Controller
{
    private readonly ITodoRepository _todoRepository;
    private readonly ILogger _logger;

    public TodoController(ITodoRepository todoRepository,
        ILogger<TodoController> logger)
    {
        _todoRepository = todoRepository;
        _logger = logger;
    }
}
```

```
public IActionResult GetById(string id)
{
    _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
    var item = _todoRepository.Find(id);
    if (item == null)
    {
        _logger.LogWarning(LoggingEvents.GetItemNotFound, "GetById({ID}) NOT FOUND", id);
        return NotFound();
    }
    return new ObjectResult(item);
}
```

O exemplo acima cria logs com `Information` e `Warning` como o *nível* e a classe `TodoController` como a *categoria*.

O *nível* de log indica a gravidade do evento registrado. A *categoria* do log é uma cadeia de caracteres associada a cada log. A instância `ILogger<T>` cria logs que têm o nome totalmente qualificado do tipo `T` como a categoria.

[Níveis](#) e [categorias](#) serão explicados com mais detalhes posteriormente neste artigo.

## Criar logs na inicialização

Para gravar logs na classe `Startup`, inclua um parâmetro `ILogger` na assinatura de construtor:

```
public class Startup
{
    private readonly ILogger _logger;

    public Startup(IConfiguration configuration, ILogger<Startup> logger)
    {
        Configuration = configuration;
        _logger = logger;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc();

        // Add our repository type
        services.AddSingleton<ITodoRepository, TodoRepository>();
        _logger.LogInformation("Added TodoRepository to services");
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        if (env.IsDevelopment())
        {
            _logger.LogInformation("In Development environment");
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Error");
            app.UseHsts();
        }

        app.UseHttpsRedirection();
        app.UseStaticFiles();
        app.UseCookiePolicy();

        app.UseMvc();
    }
}
```

## Criar logs no programa

Para gravar logs na classe `Program`, obtenha uma instância `ILogger` da DI:

```

public static void Main(string[] args)
{
    var host = BuildWebHost(args);

    var todoRepository = host.Services.GetRequiredService<ITodoRepository>();
    todoRepository.Add(new Core.Model.TodoItem() { Name = "Feed the dog" });
    todoRepository.Add(new Core.Model.TodoItem() { Name = "Walk the dog" });

    var logger = host.Services.GetRequiredService<ILogger<Program>>();
    logger.LogInformation("Seeded the database.");

    host.Run();
}

public static IWebHost BuildWebHost(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .ConfigureLogging(logging =>
    {
        logging.ClearProviders();
        logging.AddConsole();
    })
        .Build();

```

## Sem métodos de agente assíncronos

O registro em log deve ser tão rápido que não justifique o custo de desempenho de código assíncrono. Se o armazenamento de dados em log estiver lento, não grave diretamente nele. Grave as mensagens de log em um repositório rápido primeiro e, depois, mova-as para um repositório lento. Por exemplo, registre em uma fila de mensagens que seja lida e persistida em um armazenamento lento por outro processo.

## Configuração

A configuração do provedor de logs é fornecida por um ou mais provedores de sincronização:

- Formatos de arquivo (INI, JSON e XML).
- Argumentos de linha de comando.
- Variáveis de ambiente.
- Objetos do .NET na memória.
- O armazenamento do [Secret Manager](#) não criptografado.
- Um repositório de usuário criptografado, como o [Azure Key Vault](#).
- Provedores personalizados (instalados ou criados).

Por exemplo, a configuração de log geralmente é fornecida pela seção `Logging` dos arquivos de configurações do aplicativo. O exemplo a seguir mostra o conteúdo de um típico arquivo `appsettings.Development.json`:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    },
    "Console": {
      "IncludeScopes": true
    }
  }
}
```

A propriedade `Logging` pode ter `LogLevel` e propriedades do provedor de logs (o Console é mostrado).

A propriedade `LogLevel` em `Logging` especifica o nível mínimo para log nas categorias selecionadas. No exemplo, as categorias `System` e `Microsoft` têm log no nível `Information`, e todas as outras no nível `Debug`.

Outras propriedades em `Logging` especificam provedores de logs. O exemplo se refere ao provedor de Console.

Se um provedor oferecer suporte a [escopos de log](#), `IncludeScopes` indicará se eles estão habilitados. Uma propriedade de provedor (como `Console`, no exemplo) também pode especificar uma propriedade `LogLevel`. `LogLevel` em um provedor especifica os níveis de log para esse provedor.

Se os níveis forem especificados em `Logging.{providername}.LogLevel`, eles substituirão o que estiver definido em `Logging.LogLevel`.

```
{  
  "Logging": {  
    "LogLevel": {  
      "Default": "Debug",  
      "System": "Information",  
      "Microsoft": "Information"  
    }  
  }  
}
```

Chaves `LogLevel` representam nomes de log. A chave `Default` aplica-se a logs não listados de forma explícita. O valor representa o nível de log aplicado ao log fornecido.

Saiba mais sobre como implementar provedores de configuração em [Configuração no ASP.NET Core](#).

## Exemplo de saída de registro em log

Com o código de exemplo mostrado na seção anterior, os logs serão exibidos no console quando o aplicativo for executado pela linha de comando. Aqui está um exemplo da saída do console:

```
info: Microsoft.AspNetCore.Hosting.Internal.WebHost[1]  
  Request starting HTTP/1.1 GET http://localhost:5000/api/todo/0  
info: Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker[1]  
  Executing action method TodoApi.Controllers.TodoController.GetById (TodoApi) with arguments (0) -  
ModelState is Valid  
info: TodoApi.Controllers.TodoController[1002]  
  Getting item 0  
warn: TodoApi.Controllers.TodoController[4000]  
  GetById(0) NOT FOUND  
info: Microsoft.AspNetCore.Mvc.StatusCodeResult[1]  
  Executing HttpStatusCodeResult, setting HTTP status code 404  
info: Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker[2]  
  Executed action TodoApi.Controllers.TodoController.GetById (TodoApi) in 42.9286ms  
info: Microsoft.AspNetCore.Hosting.Internal.WebHost[2]  
  Request finished in 148.889ms 404
```

Os logs anteriores foram gerados por meio de uma solicitação HTTP Get para o aplicativo de exemplo em `http://localhost:5000/api/todo/0`.

Veja um exemplo de como os mesmos logs aparecem na janela Depuração quando você executa o aplicativo de exemplo no Visual Studio:

```
Microsoft.AspNetCore.Hosting.Internal.WebHost:Information: Request starting HTTP/1.1 GET
http://localhost:53104/api/todo/0
Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker:Information: Executing action method
TodoApi.Controllers.TodoController.GetById (TodoApi) with arguments (0) - ModelState is Valid
TodoApi.Controllers.TodoController:Information: Getting item 0
TodoApi.Controllers.TodoController:Warning: GetById(0) NOT FOUND
Microsoft.AspNetCore.Mvc.StatusCodeResult:Information: Executing HttpStatusCodeResult, setting HTTP status code
404
Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker:Information: Executed action
TodoApi.Controllers.TodoController.GetById (TodoApi) in 152.5657ms
Microsoft.AspNetCore.Hosting.Internal.WebHost:Information: Request finished in 316.3195ms 404
```

Os logs criados pelas chamadas `ILogger` mostradas na seção anterior começam com "TodoApi.Controllers.TodoController". Os logs que começam com categorias "Microsoft" são de código da estrutura ASP.NET Core. O ASP.NET Core e o código do aplicativo estão usando a mesma API de registro em log e os mesmos provedores.

O restante deste artigo explica alguns detalhes e opções para registro em log.

## Pacotes NuGet

As interfaces `ILogger` e `ILoggerFactory` estão em [Microsoft.Extensions.Logging.Abstractions](#) e as implementações padrão para elas estão em [Microsoft.Extensions.Logging](#).

## Categoria de log

Quando um objeto `ILogger` é criado, uma *categoria* é especificada para ele. Essa categoria é incluída em cada mensagem de log criada por essa instância de `Ilogger`. A categoria pode ser qualquer cadeia de caracteres, mas a convenção é usar o nome da classe, como "TodoApi.Controllers.TodoController".

Use `ILogger<T>` para obter uma instância `ILogger` que usa o nome de tipo totalmente qualificado do `T` como a categoria:

```
public class TodoController : Controller
{
    private readonly ITodoRepository _todoRepository;
    private readonly ILogger _logger;

    public TodoController(ITodoRepository todoRepository,
        ILogger<TodoController> logger)
    {
        _todoRepository = todoRepository;
        _logger = logger;
    }
}
```

```
public class TodoController : Controller
{
    private readonly ITodoRepository _todoRepository;
    private readonly ILogger _logger;

    public TodoController(ITodoRepository todoRepository,
        ILogger<TodoController> logger)
    {
        _todoRepository = todoRepository;
        _logger = logger;
    }
}
```

Para especificar explicitamente a categoria, chame `ILoggerFactory.CreateLogger`:

```

public class TodoController : Controller
{
    private readonly ITodoRepository _todoRepository;
    private readonly ILogger _logger;

    public TodoController(ITodoRepository todoRepository,
        ILoggerFactory logger)
    {
        _todoRepository = todoRepository;
        _logger = logger.CreateLogger("TodoApiSample.Controllers.TodoController");
    }
}

```

```

public class TodoController : Controller
{
    private readonly ITodoRepository _todoRepository;
    private readonly ILogger _logger;

    public TodoController(ITodoRepository todoRepository,
        ILoggerFactory logger)
    {
        _todoRepository = todoRepository;
        _logger = logger.CreateLogger("TodoApiSample.Controllers.TodoController");
    }
}

```

`ILogger<T>` é equivalente a chamar `createLogger` com o nome de tipo totalmente qualificado de `T`.

## Nível de log

Todo log especifica um valor `LogLevel`. O nível de log indica a gravidade ou importância. Por exemplo, você pode gravar um log `Information` quando um método é finalizado normalmente e um log `Warning` quando um método retorna um código de status `404 Não Encontrado`.

O código a seguir cria os logs `Information` e `Warning`:

```

public IActionResult GetById(string id)
{
    _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
    var item = _todoRepository.Find(id);
    if (item == null)
    {
        _logger.LogWarning(LoggingEvents.GetItemNotFound, ".GetById({ID}) NOT FOUND", id);
        return NotFound();
    }
    return new ObjectResult(item);
}

```

```

public IActionResult GetById(string id)
{
    _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
    var item = _todoRepository.Find(id);
    if (item == null)
    {
        _logger.LogWarning(LoggingEvents.GetItemNotFound, ".GetById({ID}) NOT FOUND", id);
        return NotFound();
    }
    return new ObjectResult(item);
}

```

No código anterior, o primeiro parâmetro é a [ID de evento de log](#). O segundo parâmetro é um modelo de

mensagem com espaços reservados para valores de argumento fornecidos pelos parâmetros de método restantes. Os parâmetros de método serão explicados com posteriormente neste artigo, na [seção de modelos de mensagem](#).

Os métodos de log que incluem o nível no nome do método (por exemplo, `.LogInformation` e `.LogWarning`) são [métodos de extensão para ILogger](#). Esses métodos chamam um método `Log` que recebe um parâmetro `LogLevel`. Você pode chamar o método `Log` diretamente em vez de um desses métodos de extensão, mas a sintaxe é relativamente complicada. Para saber mais, veja [ILogger](#) e o [código-fonte de extensões de agente](#).

O ASP.NET Core define os seguintes níveis de log, ordenados aqui da menor para a maior gravidade.

- Trace = 0

Para obter informações que normalmente são valiosas somente para depuração. Essas mensagens podem conter dados confidenciais de aplicativos e, portanto, não devem ser habilitadas em um ambiente de produção. *Desabilitado por padrão*.

- Debug = 1

Para obter informações que possam ser úteis durante o desenvolvimento e a depuração. Exemplo:

`Entering method Configure with flag set to true.` habilite logs de nível `Debug` em produção somente ao solucionar problemas, devido ao alto volume de logs.

- Information = 2

Para rastrear o fluxo geral do aplicativo. Esses logs normalmente têm algum valor a longo prazo. Exemplo:

`Request received for path /api/todo`

- Warning = 3

Para eventos anormais ou inesperados no fluxo de aplicativo. Eles podem incluir erros ou outras condições que não fazem com que o aplicativo pare, mas que talvez precisem ser investigados. Exceções manipuladas são um local comum para usar o nível de log `Warning`. Exemplo:

`FileNotFoundException for file quotes.txt.`

- Error = 4

Para erros e exceções que não podem ser manipulados. Essas mensagens indicam uma falha na atividade ou na operação atual (como a solicitação HTTP atual) e não uma falha em todo o aplicativo. Mensagem de log de exemplo: `Cannot insert record due to duplicate key violation.`

- Critical = 5

Para falhas que exigem atenção imediata. Exemplos: cenários de perda de dados, espaço em disco insuficiente.

Use o nível de log para controlar a quantidade de saída de log que é gravada em uma mídia de armazenamento específica ou em uma janela de exibição. Por exemplo:

- Em produção, envie `Trace` por meio do nível `Information` para um armazenamento de dados com volume. Envie `Warning` por meio de `Critical` para um armazenamento de dados com valor.
- Durante o desenvolvimento, envie `Warning` por meio `Critical` para o console e adicione `Trace` por meio de `Information` ao solucionar problemas.

A seção [Filtragem de log](#) mais adiante neste artigo explicará como controlar os níveis de log que um provedor manipula.

O ASP.NET Core grava logs para eventos de estrutura. Os exemplos de log anteriores neste artigo excluíram logs abaixo do nível `Information`, portanto, logs de nível `Debug` ou `Trace` não foram criados. Veja um exemplo de logs de console produzidos por meio da execução do aplicativo de exemplo configurado para mostrar logs `Debug`:

```

info: Microsoft.AspNetCore.Hosting.Internal.WebHost[1]
      Request starting HTTP/1.1 GET http://localhost:62555/api/todo/0
dbug: Microsoft.AspNetCore.Routing.Tree.TreeRouter[1]
      Request successfully matched the route with name 'GetTodo' and template 'api/Todo/{id}'.
dbug: Microsoft.AspNetCore.Mvc.Internal.ActionSelector[2]
      Action 'TodoApi.Controllers.TodoController.Update (TodoApi)' with id '089d59b6-92ec-472d-b552-cc613dfd625d' did not match the constraint 'Microsoft.AspNetCore.Mvc.Internal.HttpMethodActionConstraint'
dbug: Microsoft.AspNetCore.Mvc.Internal.ActionSelector[2]
      Action 'TodoApi.Controllers.TodoController.Delete (TodoApi)' with id 'f3476abe-4bd9-4ad3-9261-3ead09607366' did not match the constraint 'Microsoft.AspNetCore.Mvc.Internal.HttpMethodActionConstraint'
dbug: Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker[1]
      Executing action TodoApi.Controllers.TodoController.GetById (TodoApi)
info: Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker[1]
      Executing action method TodoApi.Controllers.TodoController.GetById (TodoApi) with arguments (0) -
ModelState is Valid
info: TodoApi.Controllers.TodoController[1002]
      Getting item 0
warn: TodoApi.Controllers.TodoController[4000]
      GetById(0) NOT FOUND
dbug: Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker[2]
      Executed action method TodoApi.Controllers.TodoController.GetById (TodoApi), returned result
Microsoft.AspNetCore.Mvc.NotFoundResult.
info: Microsoft.AspNetCore.Mvc.StatusCodeResult[1]
      Executing HttpStatusCodeResult, setting HTTP status code 404
info: Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker[2]
      Executed action TodoApi.Controllers.TodoController.GetById (TodoApi) in 0.8788ms
dbug: Microsoft.AspNetCore.Server.Kestrel[9]
      Connection id "0HL6L7NEFF2QD" completed keep alive response.
info: Microsoft.AspNetCore.Hosting.Internal.WebHost[2]
      Request finished in 2.7286ms 404

```

## ID de evento de log

Cada log pode especificar uma *ID do evento*. O aplicativo de exemplo faz isso usando uma classe `LoggingEvents` definida localmente:

```

public IActionResult GetById(string id)
{
    _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
    var item = _todoRepository.Find(id);
    if (item == null)
    {
        _logger.LogWarning(LoggingEvents.GetItemNotFound, ".GetById({ID}) NOT FOUND", id);
        return NotFound();
    }
    return new ObjectResult(item);
}

```

```

public class LoggingEvents
{
    public const int GenerateItems = 1000;
    public const int ListItems = 1001;
    public const int GetItem = 1002;
    public const int InsertItem = 1003;
    public const int UpdateItem = 1004;
    public const int DeleteItem = 1005;

    public const int GetItemNotFound = 4000;
    public const int UpdateItemNotFound = 4001;
}

```

```
public IActionResult GetById(string id)
{
    _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
    var item = _todoRepository.Find(id);
    if (item == null)
    {
        _logger.LogWarning(LoggingEvents.GetItemNotFound, "GetById({ID}) NOT FOUND", id);
        return NotFound();
    }
    return new ObjectResult(item);
}
```

```
public class LoggingEvents
{
    public const int GenerateItems = 1000;
    public const int ListItems = 1001;
    public const int GetItem = 1002;
    public const int InsertItem = 1003;
    public const int UpdateItem = 1004;
    public const int DeleteItem = 1005;

    public const int GetItemNotFound = 4000;
    public const int UpdateItemNotFound = 4001;
}
```

Uma ID de evento associa um conjunto de eventos. Por exemplo, todos os logs relacionados à exibição de uma lista de itens em uma página podem ser 1001.

O provedor de logs pode armazenar a ID do evento em um campo de ID na mensagem de log ou não armazenar. O provedor de Depuração não mostra IDs de eventos. O provedor de console mostra IDs de evento entre colchetes após a categoria:

```
info: TodoApi.Controllers.TodoController[1002]
      Getting item invalidid
warn: TodoApi.Controllers.TodoController[4000]
      GetById(invalidid) NOT FOUND
```

## Modelo de mensagem de log

Cada log especifica um modelo de mensagem. O modelo de mensagem pode conter espaços reservados para os quais são fornecidos argumentos. Use nomes para os espaços reservados, não números.

```
public IActionResult GetById(string id)
{
    _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
    var item = _todoRepository.Find(id);
    if (item == null)
    {
        _logger.LogWarning(LoggingEvents.GetItemNotFound, "GetById({ID}) NOT FOUND", id);
        return NotFound();
    }
    return new ObjectResult(item);
}
```

```
public IActionResult GetById(string id)
{
    _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
    var item = _todoRepository.Find(id);
    if (item == null)
    {
        _logger.LogWarning(LoggingEvents.GetItemNotFound, ".GetById({ID}) NOT FOUND", id);
        return NotFound();
    }
    return new ObjectResult(item);
}
```

A ordem dos espaços reservados e não de seus nomes, determina quais parâmetros serão usados para fornecer seus valores. No código a seguir, observe que os nomes de parâmetro estão fora de sequência no modelo de mensagem:

```
string p1 = "parm1";
string p2 = "parm2";
_logger.LogInformation("Parameter values: {p2}, {p1}", p1, p2);
```

Esse código cria uma mensagem de log com os valores de parâmetro na sequência:

```
Parameter values: parm1, parm2
```

A estrutura de registros funciona dessa maneira para que os provedores de logs possam implementar [registro em log semântico, também conhecido como registro em log estruturado](#). Os próprios argumentos são passados para o sistema de registro em log, não apenas o modelo de mensagem formatado. Essas informações permitem que os provedores de log armazenem os valores de parâmetro como campos. Por exemplo, suponha que as chamadas de método do agente sejam assim:

```
_logger.LogInformation("Getting item {ID} at {RequestTime}", id, DateTime.Now);
```

Se você estiver enviando os logs para o Armazenamento de Tabelas do Azure, cada entidade da Tabela do Azure poderá ter propriedades `ID` e `RequestTime`, o que simplificará as consultas nos dados de log. Uma consulta pode encontrar todos os logs em determinado intervalo de `RequestTime` sem analisar o tempo limite da mensagem de texto.

## Exceções de registro em log

Os métodos de agente têm sobrecargas que permitem que você passe uma exceção, como no exemplo a seguir:

```
catch (Exception ex)
{
    _logger.LogWarning(LoggingEvents.GetItemNotFound, ex, ".GetById({ID}) NOT FOUND", id);
    return NotFound();
}
return new ObjectResult(item);
```

```

    catch (Exception ex)
    {
        _logger.LogWarning(LoggingEvents.GetItemNotFound, ex, " GetById({ID}) NOT FOUND", id);
        return NotFound();
    }
    return new ObjectResult(item);
}

```

Provedores diferentes manipulam as informações de exceção de maneiras diferentes. Aqui está um exemplo da saída do provedor Depuração do código mostrado acima.

```

TodoApi.Controllers.TodoController:Warning: GetById(036dd898-fb01-47e8-9a65-f92eb73cf924) NOT FOUND

System.Exception: Item not found exception.
   at TodoApi.Controllers.TodoController.GetById(String id) in
C:\logging\sample\src\TodoApi\Controllers\TodoController.cs:line 226

```

## Filtragem de log

Você pode especificar um nível de log mínimo para um provedor e uma categoria específicos ou para todos os provedores ou todas as categorias. Os logs abaixo do nível mínimo não serão passados para esse provedor, para que não sejam exibidos ou armazenados.

Para suprimir todos os logs, especifique `LogLevel.None` como o nível de log mínimo. O valor inteiro de `LogLevel.None` é 6, que é maior do que `LogLevel.Critical` (5).

### Criar regras de filtro na configuração

O código do modelo de projeto chama `CreateDefaultBuilder` para configurar o registro em log para os provedores Console e Depuração. O método `CreateDefaultBuilder` também configura o registro em log para procurar a configuração em uma seção `Logging`, usando código semelhante ao seguinte:

```

public static void Main(string[] args)
{
    var webHost = new WebHostBuilder()
        .UseKestrel()
        .UseContentRoot(Directory.GetCurrentDirectory())
        .ConfigureAppConfiguration((hostingContext, config) =>
    {
        var env = hostingContext.HostingEnvironment;
        config.AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
            .AddJsonFile($"appsettings.{env.EnvironmentName}.json",
                optional: true, reloadOnChange: true);
        config.AddEnvironmentVariables();
    })
    .ConfigureLogging((hostingContext, logging) =>
    {
        logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));
        logging.AddConsole();
        logging.AddDebug();
        logging.AddEventSourceLogger();
    })
    .UseStartup<Startup>()
    .Build();

    webHost.Run();
}

```

Os dados de configuração especificam níveis de log mínimo por provedor e por categoria, como no exemplo a seguir:

```
{
    "Logging": {
        "Debug": {
            "LogLevel": {
                "Default": "Information"
            }
        },
        "Console": {
            "IncludeScopes": false,
            "LogLevel": {
                "Microsoft.AspNetCore.Mvc.Razor.Internal": "Warning",
                "Microsoft.AspNetCore.Mvc.Razor": "Debug",
                "Microsoft.AspNetCore.Mvc.Razor": "Error",
                "Default": "Information"
            }
        },
        "LogLevel": {
            "Default": "Debug"
        }
    }
}
```

Este JSON cria seis regras de filtro, uma para o provedor Depuração, quatro para o provedor Console e uma para todos os provedores. Apenas uma regra é escolhida para cada provedor quando um objeto `ILogger` é criado.

## Regras de filtro no código

O exemplo a seguir mostra como registrar regras de filtro no código:

```
WebHost.CreateDefaultBuilder(args)
    .UseStartup<Startup>()
    .ConfigureLogging(logging =>
        logging.AddFilter("System", LogLevel.Debug)
        .AddFilter<DebugLoggerProvider>("Microsoft", LogLevel.Trace))
    .Build();
```

O segundo `AddFilter` especifica o provedor Depuração usando seu nome de tipo. O primeiro `AddFilter` se aplica a todos os provedores porque ele não especifica um tipo de provedor.

## Como as regras de filtragem são aplicadas

Os dados de configuração e o código `AddFilter`, mostrados nos exemplos anteriores, criam as regras mostradas na tabela a seguir. As primeiras seis vêm do exemplo de configuração e as últimas duas vêm do exemplo de código.

NÚMERO	PROVIDER	CATEGORIAS QUE COMEÇAM COM...	NÍVEL DE LOG MÍNIMO
1	Depurar	Todas as categorias	Informações
2	Console	Microsoft.AspNetCore.Mvc.Razor.Internal	Aviso
3	Console	Microsoft.AspNetCore.Mvc.Razor.Razor	Depurar
4	Console	Microsoft.AspNetCore.Mvc.Razor.Razor	Erro
5	Console	Todas as categorias	Informações

NÚMERO	PROVIDER	CATEGORIAS QUE COMEÇAM COM...	NÍVEL DE LOG MÍNIMO
6	Todos os provedores	Todas as categorias	Depurar
7	Todos os provedores	Sistema	Depurar
8	Depurar	Microsoft	Rastrear

Quando um objeto `ILogger` é criado, o objeto `ILoggerFactory` seleciona uma única regra por provedor para aplicar a esse agente. Todas as mensagens gravadas pela instância `ILogger` são filtradas com base nas regras selecionadas. A regra mais específica possível para cada par de categoria e provedor é selecionada dentre as regras disponíveis.

O algoritmo a seguir é usado para cada provedor quando um `ILogger` é criado para uma determinada categoria:

- Selecione todas as regras que correspondem ao provedor ou seu alias. Se nenhuma correspondência for encontrada, selecione todas as regras com um provedor vazio.
- Do resultado da etapa anterior, selecione as regras com o prefixo de categoria de maior correspondência. Se nenhuma correspondência for encontrada, selecione todas as regras que não especificam uma categoria.
- Se várias regras forem selecionadas, use a **última**.
- Se nenhuma regra for selecionada, use `MinimumLevel`.

Com a lista anterior de regras, suponha que você crie um objeto `ILogger` para a categoria "Microsoft.AspNetCore.Mvc.Razor.RazorViewEngine":

- Para o provedor Depuração as regras 1, 6 e 8 se aplicam. A regra 8 é mais específica, portanto é a que será selecionada.
- Para o provedor Console as regras 3, 4, 5 e 6 se aplicam. A regra 3 é a mais específica.

A instância `ILogger` resultante envia logs de nível `Trace` e superior para o provedor Depuração. Logs de nível `Debug` e superior são enviados para o provedor Console.

## Aliases de provedor

Cada provedor define um *alias* que pode ser usado na configuração no lugar do nome do tipo totalmente qualificado. Para os provedores internos, use os seguintes aliases:

- Console
- Depurar
- EventLog
- AzureAppServices
- TraceSource
- EventSource

## Nível mínimo padrão

Há uma configuração de nível mínimo que entra em vigor somente se nenhuma regra de código ou de configuração se aplicar a um provedor e uma categoria determinados. O exemplo a seguir mostra como definir o nível mínimo:

```
WebHost.CreateDefaultBuilder(args)
    .UseStartup<Startup>()
    .ConfigureLogging(logging => logging.SetMinimumLevel(LogLevel.Warning))
    .Build();
```

Se você não definir explicitamente o nível mínimo, o valor padrão será `Information`, o que significa que logs `Trace` e `Debug` serão ignorados.

## Funções de filtro

Uma função de filtro é invocada para todos os provedores e categorias que não têm regras atribuídas a eles por configuração ou código. O código na função tem acesso ao tipo de provedor, à categoria e ao nível de log. Por exemplo:

```
WebHost.CreateDefaultBuilder(args)
    .UseStartup<Startup>()
    .ConfigureLogging(logBuilder =>
{
    logBuilder.AddFilter((provider, category, logLevel) =>
    {
        if (provider == "Microsoft.Extensions.Logging.Console.ConsoleLoggerProvider" &&
            category == "TodoApiSample.Controllers.TodoController")
        {
            return false;
        }
        return true;
    });
})
.Build();
```

Alguns provedores de log permitem especificar quando os logs devem ser gravados em uma mídia de armazenamento ou ignorados, com base no nível de log e na categoria.

Os métodos de extensão `AddConsole` e `AddDebug` fornecem sobrecargas que aceitam critérios de filtragem. O código de exemplo a seguir faz com que o provedor de console ignore os logs abaixo do nível `Warning`, enquanto o provedor Depuração ignora os logs criados pela estrutura.

```
public void Configure(IApplicationBuilder app,
    IHostingEnvironment env,
    ILoggerFactory loggerFactory)
{
    loggerFactory
        .AddConsole(LogLevel.Warning)
        .AddDebug((category, logLevel) => (category.Contains("TodoApi") && logLevel >= LogLevel.Trace));
```

O método `AddEventLog` tem uma sobrecarga que recebe uma instância `EventLogSettings`, que pode conter uma função de filtragem em sua propriedade `Filter`. O provedor TraceSource não fornece nenhuma dessas sobrecargas, porque seu nível de registro em log e outros parâmetros são baseados no `SourceSwitch` e no `TraceListener` que ele usa.

Para definir regras de filtragem para todos os provedores registrados com uma instância `ILoggerFactory`, use o método de extensão `WithFilter`. O exemplo abaixo limita os logs de estrutura (categoria começa com "Microsoft" ou "Sistema") a avisos, enquanto registra em nível de depuração os logs criados por código de aplicativo.

```

public void Configure(IApplicationBuilder app,
    IHostingEnvironment env,
    ILoggerFactory loggerFactory)
{
    loggerFactory
        .WithFilter(new FilterLoggerSettings
        {
            { "Microsoft", LogLevel.Warning },
            { "System", LogLevel.Warning },
            { "ToDoApi", LogLevel.Debug }
        })
        .AddConsole()
        .AddDebug();
}

```

Para impedir a gravação de logs, especifique `LogLevel.None` como o nível de log mínimo. O valor inteiro de `LogLevel.None` é 6, que é maior do que `LogLevel.Critical` (5).

O método de extensão `WithFilter` é fornecido pelo pacote NuGet [Microsoft.Extensions.Logging.Filter](#). O método retorna um nova instância `ILoggerFactory`, que filtrará as mensagens de log passadas para todos os provedores de agente registrados com ela. Isso não afeta nenhuma outra instância de `ILoggerFactory`, incluindo a instância de `ILoggerFactory` original.

## Categorias e níveis de sistema

Veja algumas categorias usadas pelo ASP.NET Core e Entity Framework Core, com anotações sobre quais logs esperar delas:

CATEGORIA	OBSERVAÇÕES
Microsoft.AspNetCore	Diagnóstico geral de ASP.NET Core.
Microsoft.AspNetCore.DataProtection	Quais chaves foram consideradas, encontradas e usadas.
Microsoft.AspNetCore.HostFiltering	Hosts permitidos.
Microsoft.AspNetCore.Hosting	Quanto tempo levou para que as solicitações de HTTP fossem concluídas e em que horário foram iniciadas. Quais assemblies de inicialização de hospedagem foram carregados.
Microsoft.AspNetCore.Mvc	Diagnóstico do MVC e Razor. Model binding, execução de filtro, compilação de exibição, seleção de ação.
Microsoft.AspNetCore.Routing	Informações de correspondência de rotas.
Microsoft.AspNetCore.Server	Respostas de início, parada e atividade da conexão. Informações sobre o certificado HTTPS.
Microsoft.AspNetCore.StaticFiles	Arquivos atendidos.
Microsoft.EntityFrameworkCore	Diagnóstico geral do Entity Framework Core. Atividade e configuração do banco de dados, detecção de alterações, migrações.

## Escopos de log

Um *escopo* pode agrupar um conjunto de operações lógicas. Esse agrupamento pode ser usado para anexar os

mesmos dados para cada log criado como parte de um conjunto. Por exemplo, todo log criado como parte do processamento de uma transação pode incluir a ID da transação.

Um escopo é um tipo `IDisposable` retornado pelo método `BeginScope` e que dura até que seja descartado. Use um escopo por meio do encapsulamento de chamadas de agente em um bloco `using`:

```
public IActionResult GetById(string id)
{
    TodoItem item;
    using (_logger.BeginScope("Message attached to logs created in the using block"))
    {
        _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
        item = _todoRepository.Find(id);
        if (item == null)
        {
            _logger.LogWarning(LoggingEvents.GetItemNotFound, "GetById({ID}) NOT FOUND", id);
            return NotFound();
        }
    }
    return new ObjectResult(item);
}
```

O código a seguir habilita os escopos para o provedor de console:

*Program.cs:*

```
.ConfigureLogging((hostingContext, logging) =>
{
    logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));
    logging.AddConsole(options => options.IncludeScopes = true);
    logging.AddDebug();
})
```

#### NOTE

A configuração da opção de agente de console `IncludeScopes` é necessária para habilitar o registro em log baseado em escopo.

Saiba mais sobre como configurar na seção [Configuração](#).

*Program.cs:*

```
.ConfigureLogging((hostingContext, logging) =>
{
    logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));
    logging.AddConsole(options => options.IncludeScopes = true);
    logging.AddDebug();
})
```

#### NOTE

A configuração da opção de agente de console `IncludeScopes` é necessária para habilitar o registro em log baseado em escopo.

*Startup.cs:*

```
public void Configure(IApplicationBuilder app,
    IHostingEnvironment env,
    ILoggerFactory loggerFactory)
{
    loggerFactory
        .AddConsole(includeScopes: true)
        .AddDebug();
```

Cada mensagem de log inclui as informações com escopo definido:

```
info: TodoApi.Controllers.TodoController[1002]
    => RequestId:0HKV9C49II9CK RequestPath:/api/todo/0 => TodoApi.Controllers.TodoController.GetById
(TodoApi) => Message attached to logs created in the using block
    Getting item 0
warn: TodoApi.Controllers.TodoController[4000]
    => RequestId:0HKV9C49II9CK RequestPath:/api/todo/0 => TodoApi.Controllers.TodoController.GetById
(TodoApi) => Message attached to logs created in the using block
    GetById(0) NOT FOUND
```

## Provedores de log internos

O ASP.NET Core vem com os seguintes provedores:

- [Console](#)
- [Depurar](#)
- [EventSource](#)
- [EventLog](#)
- [TraceSource](#)

As opções de [Registro em log no Azure](#) serão abordadas mais adiante, neste artigo.

Para saber mais sobre log de stdout, confira [Solucionar problemas do ASP.NET Core no IIS](#) e [Solucionar problemas no ASP.NET Core no Serviço de Aplicativo do Azure](#).

### Provedor do console

O pacote de provedor [Microsoft.Extensions.Logging.Console](#) envia a saída de log para o console.

```
logging.AddConsole();
```

```
loggerFactory.AddConsole();
```

As [sobrecargas do AddConsole](#) permitem que você passe um nível de log mínimo, uma função de filtro e um valor booleano que indica se escopos são compatíveis. Outra opção é passar um objeto `IConfiguration`, que pode especificar suporte de escopos e níveis de log.

O provedor de console tem um impacto significativo no desempenho e geralmente não é adequado para uso em produção.

Quando você cria um novo projeto no Visual Studio, o método `AddConsole` tem essa aparência:

```
loggerFactory.AddConsole(Configuration.GetSection("Logging"));
```

Esse código se refere à seção `Logging` do arquivo `appSettings.json`:

```
{  
  "Logging": {  
    "Console": {  
      "IncludeScopes": false  
    },  
    "LogLevel": {  
      "Default": "Debug",  
      "System": "Information",  
      "Microsoft": "Information"  
    }  
  }  
}
```

As configurações mostradas limitam os logs de estrutura a avisos, permitindo que o aplicativo faça registros no nível de depuração, conforme explicado na [Filtragem de log](#) seção. Para obter mais informações, consulte [Configuração](#).

Para ver a saída de registro em log de console, abra um prompt de comando na pasta do projeto e execute o seguinte comando:

```
dotnet run
```

## Depurar provedor

O pacote de provedor [Microsoft.Extensions.Logging.Debug](#) grava a saída de log usando a classe [System.Diagnostics.Debug](#) (chamadas de método `Debug.WriteLine`).

No Linux, esse provedor grava logs em `/var/log/message`.

```
logging.AddDebug();
```

```
loggerFactory.AddDebug();
```

As [sobrecargas de AddDebug](#) permitem que você passe um nível de log mínimo ou uma função de filtro.

## Provedor EventSource

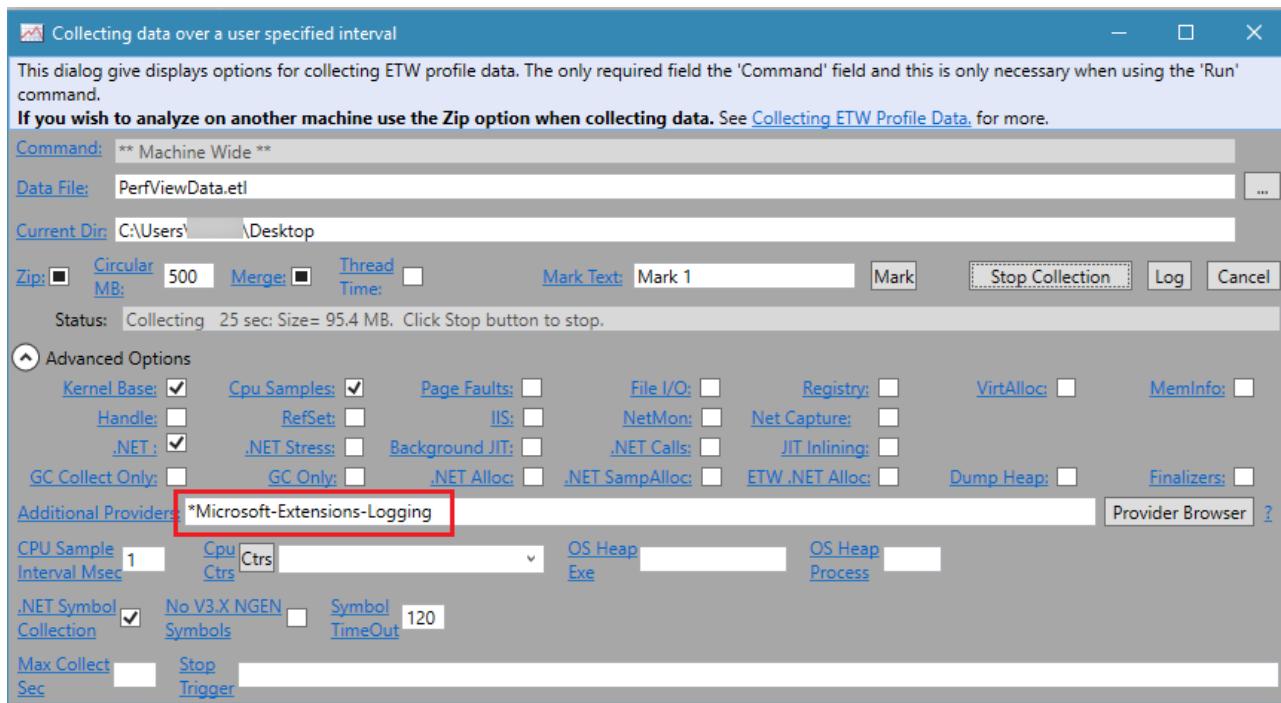
Para aplicativos que se destinam ao ASP.NET Core 1.1.0 ou posterior, o pacote de provedor [Microsoft.Extensions.Logging.EventSource](#) pode implementar o rastreamento de eventos. No Windows, ele usa [ETW](#). O provedor é multiplataforma, mas ainda não há ferramentas de coleta e exibição de eventos para Linux ou macOS.

```
logging.AddEventSourceLogger();
```

```
loggerFactory.AddEventSourceLogger();
```

Uma boa maneira de coletar e exibir logs é usar o [utilitário PerfView](#). Há outras ferramentas para exibir os logs do ETW, mas o PerfView proporciona a melhor experiência para trabalhar com os eventos de ETW emitidos pelo ASP.NET.

Para configurar o PerfView para coletar eventos registrados por esse provedor, adicione a cadeia de caracteres `*Microsoft-Extensions-Logging` à lista **Provedores Adicionais**. (Não se esqueça do asterisco no início da cadeia de caracteres).



## Provedor EventLog do Windows

O pacote de provedor [Microsoft.Extensions.Logging.EventLog](#) envia a saída de log para o Log de Eventos do Windows.

```
logging.AddEventLog();
```

```
loggerFactory.AddEventLog();
```

As [sobrecargas de AddEventLog](#) permitem que você passe `EventLogSettings` ou um nível de log mínimo.

## Provedor TraceSource

O pacote de provedor [Microsoft.Extensions.Logging.TraceSource](#) usa as bibliotecas e provedores de [TraceSource](#).

```
logging.AddTraceSource(sourceSwitchName);
```

```
loggerFactory.AddTraceSource(sourceSwitchName);
```

As [sobrecargas de AddTraceSource](#) permitem que você passe um comutador de fonte e um ouvinte de rastreamento.

Para usar esse provedor, o aplicativo deve ser executado no .NET Framework (em vez do .NET Core). O provedor pode rotear mensagens a uma variedade de [ouvintes](#), como o [TextWriterTraceListener](#) usado no aplicativo de exemplo.

O exemplo a seguir configura um provedor `TraceSource` que registra mensagens `Warning` e superiores na janela de console.

```
public void Configure(IApplicationBuilder app,
    IHostingEnvironment env,
    ILoggerFactory loggerFactory)
{
    loggerFactory
        .AddDebug();

    // add Trace Source logging
    var testSwitch = new SourceSwitch("sourceSwitch", "Logging Sample");
    testSwitch.Level = SourceLevels.Warning;
    loggerFactory.AddTraceSource(testSwitch,
        new TextWriterTraceListener(writer: Console.Out));
}
```

## Registro em log no Azure

Para saber mais sobre registro em log no Azure, consulte as seguintes seções:

- [Provedor do Serviço de Aplicativo do Azure](#)
- [Fluxo de log do Azure](#)
- [Log de rastreamento do Azure Application Insights](#)

### Provedor do Serviço de Aplicativo do Azure

O pacote de provedor [Microsoft.Extensions.Logging.AzureAppServices](#) grava logs em arquivos de texto no sistema de arquivos de um aplicativo do Serviço de Aplicativo do Azure e no [armazenamento de blobs](#) em uma conta de Armazenamento do Azure. O pacote de provedor está disponível para aplicativos destinados ao .NET Core 1.1 ou posterior.

Se você estiver direcionando para o .NET Core, observe os seguintes pontos:

- O pacote de provedor está incluído no [metapacote Microsoft.AspNetCore.All](#) do ASP.NET Core.
- O pacote de provedor não está incluído no [metapacote Microsoft.AspNetCore.App](#). Para usar o provedor, instale o pacote.
- Não chame explicitamente [AddAzureWebAppDiagnostics](#). Quando o aplicativo for implantado no Serviço de Aplicativo do Azure, o provedor ficará automaticamente disponível para ele.

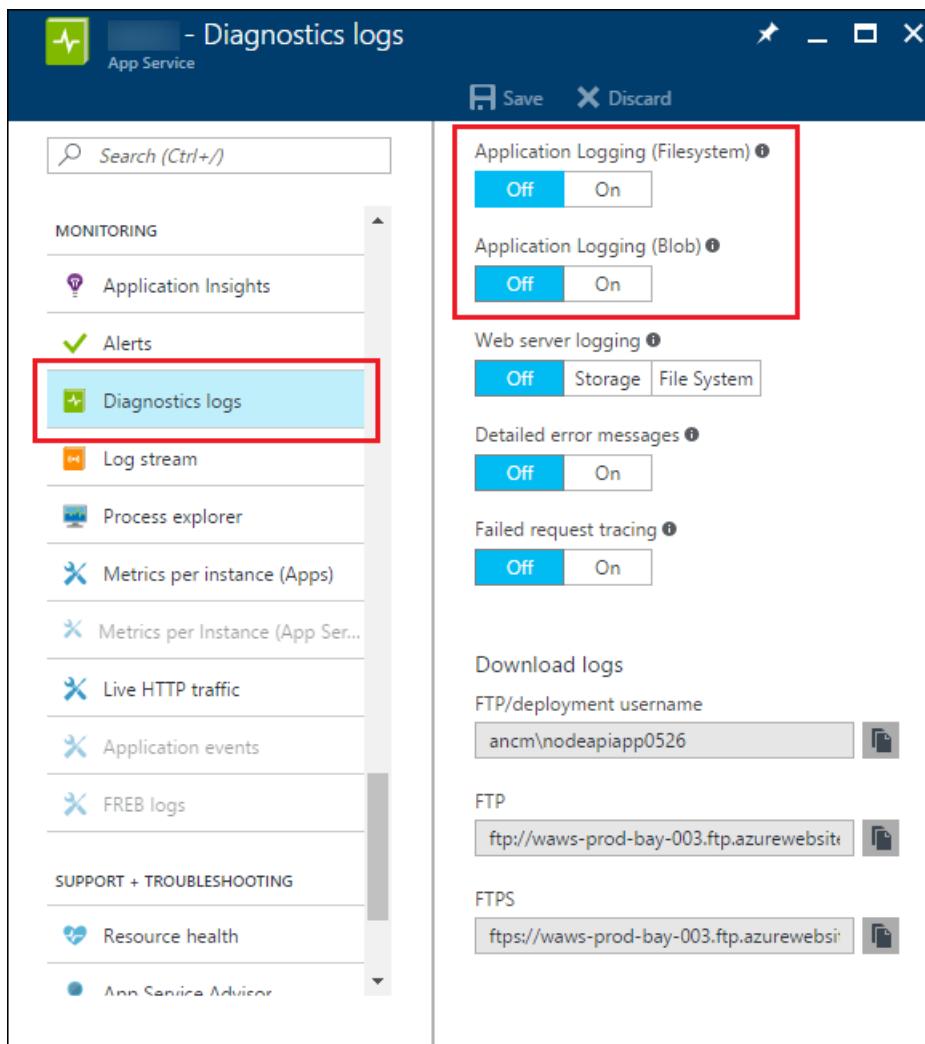
Se você estiver direcionando para o .NET Framework ou referenciando o metapacote [Microsoft.AspNetCore.App](#), adicione o pacote do provedor ao projeto. Invoque [AddAzureWebAppDiagnostics](#) em uma instância [ILoggerFactory](#):

```
logging.AddAzureWebAppDiagnostics();
```

```
loggerFactory.AddAzureWebAppDiagnostics();
```

Uma sobrecarga [AddAzureWebAppDiagnostics](#) permite passar [AzureAppServicesDiagnosticsSettings](#). O objeto settings pode substituir as configurações padrão, como o modelo de saída de registro em log, o nome do blob e o limite do tamanho do arquivo. (O *modelo Output* é um modelo de mensagem aplicado a todos os logs, além daquele fornecido com uma chamada ao método [ILogger](#)).

Ao implantar um aplicativo do Serviço de Aplicativo, o aplicativo respeita as configurações na seção [Logs de Diagnóstico](#) da página [Serviço de Aplicativo](#) do portal do Azure. Quando essas configurações são atualizadas, as alterações entram em vigor imediatamente sem a necessidade de uma reinicialização ou reimplantação do aplicativo.



O local padrão para arquivos de log é na pasta `D:\home\LogFiles\Application` e o nome de arquivo padrão é `diagnostics-aaaammdd.txt`. O limite padrão de tamanho do arquivo é 10 MB e o número padrão máximo de arquivos mantidos é 2. O nome de blob padrão é `{app-name}{timestamp}/aaaa/mm/dd/hh/{guid}-applicationLog.txt`. Para saber mais sobre o comportamento padrão, confira [AzureAppServicesDiagnosticsSettings](#).

O provedor funciona somente quando o projeto é executado no ambiente do Azure. Ele não tem nenhum efeito quando o projeto é executado localmente—ele não grava em arquivos locais ou no armazenamento de desenvolvimento local para blobs.

### Fluxo de log do Azure

O fluxo de log do Azure permite que você exiba a atividade de log em tempo real:

- O servidor de aplicativos
- Do servidor Web
- De uma solicitação de rastreio com falha

Para configurar o fluxo de log do Azure:

- Navegue até a página **Logs de Diagnóstico** da página do portal do seu aplicativo.
- Defina o **Log de aplicativo (Sistema de Arquivos)** como **Ativado**.

The screenshot shows the 'Diagnostics logs' configuration page for an Azure App Service. The left sidebar lists various monitoring and troubleshooting tools, with 'Diagnostics logs' highlighted by a red box. The main area contains settings for Application Logging (Filesystem) and Application Logging (Blob), both set to 'On'. It also includes options for Web server logging (Storage or File System), Detailed error messages (On), and Failed request tracing (Off). Log download settings are shown for FTP, FTPS, and an FTP deployment username ('azure-raffle\azure-ftp-deployment-user').

Application Logging (Filesystem) On

Level: Warning

Application Logging (Blob) On

Web server logging Storage

Detailed error messages On

Failed request tracing Off

Download logs

FTP/deployment username: azure-raffle\azure-ftp-deployment-user [Copy]

FTP: ftp://waws-prod-am2-135.ftp.azurewebsites.windows.net [Copy]

FTPS: https://waws-prod-am2-135.ftp.azurewebsites.windows.net [Copy]

Log stream

Process explorer

Resource health

App Service Advisor

New support request

Navegue até a página **Fluxo de Log** para exibir as mensagens de aplicativo. Elas são registradas pelo aplicativo por meio da interface `ILogger`.

The screenshot shows the Azure Log stream interface for an App Service. The left sidebar contains navigation links for Performance test, Resource explorer, Testing in production, Extensions, MOBILE (Easy tables, Easy APIs, Data connections), API (API definition, CORS), MONITORING (Application Insights, checked), Alerts, Diagnostics logs, Log stream (highlighted with a red box), Process explorer, SUPPORT + TROUBLESHOOTING (Resource health, App Service Advisor). The main area is titled 'Application logs' and shows log entries from November 16, 2017, at 18:39:47. The logs include details about browser user agents and Azure infrastructure.

```

2017-11-16T18:39:47 Welcome, you are now connected to log-streaming service.
2017-11-16 18:39:04 ~1/ GET /api/vfs/site/wwwroot/_=1510857327404&X-ARR-LOG-ID=36b4e37b-e31e-433d-82dd-(Windows+NT+10.0;+Win64;+x64)+AppleWebKit/537.36+(KHTML,+like+Gecko)+Chrome/62.0.3202.94+Safari/537.36 - https://we
ntent/WebsitesIndex?cacheability=3&region=eastus&cacheVersion=0&defaultCloudName=azure&extensionName=WebsitesExtens
uthority=portal.azure.com .scm.azurewebsites.net 200 0 0 966 1697 156
2017-11-16 18:39:06 GET / X-ARR-LOG-ID=784009e2-ceb-4815-968a-f63b9e8879a6 80 - 73.130.153.74 Mozilla
/537.36+(KHTML,+like+Gecko)+Chrome/62.0.3202.94+Safari/537.36 ARRAffinity=146838c347f645f6238d55649b145f0b07798967
websites.net 403 14 0 370 943 296
2017-11-16 18:39:11 ~1/ GET /api/siteextensions X-ARR-LOG-ID=7850d378-909d-4b1d-a425-9408dab70910 443 - o
zillia/5.0+(Windows+NT+10.0;+Win64;+x64)+AppleWebKit/537.36+(KHTML,+like+Gecko)+Chrome/62.0.3202.94+Safari/537.36
0 499 1190 703
2017-11-16 18:39:39 . GET /ticket.html X-ARR-LOG-ID=b7e33a93-f31e-436f-8214-672dfe954c74 80 - 73.130.153
+AppleWebKit/537.36+(KHTML,+like+Gecko)+Chrome/62.0.3202.94+Safari/537.36 ARRAffinity=146838c347f645f6238d55649b145
affle.azurewebsites.net 304 0 0 314 1087 696
2017-11-16 18:39:40 AZURE-RAFFLE GET /ticket.html X-ARR-LOG-ID=70f2946b-4139-4f65-a936-ba437411cdf9 80 - 73.130.153
+AppleWebKit/537.36+(KHTML,+like+Gecko)+Chrome/62.0.3202.94+Safari/537.36 ARRAffinity=146838c347f645f6238d55649b145
affle.azurewebsites.net 304 0 0 314 1087 218
2017-11-16 18:39:40 AZURE-RAFFLE GET /ticket.html X-ARR-LOG-ID=c5f3593f-4d4b-4811-b150-500d58341622 80 - 73.130.153
+AppleWebKit/537.36+(KHTML,+like+Gecko)+Chrome/62.0.3202.94+Safari/537.36 ARRAffinity=146838c347f645f6238d55649b145
affle.azurewebsites.net 304 0 0 314 1087 0
2017-11-16 18:39:54 AZURE-RAFFLE GET /ticket.html X-ARR-LOG-ID=ae4526f8-c8c7-4399-9f19-9c3fd8e7cdcb 80 - 73.130.153
+AppleWebKit/537.36+(KHTML,+like+Gecko)+Chrome/62.0.3202.94+Safari/537.36 ARRAffinity=146838c347f645f6238d55649b145
alle.azurewebsites.net 304 0 0 314 1087 15

```

## Log de rastreamento do Azure Application Insights

O SDK do Application Insights pode coletar e relatar logs gerados por meio da infraestrutura de log do ASP.NET Core. Para obter mais informações, consulte os seguintes recursos:

- [Visão geral do Application Insights](#)
- [Application Insights para ASP.NET Core](#)
- [Application Insights logging adapters](#) (Adaptadores de registro em log do Application Insights).

## Provedores de log de terceiros

Estruturas de log de terceiros que funcionam com o ASP.NET Core:

- [elmah.io \(repositório GitHub\)](#)
- [Gelf \(repositório do GitHub\)](#)
- [JSNLog \(repositório GitHub\)](#)
- [KissLog.net \(Repositório do GitHub\)](#)
- [Loggr \(repositório GitHub\)](#)
- [NLog \(repositório GitHub\)](#)
- [Sentry \(repositório GitHub\)](#)
- [Serilog \(repositório GitHub\)](#)
- [Stackdriver \(repositório Github\)](#)

Algumas estruturas de terceiros podem fazer o [log semântico](#), também conhecido como [registro em log estruturado](#).

Usar uma estrutura de terceiros é semelhante ao uso de um dos provedores internos:

1. Adicione um pacote NuGet ao projeto.

2. Chame um `ILoggerFactory`.

Para saber mais, consulte a documentação de cada provedor. Não há suporte para provedores de log de terceiros na Microsoft.

## Recursos adicionais

- [Registro em log de alto desempenho com o LoggerMessage no ASP.NET Core](#)

# Tratar erros no ASP.NET Core

08/01/2019 • 14 minutes to read • [Edit Online](#)

Por [Steve Smith](#) e [Tom Dykstra](#)

Este artigo apresenta abordagens comuns para o tratamento de erros em aplicativos ASP.NET Core.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## A página de exceção do desenvolvedor

Para configurar um aplicativo para exibir uma página que mostra informações detalhadas sobre exceções, use a *página de exceção do desenvolvedor*. A página é disponibilizada pelo pacote [Microsoft.AspNetCore.Diagnostics](#), que está disponível no [metapacote Microsoft.AspNetCore.App](#). Adicione uma linha ao método

`Startup.Configure` :

Para configurar um aplicativo para exibir uma página que mostra informações detalhadas sobre exceções, use a *página de exceção do desenvolvedor*. A página é disponibilizada pelo pacote [Microsoft.AspNetCore.Diagnostics](#), que está disponível no [metapacote Microsoft.AspNetCore.All](#). Adicione uma linha ao método `Startup.Configure` :

Para configurar um aplicativo para exibir uma página que mostra informações detalhadas sobre exceções, use a *página de exceção do desenvolvedor*. A página é disponibilizada adicionando uma referência de pacote para o pacote [Microsoft.AspNetCore.Diagnostics](#) no arquivo de projeto. Adicione uma linha ao método

`Startup.Configure` :

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    env.EnvironmentName = EnvironmentName.Production;

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/error");
    }
}
```

Coloque a chamada para [UseDeveloperExceptionPage](#) na frente de qualquer middleware no qual você deseja capturar exceções, tais como `app.UseMvc`.

### WARNING

Habilite a página de exceção do desenvolvedor **somente quando o aplicativo estiver em execução no ambiente de desenvolvimento**. Não é recomendável compartilhar informações de exceção detalhadas publicamente quando o aplicativo é executado em produção. [Saiba mais sobre como configurar ambientes](#).

Para ver a página de exceção do desenvolvedor, execute o aplicativo de exemplo com o ambiente definido como `Development` e adicione `?throw=true` à URL base do aplicativo. A página inclui várias guias com informações sobre a exceção e a solicitação. A primeira guia inclui um rastreamento de pilha:

Internal Server Error × +

localhost:13930/?throw=true

An unhandled exception occurred while processing the request.

Exception: Exception triggered!

```
ErrorHandlingSample.Startup+<>c+<<Configure>b_1_4>d.MoveNext() in Startup.cs, line 67
```

Stack Query Cookies Headers

Exception: Exception triggered!

```
ErrorHandlingSample.Startup+<>c+<<Configure>b_1_4>d.MoveNext() in Startup.cs
+ 67.         var builder = new StringBuilder();
System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw()
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task task)
```

```
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task task)
Microsoft.AspNetCore.Diagnostics.DeveloperExceptionPageMiddleware+<Invoke>d_7.MoveNext()
```

Show raw exception details

A próxima guia mostra os parâmetros da cadeia de caracteres de consulta, se houver:

Internal Server Error × +

localhost:13930/?throw=true

An unhandled exception occurred while processing the request.

Exception: Exception triggered!

```
ErrorHandlingSample.Startup+<>c+<<Configure>b_1_4>d.MoveNext() in Startup.cs, line 67
```

Stack Query Cookies Headers

Variable	Value
throw	true

Se a solicitação tem cookies, eles serão exibidos na guia **Cookies**. Os cabeçalhos são vistos na última guia:

An unhandled exception occurred while processing the request.

Exception: Exception triggered!

ErrorHandlingSample.Startup+<>c+<<Configure>b\_1\_4>d.MoveNext() in Startup.cs, line 67

Stack	Query	Cookies	Headers
Variable	Value		
Accept	text/html, application/xhtml+xml, image/jxr, */*		
Accept-Encoding	gzip, deflate		
Accept-Language	en-US		
Connection	Keep-Alive		
Host	localhost:13930		
MS-ASPNETCORE-TOKEN	55ddb2cf-3cf5-4734-afa9-7abcf38839f		
Referer	http://localhost:13930/		
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.79 Safari/537.36 Edge/14.14393		
X-Original-For	127.0.0.1:20566		
X-Original-Proto	http		

## Configurar uma página de tratamento de exceção personalizada

Configure uma página do manipulador de exceção para usar quando o aplicativo não estiver em execução no ambiente `Development`:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    env.EnvironmentName = EnvironmentName.Production;

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/error");
    }
}
```

Em um aplicativo Razor Pages, o modelo `dotnet new` fornece uma página de erro e uma classe de erro `PageModel`, na pasta *Páginas* do Razor Pages.

Em um aplicativo MVC, não decore o método de ação do manipulador de erro com atributos de método HTTP, como `HttpGet`. Verbos explícitos impedem algumas solicitações de chegar ao método. Permita acesso anônimo ao método para que os usuários não autenticados possam capazes receber a exibição de erro.

Por exemplo, o seguinte método de manipulador de erro é fornecido pelo modelo MVC `dotnet novo` e aparece no controlador Home:

```
[AllowAnonymous]
public IActionResult Error()
{
    return View(new ErrorViewModel
    { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
}
```

## Configurar páginas de código de status

Por padrão, o aplicativo não fornece uma página de código de status detalhada para códigos de status HTTP, como *404 Não Encontrado*. Para fornecer páginas de código de status, use o middleware das páginas de código de status.

O middleware é disponibilizado pelo pacote [Microsoft.AspNetCore.Diagnostics](#), que está disponível no metapacote [Microsoft.AspNetCore.App](#).

O middleware é disponibilizado pelo pacote [Microsoft.AspNetCore.Diagnostics](#), que está disponível no metapacote [Microsoft.AspNetCore.All](#).

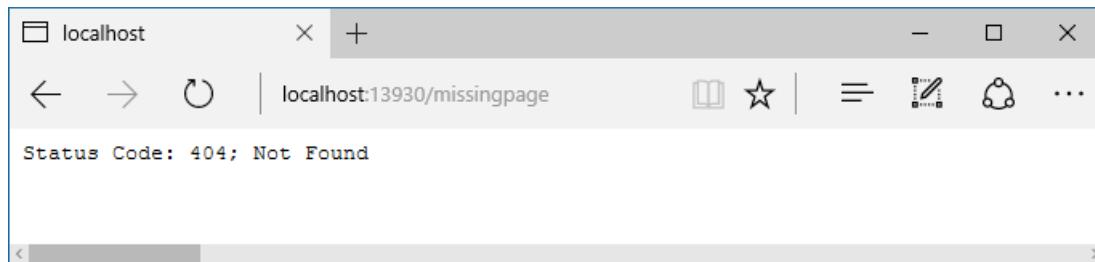
O middleware é disponibilizado adicionando uma referência de pacote para o pacote [Microsoft.AspNetCore.Diagnostics](#) no arquivo de projeto.

Adicione uma linha ao método `Startup.Configure`:

```
app.UseStatusCodePages();
```

`UseStatusCodePages` deve ser chamado antes dos middlewares de tratamento da solicitação no pipeline (por exemplo, o middleware de arquivos estáticos e o middleware MVC).

Por padrão, o middleware de páginas de código de status adiciona manipuladores, somente de texto, para os códigos de status comuns, como o 404:



O middleware permite vários métodos de extensão. Um método usa uma expressão lambda:

```
// Expose the members of the 'Microsoft.AspNetCore.Http' namespace
// at the top of the file:
// using Microsoft.AspNetCore.Http;
app.UseStatusCodePages(async context =>
{
    context.HttpContext.Response.ContentType = "text/plain";

    await context.HttpContext.Response.WriteAsync(
        "Status code page, status code: " +
        context.HttpContext.Response.StatusCode);
});
```

Uma sobrecarga de `UseStatusCodePages` usa uma cadeia de caracteres de formato e de tipo de conteúdo:

```
app.UseStatusCodePages("text/plain", "Status code page, status code: {0}");
```

## Métodos de extensão de redirecionamento e de nova execução

### [UseStatusCodePagesWithRedirects](#):

- Envia um código de status *302 – Encontrado* ao cliente.
- Redireciona o cliente para o local fornecido no modelo de URL.

O modelo pode incluir um espaço reservado de `{0}` para o código de status. O modelo deve começar com uma barra (`/`).

```
app.UseStatusCodePagesWithRedirects("/error/{0}");
```

### [UseStatusCodePagesWithReExecute](#):

- Retorna o código de status original ao cliente.
- Especifica que o corpo da resposta deve ser gerado, executando novamente o pipeline de solicitação por meio de um caminho alternativo.

O modelo pode incluir um espaço reservado de `{0}` para o código de status. O modelo deve começar com uma barra (`/`).

```
app.UseStatusCodePagesWithReExecute("/error/{0}");
```

As páginas de código de status podem ser desabilitadas para solicitações específicas em um método de manipulador de Páginas Razor ou em um controlador do MVC. Para desabilitar as páginas de código de status, tente recuperar o [IStatusCodePagesFeature](#) da coleção [HttpContext.Features](#) da solicitação e desabilitar o recurso se ele estiver disponível:

```
var statusCodePagesFeature = HttpContext.Features.Get<IStatusCodePagesFeature>();  
  
if (statusCodePagesFeature != null)  
{  
    statusCodePagesFeature.Enabled = false;  
}
```

Para usar uma sobrecarga `UseStatusCodePages*` que aponta para um ponto de extremidade dentro do aplicativo, crie um modo de exibição do MVC ou Razor Page para o ponto de extremidade. Por exemplo, o modelo [dotnet novo](#) para um aplicativo Razor Pages produz a seguinte página e classe de modelo de página:

*Error.cshtml*:

```

@page
@model ErrorModel
 @{
     ViewData["Title"] = "Error";
 }

<h1 class="text-danger">Error.</h1>
<h2 class="text-danger">An error occurred while processing your request.</h2>

@if (Model.ShowRequestId)
{
    <p>
        <strong>Request ID:</strong> <code>@Model.RequestId</code>
    </p>
}

<h3>Development Mode</h3>
<p>
    Swapping to <strong>Development</strong> environment will display more detailed
    information about the error that occurred.
</p>
<p>
    <strong>Development environment should not be enabled in deployed applications
    </strong>, as it can result in sensitive information from exceptions being
    displayed to end users. For local debugging, development environment can be
    enabled by setting the <strong>ASPNETCORE_ENVIRONMENT</strong> environment
    variable to <strong>Development</strong>, and restarting the application.
</p>

```

*Error.cshtml.cs:*

```

public class ErrorModel : PageModel
{
    public string RequestId { get; set; }

    public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);

    [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None,
      NoStore = true)]
    public void OnGet()
    {
        RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier;
    }
}

```

## Código de tratamento de exceção

Código em páginas de tratamento de exceção pode gerar exceções. Geralmente, é uma boa ideia que páginas de erro de produção sejam compostas por conteúdo puramente estático.

Além disso, esteja ciente de que, depois que os cabeçalhos de uma resposta forem enviados, você não poderá alterar o código de status da resposta e nenhuma página de exceção ou manipulador poderá ser executado. A resposta deve ser concluída ou a conexão será anulada.

## Tratamento de exceções do servidor

Além da lógica de tratamento de exceção no aplicativo, o [servidor](#) que hospeda o aplicativo executa uma parte do tratamento de exceção. Se o servidor capturar uma exceção antes que os cabeçalhos sejam enviados, o servidor enviará uma resposta *500 Erro Interno do Servidor* sem corpo. Se o servidor capturar uma exceção depois que os cabeçalhos forem enviados, o servidor fechará a conexão. As solicitações que não são

manipuladas pelo aplicativo são manipuladas pelo servidor. Qualquer exceção ocorrida é tratada pelo tratamento de exceção do servidor. As páginas de erro personalizadas, o middleware de tratamento de exceção ou os filtros configurados não afetam esse comportamento.

## Tratamento de exceção na inicialização

Apenas a camada de hospedagem pode tratar exceções que ocorrem durante a inicialização do aplicativo. Usando o [Host da Web](#), você pode [configurar como o host se comporta em resposta a erros durante a inicialização](#) usando as chaves `captureStartupErrors` e `detailedErrors`.

A hospedagem apenas poderá mostrar uma página de erro para um erro de inicialização capturado se o erro ocorrer após a associação de endereço do host/porta. Se alguma associação falhar por algum motivo, a camada de hospedagem registrará uma exceção crítica em log, o processo do dotnet falhará e nenhuma página de erro será exibida quando o aplicativo estiver sendo executado no servidor [Kestrel](#).

Quando executado no [IIS](#) ou no [IIS Express](#), um *502.5 Falha no Processo* será retornado pelo [Módulo do ASP.NET Core](#) se o processo não puder ser iniciado. Para obter informações sobre como solucionar problemas de inicialização ao hospedar com o IIS, consulte [Solucionar problemas do ASP.NET Core no IIS](#). Para obter informações sobre como solucionar problemas de inicialização com o Serviço de Aplicativo do Azure, consulte [Solucionar problemas no ASP.NET Core no Serviço de Aplicativo do Azure](#).

## Tratamento de erro do ASP.NET Core MVC

Os aplicativos [MVC](#) contêm algumas opções adicionais para o tratamento de erros, como configurar filtros de exceção e executar a validação do modelo.

### Filtros de exceção

Os filtros de exceção podem ser configurados globalmente ou por controlador ou por ação em um aplicativo MVC. Esses filtros tratam qualquer exceção sem tratamento ocorrida durante a execução de uma ação do controlador ou de outro filtro. Esses filtros não são chamados de outra forma. Para saber mais, consulte [Filtros](#).

#### TIP

Filtros de exceção são bons para interceptar exceções que ocorrem em ações do MVC, mas não são tão flexíveis quanto o middleware de tratamento de erro. Dê preferência ao uso de middleware no geral e use filtros apenas quando precisar fazer o tratamento de erro de modo *diferente*, dependendo da ação do MVC escolhida.

### Tratando erros do estado do modelo

A [validação do modelo](#) ocorre antes da invocação de cada ação do controlador e é responsabilidade do método de ação inspecionar `ModelState.IsValid` e responder de forma adequada.

Alguns aplicativos optam por seguir uma convenção padrão para lidar com erros de validação do modelo, caso em que um [filtro](#) pode ser um local adequado para implementar uma política como essa. É necessário testar o comportamento das ações com estados de modelo inválidos. Saiba mais em [Testar a lógica do controlador](#).

## Recursos adicionais

- [Referência de erros comuns para o Serviço de Aplicativo do Azure e o IIS com o ASP.NET Core](#)
- [Solucionar problemas do ASP.NET Core no IIS](#)
- [Solucionar problemas no ASP.NET Core no Serviço de Aplicativo do Azure](#)

# Middleware do ASP.NET Core

17/01/2019 • 20 minutes to read • [Edit Online](#)

Por [Rick Anderson](#) e [Steve Smith](#)

O middleware é um software montado em um pipeline de aplicativo para manipular solicitações e respostas. Cada componente:

- Escolhe se deseja passar a solicitação para o próximo componente no pipeline.
- Pode executar o trabalho antes e depois do próximo componente no pipeline.

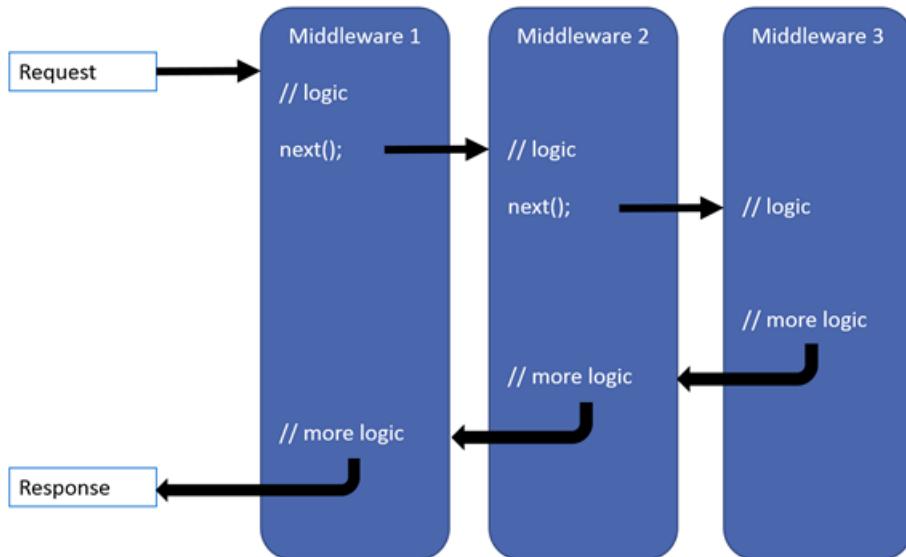
Os delegados de solicitação são usados para criar o pipeline de solicitação. Os delegados de solicitação manipulam cada solicitação HTTP.

Os delegados de solicitação são configurados usando os métodos de extensão [Run](#), [Map](#) e [Use](#). Um delegado de solicitação individual pode ser especificado em linha como um método anônimo (chamado de middleware em linha) ou pode ser definido em uma classe reutilizável. Essas classes reutilizáveis e os métodos anônimos em linha são o *middleware*, também chamado de *componentes do middleware*. Cada componente de middleware no pipeline de solicitação é responsável por invocar o próximo componente no pipeline ou causar um curto-circuito do pipeline.

[Migrar módulos e manipuladores HTTP para middleware do ASP.NET Core](#) explica a diferença entre pipelines de solicitação no ASP.NET Core e no ASP.NET 4.x e fornece mais exemplos do middleware.

## Criar um pipeline do middleware com o `IApplicationBuilder`

O pipeline de solicitação do ASP.NET Core consiste em uma sequência de delegados de solicitação, chamados um após o outro. O diagrama a seguir demonstra o conceito. O thread de execução segue as setas pretas.



Cada delegado pode executar operações antes e depois do próximo delegado. Um delegado também pode optar por não transmitir uma solicitação ao próximo delegado, o que também é chamado de *causar um curto-circuito do pipeline de solicitação*. O curto-circuito geralmente é desejável porque ele evita trabalho desnecessário. Por exemplo, o Middleware de Arquivo Estático pode retornar uma solicitação para um arquivo estático e ligar o restante do pipeline em curto-circuito. Os delegados de

tratamento de exceção são chamados no início do pipeline para que possam detectar exceções que ocorrem em etapas posteriores do pipeline.

O aplicativo ASP.NET Core mais simples possível define um delegado de solicitação única que controla todas as solicitações. Este caso não inclui um pipeline de solicitação real. Em vez disso, uma única função anônima é chamada em resposta a cada solicitação HTTP.

```
public class Startup
{
    public void Configure(IApplicationBuilder app)
    {
        app.Run(async context =>
        {
            await context.Response.WriteAsync("Hello, World!");
        });
    }
}
```

O primeiro delegado `Run` encerra o pipeline.

Encadeie vários delegados de solicitação junto com o `Use`. O parâmetro `next` representa o próximo delegado no pipeline. Lembre-se de que você pode causar um curto-circuito no pipeline ao *não* chamar o parâmetro `next`. Normalmente, você pode executar ações antes e depois do próximo delegado, conforme o exemplo a seguir demonstra:

```
public class Startup
{
    public void Configure(IApplicationBuilder app)
    {
        app.Use(async (context, next) =>
        {
            // Do work that doesn't write to the Response.
            await next.Invoke();
            // Do logging or other work that doesn't write to the Response.
        });

        app.Run(async context =>
        {
            await context.Response.WriteAsync("Hello from 2nd delegate.");
        });
    }
}
```

#### WARNING

Não chame `next.Invoke` depois que a resposta tiver sido enviada ao cliente. Altera para `HttpResponse` depois de a resposta ser iniciada e lança uma exceção. Por exemplo, mudanças como a configuração de cabeçalhos e o código de status lançam uma exceção. Gravar no corpo da resposta após a chamada `next`:

- Pode causar uma violação do protocolo. Por exemplo, gravar mais do que o `Content-Length` indicado.
- Pode corromper o formato do corpo. Por exemplo, gravar um rodapé HTML em um arquivo CSS.

`HasStarted` é uma dica útil para indicar se os cabeçalhos foram enviados ou o corpo foi gravado.

## Pedido

A ordem em que os componentes do middleware são adicionados ao método `Startup.Configure` define a ordem em que os componentes de middleware são invocados nas solicitações e a ordem

inversa para a resposta. A ordem é crítica para a segurança, o desempenho e a funcionalidade.

O método `Startup.Configure` a seguir adiciona componentes de middleware para cenários de aplicativo comuns:

1. Exceção/tratamento de erro
2. Protocolo HTTP Strict Transport Security
3. Redirecionamento para HTTPS
4. Servidor de arquivos estático
5. Imposição de política de cookies
6. Autenticação
7. Session
8. MVC

```
public void Configure(IApplicationBuilder app)
{
    if (env.IsDevelopment())
    {
        // When the app runs in the Development environment:
        // Use the Developer Exception Page to report app runtime errors.
        // Use the Database Error Page to report database runtime errors.
        app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
    }
    else
    {
        // When the app doesn't run in the Development environment:
        // Enable the Exception Handler Middleware to catch exceptions
        // thrown in the following middlewares.
        // Use the HTTP Strict Transport Security Protocol (HSTS)
        // Middleware.
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }

    // Use HTTPS Redirection Middleware to redirect HTTP requests to HTTPS.
    app.UseHttpsRedirection();

    // Return static files and end the pipeline.
    app.UseStaticFiles();

    // Use Cookie Policy Middleware to conform to EU General Data
    // Protection Regulation (GDPR) regulations.
    app.UseCookiePolicy();

    // Authenticate before the user accesses secure resources.
    app.UseAuthentication();

    // If the app uses session state, call Session Middleware after Cookie
    // Policy Middleware and before MVC Middleware.
    app.UseSession();

    // Add MVC to the request pipeline.
    app.UseMvc();
}
```

1. Exceção/tratamento de erro
2. Arquivos estáticos
3. Autenticação
4. Session
5. MVC

```

public void Configure(IApplicationBuilder app)
{
    // Enable the Exception Handler Middleware to catch exceptions
    // thrown in the following middlewares.
    app.UseExceptionHandler("/Home/Error");

    // Return static files and end the pipeline.
    app.UseStaticFiles();

    // Authenticate before you access secure resources.
    app.UseIdentity();

    // If the app uses session state, call UseSession before
    // MVC Middleware.
    app.UseSession();

    // Add MVC to the request pipeline.
    app.UseMvcWithDefaultRoute();
}

```

No código de exemplo anterior, cada método de extensão de middleware é exposto em [IApplicationBuilder](#) por meio do namespace [Microsoft.AspNetCore.Builder](#).

[UseExceptionHandler](#) é o primeiro componente de middleware adicionado ao pipeline. Portanto, o middleware de manipulador de exceção captura todas as exceções que ocorrem em chamadas posteriores.

O Middleware de Arquivo Estático é chamado no início do pipeline para que possa controlar as solicitações e causar o curto-circuito sem passar pelos componentes restantes. O Middleware de Arquivo Estático não fornece **nenhuma** verificação de autorização. Todos os arquivos atendidos, incluindo aqueles em `wwwroot`, estão disponíveis publicamente. Para conhecer uma abordagem para proteger arquivos estáticos, veja [Arquivos estáticos no ASP.NET Core](#).

Se a solicitação não for controlada pelo Middleware de Arquivo Estático, ela será transmitida para o Middleware de Autenticação ([UseAuthentication](#)), que executa a autenticação. A autenticação causa curto-circuito em solicitações não autenticadas. Embora o middleware de autenticação autentique as solicitações, a autorização (e a rejeição) ocorre somente depois que o MVC seleciona uma Página Razor específica ou um controlador MVC e uma ação.

Se a solicitação não for controlada pelo Middleware de Arquivo Estático, ela será transmitida para o Middleware de Identidade ([UseIdentity](#)), que executará a autenticação. A identidade não liga as solicitações não autenticadas em curto-circuito. Embora a identidade autentique as solicitações, a autorização (e a rejeição) ocorre somente depois que o MVC seleciona um controlador específico e uma ação.

O exemplo a seguir demonstra uma solicitação de middleware cujas solicitações de arquivos estáticos são manipuladas pelo Middleware de Arquivo Estático antes do Middleware de Compactação de Resposta. Arquivos estáticos não são compactados com este pedido de middleware. As respostas do MVC de [UseMvcWithDefaultRoute](#) podem ser compactadas.

```

public void Configure(IApplicationBuilder app)
{
    // Static files not compressed by Static File Middleware.
    app.UseStaticFiles();
    app.UseResponseCompression();
    app.UseMvcWithDefaultRoute();
}

```

## Use, Run e Map

Configure o pipeline de HTTP usando `Use`, `Run` e `Map`. O método `Use` pode ligar o pipeline em curto-circuito (ou seja, se ele não chamar um delegado de solicitação `next`). `Run` é uma convenção e alguns componentes de middleware podem expor os métodos `Run[Middleware]` que são executados no final do pipeline.

As extensões `Map` são usadas como uma convenção de ramificação do pipeline. `Map*` ramifica o pipeline de solicitação com base na correspondência do caminho da solicitação em questão. Se o caminho da solicitação iniciar com o caminho especificado, o branch será executado.

```
public class Startup
{
    private static void HandleMapTest1(IApplicationBuilder app)
    {
        app.Run(async context =>
        {
            await context.Response.WriteAsync("Map Test 1");
        });
    }

    private static void HandleMapTest2(IApplicationBuilder app)
    {
        app.Run(async context =>
        {
            await context.Response.WriteAsync("Map Test 2");
        });
    }

    public void Configure(IApplicationBuilder app)
    {
        app.Map("/map1", HandleMapTest1);

        app.Map("/map2", HandleMapTest2);

        app.Run(async context =>
        {
            await context.Response.WriteAsync("Hello from non-Map delegate. <p>");
        });
    }
}
```

A tabela a seguir mostra as solicitações e as respostas de `http://localhost:1234` usando o código anterior.

SOLICITAÇÃO	RESPOSTA
localhost:1234	Saudação do delegado diferente de Map.
localhost:1234/map1	Teste de Map 1
localhost:1234/map2	Teste de Map 2
localhost:1234/map3	Saudação do delegado diferente de Map.

Quando `Map` é usado, os segmentos de caminho correspondentes são removidos do `HttpRequest.Path` e anexados em `HttpRequest.PathBase` para cada solicitação.

`MapWhen` ramifica o pipeline de solicitação com base no resultado do predicado em questão. Qualquer predicado do tipo `Func<HttpContext, bool>` pode ser usado para mapear as solicitações para um novo branch do pipeline. No exemplo a seguir, um predicado é usado para detectar a presença de uma

variável de cadeia de caracteres de consulta `branch`:

```
public class Startup
{
    private static void HandleBranch(IApplicationBuilder app)
    {
        app.Run(async context =>
        {
            var branchVer = context.Request.Query["branch"];
            await context.Response.WriteAsync($"Branch used = {branchVer}");
        });
    }

    public void Configure(IApplicationBuilder app)
    {
        app.MapWhen(context => context.Request.Query.ContainsKey("branch"),
                    HandleBranch);

        app.Run(async context =>
        {
            await context.Response.WriteAsync("Hello from non-Map delegate. <p>");
        });
    }
}
```

A tabela a seguir mostra as solicitações e as respostas de `http://localhost:1234` usando o código anterior.

SOLICITAÇÃO	RESPOSTA
localhost:1234	Saudação do delegado diferente de Map.
localhost:1234/?branch=master	Branch usado = mestre

`Map` é compatível com aninhamento, por exemplo:

```
app.Map("/level1", level1App => {
    level1App.Map("/level2a", level2AApp => {
        // "/level1/level2a" processing
    });
    level1App.Map("/level2b", level2BApp => {
        // "/level1/level2b" processing
    });
});
```

`Map` também pode ser correspondido com vários segmentos de uma vez:

```

public class Startup
{
    private static void HandleMultiSeg(IApplicationBuilder app)
    {
        app.Run(async context =>
        {
            await context.Response.WriteAsync("Map multiple segments.");
        });
    }

    public void Configure(IApplicationBuilder app)
    {
        app.Map("/map1/seg1", HandleMultiSeg);

        app.Run(async context =>
        {
            await context.Response.WriteAsync("Hello from non-Map delegate.");
        });
    }
}

```

## Middleware interno

O ASP.NET Core é fornecido com os seguintes componentes de middleware. A coluna *Ordem* fornece observações sobre o posicionamento do middleware no pipeline de solicitação e sob quais condições o middleware podem encerrar a solicitação e impedir que outro middleware processe uma solicitação.

MIDDLEWARE	DESCRIÇÃO	PEDIDO
<a href="#">Autenticação</a>	Fornece suporte à autenticação.	Antes de <code>HttpContext.User</code> ser necessário. Terminal para retornos de chamada OAuth.
<a href="#">Política de cookies</a>	Acompanha o consentimento dos usuários para o armazenamento de informações pessoais e impõe padrões mínimos para campos de cookie, tais como <code>secure</code> e <code>SameSite</code> .	Antes do middleware que emite cookies. Exemplos: autenticação, sessão e MVC (TempData).
<a href="#">CORS</a>	Configura o Compartilhamento de Recursos entre Origens.	Antes de componentes que usam o CORS.
<a href="#">Diagnóstico</a>	Configura o diagnóstico.	Antes dos componentes que geram erros.
<a href="#">Cabeçalhos encaminhados</a>	Encaminha cabeçalhos como proxy para a solicitação atual.	Antes dos componentes que consomem os campos atualizados. Exemplos: esquema, host, IP do cliente e método.
<a href="#">Verificações de integridade</a>	Verifica a integridade de um aplicativo ASP.NET Core e suas dependências, como a verificação da disponibilidade do banco de dados.	Terminal, se uma solicitação corresponde a um ponto de extremidade da verificação de integridade.

MIDDLEWARE	DESCRIÇÃO	PEDIDO
<a href="#">Substituição do Método HTTP</a>	Permite que uma solicitação de entrada POST substitua o método.	Antes dos componentes que consomem o método atualizado.
<a href="#">Redirecionamento de HTTPS</a>	Redirecione todas as solicitações HTTP para HTTPS (ASP.NET Core 2.1 ou posterior).	Antes dos componentes que consomem a URL.
<a href="#">Segurança de Transporte Estrita de HTTP (HSTS)</a>	Middleware de aprimoramento de segurança que adiciona um cabeçalho de resposta especial (ASP.NET Core 2.1 ou posterior).	Antes das respostas serem enviadas e depois dos componentes que modificam solicitações. Exemplos: cabeçalhos encaminhados, regravação de URL.
<a href="#">MVC</a>	Processa as solicitações de Razor Pages/MVC (ASP.NET Core 2.0 ou posterior).	Terminal, se uma solicitação corresponder a uma rota.
<a href="#">OWIN</a>	Interoperabilidade com aplicativos baseados em OWIN, em servidores e em middleware.	Terminal, se o middleware OWIN processa totalmente a solicitação.
<a href="#">Cache de resposta</a>	Fornece suporte para as respostas em cache.	Antes dos componentes que exigem armazenamento em cache.
<a href="#">Compactação de resposta</a>	Fornece suporte para a compactação de respostas.	Antes dos componentes que exigem compactação.
<a href="#">Localização de Solicitação</a>	Fornece suporte à localização.	Antes dos componentes de localização importantes.
<a href="#">Roteamento</a>	Define e restringe as rotas de solicitação.	Terminal de rotas correspondentes.
<a href="#">Sessão</a>	Fornece suporte para gerenciar sessões de usuário.	Antes de componentes que exigem a sessão.
<a href="#">Arquivos estáticos</a>	Fornece suporte para servir arquivos estáticos e pesquisa no diretório.	Terminal, se uma solicitação corresponde a um arquivo.
<a href="#">Regravação de URL</a>	Fornece suporte para regravar URLs e redirecionar solicitações.	Antes dos componentes que consomem a URL.
<a href="#">WebSockets</a>	Habilita o protocolo WebSockets.	Antes dos componentes que são necessários para aceitar solicitações de WebSocket.

## Gravar middleware

O middleware geralmente é encapsulado em uma classe e exposto com um método de extensão.

Considere o middleware a seguir, que define a cultura para a solicitação atual de uma cadeia de caracteres de consulta:

```
public class Startup
{
    public void Configure(IApplicationBuilder app)
    {
        app.Use((context, next) =>
        {
            var cultureQuery = context.Request.Query["culture"];
            if (!string.IsNullOrWhiteSpace(cultureQuery))
            {
                var culture = new CultureInfo(cultureQuery);

                CultureInfo.CurrentCulture = culture;
                CultureInfo.CurrentUICulture = culture;
            }

            // Call the next delegate/middleware in the pipeline
            return next();
        });

        app.Run(async (context) =>
        {
            await context.Response.WriteAsync(
                $"Hello {CultureInfo.CurrentCulture.DisplayName}");
        });
    }
}
```

O código de exemplo anterior é usado para demonstrar a criação de um componente de middleware. Para suporte de localização interna do ASP.NET Core, veja [Globalização e localização no ASP.NET Core](#).

Você pode testar o middleware ao transmitir a cultura, por exemplo `http://localhost:7997/?culture=no`.

O código a seguir move o delegado de middleware para uma classe:

```

using Microsoft.AspNetCore.Http;
using System.Globalization;
using System.Threading.Tasks;

namespace Culture
{
    public class RequestCultureMiddleware
    {
        private readonly RequestDelegate _next;

        public RequestCultureMiddleware(RequestDelegate next)
        {
            _next = next;
        }

        public async Task InvokeAsync(HttpContext context)
        {
            var cultureQuery = context.Request.Query["culture"];
            if (!string.IsNullOrWhiteSpace(cultureQuery))
            {
                var culture = new CultureInfo(cultureQuery);

                CultureInfo.CurrentCulture = culture;
                CultureInfo.CurrentUICulture = culture;
            }

            // Call the next delegate/middleware in the pipeline
            await _next(context);
        }
    }
}

```

O nome do método `Task` de middleware precisa ser `Invoke`. No ASP.NET Core 2.0 ou posterior, o nome pode ser `Invoke` OU `InvokeAsync`.

O seguinte método de extensão expõe o middleware por meio do `IApplicationBuilder`:

```

using Microsoft.AspNetCore.Builder;

namespace Culture
{
    public static class RequestCultureMiddlewareExtensions
    {
        public static IApplicationBuilder UseRequestCulture(
            this IApplicationBuilder builder)
        {
            return builder.UseMiddleware<RequestCultureMiddleware>();
        }
    }
}

```

O código a seguir chama o middleware de `Startup.Configure`:

```

public class Startup
{
    public void Configure(IApplicationBuilder app)
    {
        app.UseRequestCulture();

        app.Run(async (context) =>
        {
            await context.Response.WriteAsync(
                $"Hello {CultureInfo.CurrentCulture.DisplayName}");
        });
    }
}

```

O middleware deve seguir o [princípio de dependências explícitas](#) ao expor suas dependências em seu construtor. O middleware é construído uma vez por *tempo de vida do aplicativo*. Consulte a seção [Dependências por solicitação](#) se você precisar compartilhar serviços com middleware dentro de uma solicitação.

Os componentes de middleware podem resolver suas dependências, utilizando a [DI \(injeção de dependência\)](#) por meio de parâmetros do construtor. `UseMiddleware<T>` também pode aceitar parâmetros adicionais diretamente.

### Dependências de pré-solicitação

Uma vez que o middleware é construído durante a inicialização do aplicativo, e não por solicitação, os serviços de tempo de vida *com escopo* usados pelos construtores do middleware não são compartilhados com outros tipos de dependência inseridos durante cada solicitação. Se você tiver que compartilhar um serviço *com escopo* entre seu serviço de middleware e serviços de outros tipos, adicione esses serviços à assinatura do método `Invoke`. O método `Invoke` pode aceitar parâmetros adicionais que são preenchidos pela injeção de dependência:

```

public class CustomMiddleware
{
    private readonly RequestDelegate _next;

    public CustomMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    // IMyScopedService is injected into Invoke
    public async Task Invoke(HttpContext httpContext, IMyScopedService svc)
    {
        svc.MyProperty = 1000;
        await _next(httpContext);
    }
}

```

## Recursos adicionais

- [Migrar módulos e manipuladores HTTP para middleware do ASP.NET Core](#)
- [Inicialização de aplicativo no ASP.NET Core](#)
- [Solicitar recursos no ASP.NET Core](#)
- [Ativação de middleware baseada em alocador no ASP.NET Core](#)
- [Ativação de middleware com um contêiner de terceiros no ASP.NET Core](#)

# Host da Web e Host Genérico no ASP.NET Core

12/12/2018 • 2 minutes to read • [Edit Online](#)

Aplicativos ASP.NET Core configuram e inicializam um *host*. O host é responsável pelo gerenciamento de tempo de vida e pela inicialização do aplicativo. Duas APIs de host estão disponíveis para uso:

- [Host da Web](#) – Adequado para a hospedagem de aplicativos Web.
- [Host Genérico](#) (ASP.NET Core 2.1 ou posteriores) – Adequado para a hospedagem de aplicativos não Web (por exemplo, aplicativos que executam tarefas em segundo plano). Em uma versão futura, o Host Genérico será adequado para hospedar qualquer tipo de aplicativo, incluindo aplicativos Web. Eventualmente, o Host Genérico substituirá o Host da Web.

Para hospedar *aplicativos Web* do ASP.NET Core, os desenvolvedores devem usar o Web Host com base em [IWebHostBuilder](#). Para hospedar *aplicativos que não sejam Web*, os desenvolvedores devem usar o Host Genérico com base em [HostBuilder](#).

## [Tarefas em segundo plano com serviços hospedados no ASP.NET Core](#)

Aprenda a implementar tarefas em segundo plano com serviços hospedados no ASP.NET Core.

## [Usar assemblies de inicialização de hospedagem no ASP.NET Core](#)

Descubra como aprimorar um aplicativo ASP.NET Core por meio de um assembly referenciado ou não referenciado usando uma implementação de [IHostingStartup](#).

# Host da Web do ASP.NET Core

10/01/2019 • 28 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

Para obter a versão 1.1 deste tópico, baixe [Host da Web do ASP.NET Core \(versão 1.1, PDF\)](#).

Aplicativos ASP.NET Core configuram e inicializam um *host*. O host é responsável pelo gerenciamento de tempo de vida e pela inicialização do aplicativo. No mínimo, o host configura um servidor e um pipeline de processamento de solicitações. Este tópico aborda o Host da Web ASP.NET Core ([IWebHostBuilder](#)), que é útil para hospedagem de aplicativos Web. Para cobertura do Host Genérico .NET ([IHostBuilder](#)), veja [Host Genérico .NET](#).

## Configurar um host

Crie um host usando uma instância do [IWebHostBuilder](#). Normalmente, isso é feito no ponto de entrada do aplicativo, o método `Main`. Em modelos de projeto, `Main` está localizado em *Program.cs*. Um *Program.cs* típico chama [CreateDefaultBuilder](#) para começar a configurar um host:

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>();
}
```

`CreateDefaultBuilder` executa as seguintes tarefas:

- Configura o servidor [Kestrel](#) como o servidor Web usando provedores de configuração de hospedagem do aplicativo. Para obter as opções padrão do servidor Kestrel, confira [Implementação do servidor Web Kestrel no ASP.NET Core](#).
- Define a raiz do conteúdo como o caminho retornado por [Directory.GetCurrentDirectory](#).
- Carrega a [configuração do host](#) de:
  - Variáveis de ambiente prefixadas com `ASPNETCORE_` (por exemplo, `ASPNETCORE_ENVIRONMENT`).
  - Argumentos de linha de comando.
- Carrega a configuração do aplicativo na seguinte ordem de:
  - `appsettings.json`.
  - `appsettings.{Environment}.json`.
  - [Gerenciador de Segredo](#) quando o aplicativo é executado no ambiente `Development` usando o assembly de entrada.
  - Variáveis de ambiente.
  - Argumentos de linha de comando.
- Configura o [registro em log](#) para a saída do console e de depuração. O registro em log inclui regras de [filtragem de log](#) especificadas em uma seção de configuração de registro em log de um arquivo `appsettings.json` ou `appsettings.{Environment}.json`.

- Quando executado com a proteção do IIS com o [Módulo do ASP.NET Core](#), `CreateDefaultBuilder` habilita a [Integração do IIS](#), que configura a porta e o endereço básico do aplicativo. A Integração do IIS também configura o aplicativo para [capturar erros de inicialização](#). Para as opções padrão do IIS, veja [Hospedar o ASP.NET Core no Windows com o IIS](#).
- Definirá `ServiceProviderOptions.ValidateScopes` como `true` se o ambiente do aplicativo for de desenvolvimento. Para obter mais informações, confira [Validação de escopo](#).

A configuração definida por `CreateDefaultBuilder` pode ser substituída e aumentada por [ConfigureAppConfiguration](#), [ConfigureLogging](#) e outros métodos, bem como os métodos de extensão de [IWebHostBuilder](#). Veja a seguir alguns exemplos:

- [ConfigureAppConfiguration](#) é usado para especificar `IConfiguration` adicionais para o aplicativo. A seguinte chamada de `ConfigureAppConfiguration` adiciona um delegado para incluir a configuração do aplicativo no arquivo `appsettings.xml`. `ConfigureAppConfiguration` pode ser chamado várias vezes. Observe que essa configuração não se aplica ao host (por exemplo, URLs de servidor ou de ambiente). Consulte a seção [Valores de configuração de Host](#).

```
WebHost.CreateDefaultBuilder(args)
    .ConfigureAppConfiguration((hostingContext, config) =>
{
    config.AddXmlFile("appsettings.xml", optional: true, reloadOnChange: true);
})
...
...
```

- A seguinte chamada de `ConfigureLogging` adiciona um delegado para configurar o nível de log mínimo ([SetMinimumLevel](#)) como [LogLevel.Warning](#). Essa configuração substitui as configurações em `appsettings.Development.json` (`LogLevel.Debug`) e `appsettings.Production.json` (`LogLevel.Error`) configuradas por `CreateDefaultBuilder`. `ConfigureLogging` pode ser chamado várias vezes.

```
WebHost.CreateDefaultBuilder(args)
    .ConfigureLogging(logging =>
{
    logging.SetMinimumLevel(LogLevel.Warning);
})
...
...
```

- A seguinte chamada para `ConfigureKestrel` substitui o padrão [Limits.MaxRequestBodySize](#) de 30 milhões de bytes, estabelecido quando o Kestrel foi configurado pelo `createDefaultBuilder`:

```
WebHost.CreateDefaultBuilder(args)
    .ConfigureKestrel((context, options) =>
{
    options.Limits.MaxRequestBodySize = 20000000;
});
```

- A seguinte chamada a `UseKestrel` substitui o padrão [Limits.MaxRequestBodySize](#) de 30.000.000 bytes estabelecido quando Kestrel foi configurado por `createDefaultBuilder`:

```
WebHost.CreateDefaultBuilder(args)
    .UseKestrel(options =>
{
    options.Limits.MaxRequestBodySize = 20000000;
});
```

A raiz do conteúdo determina onde o host procura por arquivos de conteúdo, como arquivos de exibição do MVC. Quando o aplicativo é iniciado na pasta raiz do projeto, essa pasta é usada como a raiz do conteúdo. Esse é o padrão usado no [Visual Studio](#) e nos [novos modelos dotnet](#).

Para obter mais informações sobre a configuração de aplicativo, veja [Configuração no ASP.NET Core](#).

#### NOTE

Como uma alternativa ao uso do método `CreateDefaultBuilder` estático, criar um host de `WebHostBuilder` é uma abordagem compatível com o ASP.NET Core 2.x. Para obter mais informações, consulte a guia do ASP.NET Core 1.x.

Ao configurar um host, os métodos `Configure` e `ConfigureServices` podem ser fornecidos. Se uma classe `Startup` for especificada, ela deverá definir um método `Configure`. Para obter mais informações, consulte [Inicialização de aplicativo no ASP.NET Core](#). Diversas chamadas para `ConfigureServices` são acrescentadas umas às outras. Diversas chamadas para `Configure` ou `UseStartup` no `WebHostBuilder` substituem configurações anteriores.

## Valores de configuração do host

`WebHostBuilder` conta com as seguintes abordagens para definir os valores de configuração do host:

- Configuração do construtor do host, que inclui variáveis de ambiente com o formato `ASPNETCORE_{configurationKey}`. Por exemplo, `ASPNETCORE_ENVIRONMENT`.
- Extensões como `UseContentRoot` e `UseConfiguration` (consulte a seção [Configuração de substituição](#)).
- `UseSetting` e a chave associada. Ao definir um valor com `UseSetting`, o valor é definido como uma cadeia de caracteres, independentemente do tipo.

O host usa a opção que define um valor por último. Para obter mais informações, veja [Substituir configuração](#) na próxima seção.

### Chave do Aplicativo (Nome)

A propriedade `IHostingEnvironment.ApplicationName` é definida automaticamente quando `UseStartup` ou `Configure` é chamado durante a construção do host. O valor é definido para o nome do assembly que contém o ponto de entrada do aplicativo. Para definir o valor explicitamente, use o `WebHostDefaults.ApplicationKey`:

**Chave:** `applicationName`

**Tipo:** `string`

**Padrão:** o nome do assembly que contém o ponto de entrada do aplicativo.

**Definido usando:** `UseSetting`

**Variável de ambiente:** `ASPNETCORE_APPLICATIONNAME`

```
WebHost.CreateDefaultBuilder(args)
    .UseSetting(WebHostDefaults.ApplicationKey, "CustomApplicationName")
```

### Capturar erros de inicialização

Esta configuração controla a captura de erros de inicialização.

**Chave:** `captureStartupErrors`

**Tipo:** `bool` (`true` ou `1`)

**Padrão:** o padrão é `false`, a menos que o aplicativo seja executado com o Kestrel por trás do IIS, em que o padrão é `true`.

**Definido usando:** `CaptureStartupErrors`

**Variável de ambiente:** `ASPNETCORE_CAPTURESTARTUPERRORS`

Quando `false`, erros durante a inicialização resultam no encerramento do host. Quando `true`, o host captura exceções durante a inicialização e tenta iniciar o servidor.

```
WebHost.CreateDefaultBuilder(args)
    .CaptureStartupErrors(true)
```

## Raiz do conteúdo

Essa configuração determina onde o ASP.NET Core começa a procurar por arquivos de conteúdo, como exibições do MVC.

**Chave:** contentRoot

**Tipo:** `string`

**Padrão:** o padrão é a pasta em que o assembly do aplicativo reside.

**Definido usando:** `UseContentRoot`

**Variável de ambiente:** `ASPNETCORE_CONTENTROOT`

A raiz do conteúdo também é usada como o caminho base para a [Configuração da raiz da Web](#). Se o caminho não existir, o host não será iniciado.

```
WebHost.CreateDefaultBuilder(args)
    .UseContentRoot("c:\\<content-root>")
```

## Erros detalhados

Determina se erros detalhados devem ser capturados.

**Chave:** detailedErrors

**Tipo:** `bool` (`true` ou `1`)

**Padrão:** falso

**Definido usando:** `UseSetting`

**Variável de ambiente:** `ASPNETCORE_DETAILEDERRORS`

Quando habilitado (ou quando o [Ambiente](#) é definido como `Development`), o aplicativo captura exceções detalhadas.

```
WebHost.CreateDefaultBuilder(args)
    .UseSetting(WebHostDefaults.DetailedErrorsKey, "true")
```

## Ambiente

Define o ambiente do aplicativo.

**Chave:** ambiente

**Tipo:** `string`

**Padrão:** Produção

**Definido usando:** `UseEnvironment`

**Variável de ambiente:** `ASPNETCORE_ENVIRONMENT`

O ambiente pode ser definido como qualquer valor. Os valores definidos pela estrutura incluem `Development`, `Staging` e `Production`. Os valores não diferenciam maiúsculas de minúsculas. Por padrão, o [Ambiente](#) é lido da variável de ambiente `ASPNETCORE_ENVIRONMENT`. Ao usar o [Visual Studio](#), variáveis de ambiente podem ser definidas no arquivo `launchSettings.json`. Para obter mais informações, consulte [Usar vários ambientes no ASP.NET Core](#).

```
WebHost.CreateDefaultBuilder(args)
    .UseEnvironment(EnvironmentName.Development)
```

## Hospedando assemblies de inicialização

Define os assemblies de inicialização de hospedagem do aplicativo.

**Chave:** hostingStartupAssemblies

**Tipo:** string

**Padrão:** Cadeia de caracteres vazia

**Definido usando:** UseSetting

**Variável de ambiente:** ASPNETCORE\_HOSTINGSTARTUPASSEMBLIES

Uma cadeia de caracteres delimitada por ponto e vírgula de assemblies de inicialização de hospedagem para carregamento na inicialização.

Embora o valor padrão da configuração seja uma cadeia de caracteres vazia, os assemblies de inicialização de hospedagem sempre incluem o assembly do aplicativo. Quando assemblies de inicialização de hospedagem são fornecidos, eles são adicionados ao assembly do aplicativo para carregamento quando o aplicativo compilar seus serviços comuns durante a inicialização.

```
WebHost.CreateDefaultBuilder(args)
    .UseSetting(WebHostDefaults.HostingStartupAssembliesKey, "assembly1;assembly2")
```

## Porta HTTPS

Defina a porta de redirecionamento HTTPS. Uso em [aplicação de HTTPS](#).

**Chave:** https\_port **Tipo:** cadeia de caracteres **Padrão:** um valor padrão não está definido. **Definir usando:**

UseSetting **Variável de ambiente:** ASPNETCORE\_HTTPS\_PORT

```
WebHost.CreateDefaultBuilder(args)
    .UseSetting("https_port", "8080")
```

## Hospedando assemblies de exclusão de inicialização

Uma cadeia de caracteres delimitada por ponto e vírgula de assemblies de inicialização de hospedagem para exclusão na inicialização.

**Chave:** hostingStartupExcludeAssemblies

**Tipo:** string

**Padrão:** Cadeia de caracteres vazia

**Definido usando:** UseSetting

**Variável de ambiente:** ASPNETCORE\_HOSTINGSTARTUPEXCLUDEASSEMBLIES

```
WebHost.CreateDefaultBuilder(args)
    .UseSetting(WebHostDefaults.HostingStartupExcludeAssembliesKey, "assembly1;assembly2")
```

## Preferir URLs de hospedagem

Indica se o host deve escutar as URLs configuradas com o `WebHostBuilder` em vez daquelas configuradas com a implementação `IIServer`.

**Chave:** preferHostingUrls

**Tipo:** bool ( true ou 1 )

**Padrão:** true

**Definido usando:** `PreferHostingUrls`

**Variável de ambiente:** `ASPNETCORE_PREFERHOSTINGURLS`

```
WebHost.CreateDefaultBuilder(args)
    .PreferHostingUrls(false)
```

## Impedir inicialização de hospedagem

Impede o carregamento automático de assemblies de inicialização de hospedagem, incluindo assemblies de inicialização de hospedagem configurados pelo assembly do aplicativo. Para obter mais informações, consulte [Usar assemblies de inicialização de hospedagem no ASP.NET Core](#).

**Chave:** `preventHostingStartup`

**Tipo:** `bool` (`true` ou `1`)

**Padrão:** falso

**Definido usando:** `UseSetting`

**Variável de ambiente:** `ASPNETCORE_PREVENTHOSTINGSTARTUP`

```
WebHost.CreateDefaultBuilder(args)
    .UseSetting(WebHostDefaults.PreventHostingStartupKey, "true")
```

## URLs de servidor

Indica os endereços IP ou endereços de host com portas e protocolos que o servidor deve escutar para solicitações.

**Chave:** `urls`

**Tipo:** `string`

**Padrão:** `http://localhost:5000`

**Definido usando:** `UseUrls`

**Variável de ambiente:** `ASPNETCORE_URLS`

Defina como uma lista separada por ponto e vírgula (;) de prefixos de URL aos quais o servidor deve responder. Por exemplo, `http://localhost:123`. Use `"*"` para indicar que o servidor deve escutar solicitações em qualquer endereço IP ou nome do host usando a porta e o protocolo especificados (por exemplo, `http://*:5000`). O protocolo (`http://` ou `https://`) deve ser incluído com cada URL. Os formatos compatíveis variam dependendo dos servidores.

```
WebHost.CreateDefaultBuilder(args)
    .UseUrls("http://*:5000;http://localhost:5001;https://hostname:5002")
```

O Kestrel tem sua própria API de configuração de ponto de extremidade. Para obter mais informações, consulte [Implementação do servidor Web Kestrel no ASP.NET Core](#).

## Tempo limite de desligamento

Especifica o tempo de espera para o desligamento do host da Web.

**Chave:** `shutdownTimeoutSeconds`

**Tipo:** `int`

**Padrão:** 5

**Definido usando:** `UseShutdownTimeout`

**Variável de ambiente:** `ASPNETCORE_SHUTDOWNTIMEOUTSECONDS`

Embora a chave aceite um `int` com `UseSetting` (por exemplo,

```
.UseSetting(WebHostDefaults.ShutdownTimeoutKey, "10"))
```

), o método de extensão `UseShutdownTimeout` usa um

## TimeSpan

Durante o período de tempo limite, a hospedagem:

- Dispara [IApplicationLifetime.ApplicationStopping](#).
- Tenta parar os serviços hospedados, registrando em log os erros dos serviços que falham ao parar.

Se o período de tempo limite expirar antes que todos os serviços hospedados parem, os serviços ativos restantes serão parados quando o aplicativo for desligado. Os serviços serão parados mesmo se ainda não tiverem concluído o processamento. Se os serviços exigirem mais tempo para parar, aumente o tempo limite.

```
WebHost.CreateDefaultBuilder(args)
    .UseShutdownTimeout(TimeSpan.FromSeconds(10))
```

## Assembly de inicialização

Determina o assembly para pesquisar pela classe `Startup`.

**Chave:** `startupAssembly`

**Tipo:** `string`

**Padrão:** o assembly do aplicativo

**Definido usando:** `UseStartup`

**Variável de ambiente:** `ASPNETCORE_STARTUPASSEMBLY`

O assembly por nome (`string`) ou por tipo (`TStartup`) pode ser referenciado. Se vários métodos `UseStartup` forem chamados, o último terá precedência.

```
WebHost.CreateDefaultBuilder(args)
    .UseStartup("StartupAssemblyName")
```

```
WebHost.CreateDefaultBuilder(args)
    .UseStartup<TStartup>()
```

## Raiz da Web

Define o caminho relativo para os ativos estáticos do aplicativo.

**Chave:** `webroot`

**Tipo:** `string`

**Padrão:** se não for especificado, o padrão será "(Raiz do conteúdo)/wwwroot", se o caminho existir. Se o caminho não existir, um provedor de arquivo não operacional será usado.

**Definido usando:** `UseWebRoot`

**Variável de ambiente:** `ASPNETCORE_WEBROOT`

```
WebHost.CreateDefaultBuilder(args)
    .UseWebRoot("public")
```

## Substituir configuração

Use [Configuração](#) para configurar o host Web. No exemplo a seguir, a configuração do host é especificada, opcionalmente, em um arquivo `hostsettings.json`. Qualquer configuração carregada do arquivo `hostsettings.json` pode ser substituída por argumentos de linha de comando. A configuração de build (no `config`) é usada para configurar o host com [UseConfiguration](#). A configuração `IWebHostBuilder` é adicionada à configuração do aplicativo, mas o contrário não é verdade—`ConfigureAppConfiguration` não afeta a configuração

```
IWebHostBuilder .
```

Substituição da configuração fornecida por `UseUrls` pela configuração de `hostsettings.json` primeiro, e pela configuração de argumento da linha de comando depois:

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args)
    {
        var config = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("hostsettings.json", optional: true)
            .AddCommandLine(args)
            .Build();

        return WebHost.CreateDefaultBuilder(args)
            .UseUrls("http://*:5000")
            .UseConfiguration(config)
            .Configure(app =>
        {
            app.Run(context =>
                context.Response.WriteAsync("Hello, World!"));
        });
    }
}
```

`hostsettings.json`:

```
{
    urls: "http://*:5005"
}
```

#### NOTE

Atualmente, o método de extensão `UseConfiguration` não é capaz de analisar uma seção de configuração retornada por `GetSection` (por exemplo, `.UseConfiguration(Configuration.GetSection("section"))`). O método `GetSection` filtra as chaves de configuração da seção solicitada, mas deixa o nome da seção nas chaves (por exemplo, `section:urls`, `section:environment`). O método `UseConfiguration` espera que as chaves correspondam às chaves `WebHostBuilder` (por exemplo, `urls`, `environment`). A presença do nome da seção nas chaves impede que os valores da seção configurem o host. Esse problema será corrigido em uma próxima versão. Para obter mais informações e soluções alternativas, consulte [Passar a seção de configuração para `WebHostBuilder.UseConfiguration` usa chaves completas](#).

`UseConfiguration` somente copia as chaves do `IConfiguration` fornecido para a configuração do construtor de host. Portanto, definir `reloadOnChange: true` para arquivos de configuração JSON,INI e XML não tem nenhum efeito.

Para especificar o host executado em uma URL específica, o valor desejado pode ser passado em um prompt de comando ao executar `dotnet run`. O argumento de linha de comando substitui o valor `urls` do arquivo `hostsettings.json` e o servidor escuta na porta 8080:

```
dotnet run --urls "http://*:8080"
```

# Gerenciar o host

## Executar

O método `Run` inicia o aplicativo Web e bloqueia o thread de chamada até que o host seja desligado:

```
host.Run();
```

## Iniciar

Execute o host sem bloqueio, chamando seu método `Start`:

```
using (host)
{
    host.Start();
    Console.ReadLine();
}
```

Se uma lista de URLs for passada para o método `Start`, ele escutará nas URLs especificadas:

```
var urls = new List<string>()
{
    "http://*:5000",
    "http://localhost:5001"
};

var host = new WebHostBuilder()
    .UseKestrel()
    .UseStartup<Startup>()
    .Start(urls.ToArray());

using (host)
{
    Console.ReadLine();
}
```

O aplicativo pode inicializar e iniciar um novo host usando os padrões pré-configurados de `CreateDefaultBuilder`, usando um método estático conveniente. Esses métodos iniciam o servidor sem uma saída do console e com `WaitForShutdown` e aguardam uma quebra (Ctrl-C/SIGINT ou SIGTERM):

## Start(RequestDelegate app)

Inicie com um `RequestDelegate`:

```
using (var host = WebHost.Start(app => app.Response.WriteAsync("Hello, World!")))
{
    Console.WriteLine("Use Ctrl-C to shutdown the host...");
    host.WaitForShutdown();
}
```

Faça uma solicitação no navegador para `http://localhost:5000` para receber a resposta "Olá, Mundo!" `WaitForShutdown` bloqueia até que uma quebra (Ctrl-C/SIGINT ou SIGTERM) seja emitida. O aplicativo exibe a mensagem `Console.WriteLine` e aguarda um pressionamento de tecla para ser encerrado.

## Start(string url, RequestDelegate app)

Inicie com uma URL e `RequestDelegate`:

```

using (var host = WebHost.Start("http://localhost:8080", app => app.Response.WriteAsync("Hello, World!")))
{
    Console.WriteLine("Use Ctrl-C to shutdown the host...");
    host.WaitForShutdown();
}

```

Produz o mesmo resultado que **Start(RequestDelegate app)**, mas o aplicativo responde em

`http://localhost:8080`.

### Start(Action<IRouteBuilder> routeBuilder)

Use uma instância de `IRouteBuilder` ([Microsoft.AspNetCore.Routing](#)) para usar o middleware de roteamento:

```

using (var host = WebHost.Start(router => router
    .MapGet("hello/{name}", (req, res, data) =>
        res.WriteAsync($"Hello, {data.Values["name"]}!"))
    .MapGet("buenosdias/{name}", (req, res, data) =>
        res.WriteAsync($"Buenos dias, {data.Values["name"]}!"))
    .MapGet("throw/{message?}", (req, res, data) =>
        throw new Exception((string)data.Values["message"] ?? "Uh oh!"))
    .MapGet("{greeting}/{name}", (req, res, data) =>
        res.WriteAsync($"{data.Values["greeting"]}, {data.Values["name"]}!"))
    .MapGet("", (req, res, data) => res.WriteAsync("Hello, World!")))
{
    Console.WriteLine("Use Ctrl-C to shutdown the host...");
    host.WaitForShutdown();
}

```

Use as seguintes solicitações de navegador com o exemplo:

SOLICITAÇÃO	RESPOSTA
<code>http://localhost:5000/hello/Martin</code>	Hello, Martin!
<code>http://localhost:5000/buenosdias/Catrina</code>	Buenos dias, Catrina!
<code>http://localhost:5000/throw/ooops!</code>	Gera uma exceção com a cadeia de caracteres "ooops!"
<code>http://localhost:5000/throw</code>	Gera uma exceção com a cadeia de caracteres "Uh oh!"
<code>http://localhost:5000/Sante/Kevin</code>	Sante, Kevin!
<code>http://localhost:5000</code>	Olá, Mundo!

`WaitForShutdown` bloqueia até que uma quebra (Ctrl-C/SIGINT ou SIGTERM) seja emitida. O aplicativo exibe a mensagem `Console.WriteLine` e aguarda um pressionamento de tecla para ser encerrado.

### Start(string url, Action<IRouteBuilder> routeBuilder)

Use uma URL e uma instância de `IRouteBuilder`:

```

using (var host = WebHost.Start("http://localhost:8080", router => router
    .MapGet("hello/{name}", (req, res, data) =>
        res.WriteAsync($"Hello, {data.Values["name"]}!"))
    .MapGet("buenosdias/{name}", (req, res, data) =>
        res.WriteAsync($"Buenos dias, {data.Values["name"]}!"))
    .MapGet("throw/{message?}", (req, res, data) =>
        throw new Exception((string)data.Values["message"] ?? "Uh oh!"))
    .MapGet("{greeting}/{name}", (req, res, data) =>
        res.WriteAsync($"{data.Values["greeting"]}, {data.Values["name"]}!"))
    .MapGet("", (req, res, data) => res.WriteAsync("Hello, World!")))
{
    Console.WriteLine("Use Ctrl-C to shut down the host...");
    host.WaitForShutdown();
}

```

Produz o mesmo resultado que **Start(Action<IRouteBuilder> routeBuilder)**, mas o aplicativo responde em `http://localhost:8080`.

### **StartWith(Action<IApplicationBuilder> app)**

Forneça um delegado para configurar um `IApplicationBuilder`:

```

using (var host = WebHost.StartWith(app =>
    app.Use(next =>
    {
        return async context =>
        {
            await context.Response.WriteAsync("Hello World!");
        };
    }));
{
    Console.WriteLine("Use Ctrl-C to shut down the host...");
    host.WaitForShutdown();
}

```

Faça uma solicitação no navegador para `http://localhost:5000` para receber a resposta "Olá, Mundo!"  
`WaitForShutdown` bloqueia até que uma quebra (Ctrl-C/SIGINT ou SIGTERM) seja emitida. O aplicativo exibe a mensagem `Console.WriteLine` e aguarda um pressionamento de tecla para ser encerrado.

### **StartWith(string url, Action<IApplicationBuilder> app)**

Forneça um delegado e uma URL para configurar um `IApplicationBuilder`:

```

using (var host = WebHost.StartWith("http://localhost:8080", app =>
    app.Use(next =>
    {
        return async context =>
        {
            await context.Response.WriteAsync("Hello World!");
        };
    }));
{
    Console.WriteLine("Use Ctrl-C to shut down the host...");
    host.WaitForShutdown();
}

```

Produz o mesmo resultado que **StartWith(Action<IApplicationBuilder> app)**, mas o aplicativo responde em `http://localhost:8080`.

## Interface IHostingEnvironment

A interface `IHostingEnvironment` fornece informações sobre o ambiente de hospedagem na Web do aplicativo. Use a [injeção de construtor](#) para obter o `IHostingEnvironment` para usar suas propriedades e métodos de extensão:

```
public class CustomFileReader
{
    private readonly IHostingEnvironment _env;

    public CustomFileReader(IHostingEnvironment env)
    {
        _env = env;
    }

    public string ReadFile(string filePath)
    {
        var fileProvider = _env.WebRootFileProvider;
        // Process the file here
    }
}
```

Uma [abordagem baseada em convenção](#) pode ser usada para configurar o aplicativo na inicialização com base no ambiente. Como alternativa, injete o `IHostingEnvironment` no construtor `Startup` para uso em `ConfigureServices`:

```
public class Startup
{
    public Startup(IHostingEnvironment env)
    {
        HostingEnvironment = env;
    }

    public IHostingEnvironment HostingEnvironment { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        if (HostingEnvironment.IsDevelopment())
        {
            // Development configuration
        }
        else
        {
            // Staging/Production configuration
        }

        var contentRootPath = HostingEnvironment.ContentRootPath;
    }
}
```

#### NOTE

Além do método de extensão `IsDevelopment`, `IHostingEnvironment` oferece os métodos `IsStaging`, `IsProduction` e `IsEnvironment(string environmentName)`. Para obter mais informações, consulte [Usar vários ambientes no ASP.NET Core](#).

O serviço `IHostingEnvironment` também pode ser injetado diretamente no método `Configure` para configurar o pipeline de processamento:

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        // In Development, use the developer exception page
        app.UseDeveloperExceptionPage();
    }
    else
    {
        // In Staging/Production, route exceptions to /error
        app.UseExceptionHandler("/error");
    }

    var contentRootPath = env.ContentRootPath;
}

```

`IHostingEnvironment` pode ser injetado no método `Invoke` ao criar um [middleware](#) personalizado:

```

public async Task Invoke(HttpContext context, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        // Configure middleware for Development
    }
    else
    {
        // Configure middleware for Staging/Production
    }

    var contentRootPath = env.ContentRootPath;
}

```

## Interface `IApplicationLifetime`

[IApplicationLifetime](#) permite atividades pós-inicialização e desligamento. Três propriedades na interface são tokens de cancelamento usados para registrar métodos `Action` que definem eventos de inicialização e desligamento.

TOKEN DE CANCELAMENTO	ACIONADO QUANDO...
<a href="#">ApplicationStarted</a>	O host foi iniciado totalmente.
<a href="#">ApplicationStopped</a>	O host está concluindo um desligamento normal. Todas as solicitações devem ser processadas. O desligamento é bloqueado até que esse evento seja concluído.
<a href="#">ApplicationStopping</a>	O host está executando um desligamento normal. Solicitações ainda podem estar sendo processadas. O desligamento é bloqueado até que esse evento seja concluído.

```

public class Startup
{
    public void Configure(IApplicationBuilder app, IApplicationLifetime appLifetime)
    {
        appLifetime.ApplicationStarted.Register(OnStarted);
        appLifetime.ApplicationStopping.Register(OnStopping);
        appLifetime.ApplicationStopped.Register(OnStopped);

        Console.CancelKeyPress += (sender, eventArgs) =>
        {
            appLifetime.StopApplication();
            // Don't terminate the process immediately, wait for the Main thread to exit gracefully.
            eventArgs.Cancel = true;
        };
    }

    private void OnStarted()
    {
        // Perform post-startup activities here
    }

    private void OnStopping()
    {
        // Perform on-stopping activities here
    }

    private void OnStopped()
    {
        // Perform post-stopped activities here
    }
}

```

[StopApplication](#) solicita o término do aplicativo. A classe a seguir usa [StopApplication](#) para desligar normalmente um aplicativo quando o método [Shutdown](#) da classe é chamado:

```

public class MyClass
{
    private readonly IApplicationLifetime _appLifetime;

    public MyClass(IApplicationLifetime appLifetime)
    {
        _appLifetime = appLifetime;
    }

    public void Shutdown()
    {
        _appLifetime.StopApplication();
    }
}

```

## Validação de escopo

[CreateDefaultBuilder](#) define [ServiceProviderOptions.ValidateScopes](#) como [true](#) quando o ambiente do aplicativo é de desenvolvimento.

Quando [ValidateScopes](#) está definido como [true](#), o provedor de serviço padrão executa verificações para saber se:

- Os serviços com escopo não são resolvidos direta ou indiretamente pelo provedor de serviço raiz.
- Os serviços com escopo não são injetados direta ou indiretamente em singletons.

O provedor de serviços raiz é criado quando [BuildServiceProvider](#) é chamado. O tempo de vida do provedor

de serviço raiz corresponde ao tempo de vida do aplicativo/servidor quando o provedor começa com o aplicativo e é descartado quando o aplicativo é desligado.

Os serviços com escopo são descartados pelo contêiner que os criou. Se um serviço com escopo é criado no contêiner raiz, o tempo de vida do serviço é promovido efetivamente para singleton, porque ele só é descartado pelo contêiner raiz quando o aplicativo/servidor é desligado. A validação dos escopos de serviço detecta essas situações quando `BuildServiceProvider` é chamado.

Para que os escopos sempre sejam validados, incluindo no ambiente de produção, configure `ServiceProviderOptions` com `UseDefaultServiceProvider` no construtor do host:

```
WebHost.CreateDefaultBuilder(args)
    .UseDefaultServiceProvider((context, options) => {
        options.ValidateScopes = true;
    })
```

## Recursos adicionais

- [Hospedar o ASP.NET Core no Windows com o IIS](#)
- [Host ASP.NET Core no Linux com Nginx](#)
- [Hospedar o ASP.NET Core no Linux com o Apache](#)
- [Hospedar o ASP.NET Core em um serviço Windows](#)

# Host Genérico .NET

12/12/2018 • 19 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

Os aplicativos .NET Core configuram e iniciam um *host*. O host é responsável pelo gerenciamento de tempo de vida e pela inicialização do aplicativo. Este tópico aborda o Host Genérico do ASP.NET Core ([HostBuilder](#)), que é útil para a hospedagem de aplicativos que não processam solicitações HTTP. Para cobertura sobre o host da Web ([WebHostBuilder](#)), veja [Host da Web do ASP.NET Core](#).

O objetivo do Host Genérico é separar o pipeline HTTP da API de host da Web para permitir maior gama de cenários de host. Sistema de mensagens, tarefas em segundo plano e outras cargas de trabalho não HTTP com base no benefício do Host Genérico de recursos abrangentes, como configuração, DI (injeção de dependência) e log.

O Host Genérico é novo no ASP.NET Core 2.1 e não é adequado para cenários de hospedagem na Web. Para cenários de hospedagem na Web, use o [host da Web](#). O Host Genérico está em desenvolvimento para substituir o host da Web em uma versão futura e atuar como API do host principal em cenários HTTP e não HTTP.

## [Exibir ou baixar código de exemplo \(como baixar\)](#)

Ao executar o aplicativo de exemplo no [Visual Studio Code](#), use um *terminal externo ou integrado*. Não execute o exemplo em um `internalConsole`.

Para definir o console no Visual Studio Code:

1. Abra o arquivo `.vscode/launch.json`.
2. Na configuração **Inicialização do .NET Core (console)**, localize a entrada **console**. Defina o valor como `externalTerminal` ou `integratedTerminal`.

## Introdução

A biblioteca do Host Genérico está disponível no namespace [Microsoft.Extensions.Hosting](#) e é fornecida pelo pacote [Microsoft.Extensions.Hosting](#). O pacote `Microsoft.Extensions.Hosting` está incluído no [metapacote Microsoft.AspNetCore.App](#) (ASP.NET Core 2.1 ou posterior).

`IHostedService` é o ponto de entrada para a execução de código. Cada implementação do `IHostedService` é executada na ordem do [registro de serviço em ConfigureServices](#). `StartAsync` é chamado em cada `IHostedService` quando o host é iniciado e `StopAsync` é chamado na ordem inversa de registro quando o host é desligado normalmente.

## Configurar um host

`IHostBuilder` é o principal componente usado por aplicativos e bibliotecas para inicializar, compilar e executar o host:

```
public static async Task Main(string[] args)
{
    var host = new HostBuilder()
        .Build();

    await host.RunAsync();
}
```

## Opções

Os [HostOptions](#) configuram opções para o [IHost](#).

### Tempo limite de desligamento

O [ShutdownTimeout](#) define o tempo limite para [StopAsync](#). O valor padrão é cinco segundos.

A seguinte configuração de opção no [Program.Main](#) aumenta o tempo limite de desligamento padrão de cinco segundos para 20 segundos:

```
var host = new HostBuilder()
    .ConfigureServices((hostContext, services) =>
{
    services.Configure<HostOptions>(option =>
    {
        option.ShutdownTimeout = System.TimeSpan.FromSeconds(20);
    });
})
.Build();
```

## Serviços padrão

Os seguintes serviços são registrados durante a inicialização do host:

- Ambiente ([IHostingEnvironment](#))
- HostBuilderContext
- Configuração ( [IConfiguration](#))
- [IApplicationLifetime](#) ([ApplicationLifetime](#))
- [IHostLifetime](#) ([ConsoleLifetime](#))
- [IHost](#)
- Opções ([AddOptions](#))
- Registro em log ([AddLogging](#))

## Configuração do host

A configuração do host é criada:

- Chamando métodos de extensão em [IHostBuilder](#) para definir a [raiz do conteúdo](#) e o [ambiente](#).
- Lendo a configuração de provedores de configuração no [ConfigureHostConfiguration](#).

### Métodos de extensão

#### Chave do aplicativo (nome)

A propriedade [IHostingEnvironment.ApplicationName](#) é definida na configuração do host durante a construção do host. Para definir o valor explicitamente, use o [HostDefaults.ApplicationKey](#):

**Chave:** applicationName

**Tipo:** string

**Padrão:** o nome do assembly que contém o ponto de entrada do aplicativo.

**Definido usando:** HostBuilderContext.HostingEnvironment.ApplicationName

**Variável de ambiente:** <PREFIX\_APPLICATIONNAME> (<PREFIX\_> é opcional e definida pelo usuário)

## Raiz do conteúdo

Essa configuração determina onde o host começa a procurar por arquivos de conteúdo.

**Chave:** contentRoot

**Tipo:** string

**Padrão:** o padrão é a pasta em que o assembly do aplicativo reside.

**Definido usando:** UseContentRoot

**Variável de ambiente:** <PREFIX\_CONTENTROOT> (<PREFIX\_> é opcional e definida pelo usuário)

Se o caminho não existir, o host não será iniciado.

```
var host = new HostBuilder()
    .UseContentRoot("c:\\<content-root>")
```

## Ambiente

Define o [ambiente](#) do aplicativo.

**Chave:** ambiente

**Tipo:** string

**Padrão:** Production

**Definido usando:** UseEnvironment

**Variável de ambiente:** <PREFIX\_ENVIRONMENT> (<PREFIX\_> é opcional e definida pelo usuário)

O ambiente pode ser definido como qualquer valor. Os valores definidos pela estrutura incluem `Development`, `Staging` e `Production`. Os valores não diferenciam maiúsculas de minúsculas.

```
var host = new HostBuilder()
    .UseEnvironment(EnvironmentName.Development)
```

## ConfigureHostConfiguration

`ConfigureHostConfiguration` usa um [IConfigurationBuilder](#) para criar um  [IConfiguration](#) para o host. A configuração do host é usada para inicializar o [IHostingEnvironment](#) para uso no processo de build do aplicativo.

`ConfigureHostConfiguration` pode ser chamado várias vezes com resultados aditivos. O host usa a opção que define um valor por último em uma chave determinada.

A configuração do host flui automaticamente para a configuração do aplicativo ([ConfigureAppConfiguration](#) e o restante do aplicativo).

Não há provedores incluídos por padrão. Você deve especificar explicitamente os provedores de configuração exigidos pelo aplicativo em `ConfigureHostConfiguration`, incluindo:

- Configuração de arquivo (por exemplo, de um arquivo `hostsettings.json`).
- Configuração de variável de ambiente.
- Configuração de argumento de linha de comando.
- Demais provedores de configuração necessários.

A configuração de arquivo do host é habilitada especificando o caminho base do aplicativo com `SetBasePath`

seguido por uma chamada a um dos [provedores de configuração do arquivo](#). O aplicativo de exemplo usa um arquivo JSON, `hostsettings.json`, e chamadas `AddJsonFile` para consumir as definições de configuração de host do arquivo.

Para adicionar uma [configuração de variável de ambiente](#) do host, chame `AddEnvironmentVariables` no construtor do host. `AddEnvironmentVariables` aceita um prefixo opcional definido pelo usuário. O aplicativo de exemplo usa um prefixo igual a `PREFIX_`. O prefixo é removido quando as variáveis de ambiente são lidas. Quando o host do aplicativo de exemplo é configurado, o valor da variável de ambiente de `PREFIX_ENVIRONMENT` torna-se o valor de configuração de host para a chave `environment`.

Durante o desenvolvimento, ao usar o [Visual Studio](#) ou executar um aplicativo com `dotnet run`, as variáveis de ambiente poderão ser definidas no arquivo `Properties/launchSettings.json`. No [Visual Studio Code](#), as variáveis de ambiente podem ser definidas no arquivo `.vscode/launch.json` durante o desenvolvimento. Para obter mais informações, consulte [Usar vários ambientes no ASP.NET Core](#).

A [configuração de linha de comando](#) é adicionada chamando `AddCommandLine`. A configuração de linha de comando é adicionada por último para permitir que os argumentos de linha de comando substituam a configuração fornecida pelos provedores de configuração anteriores.

`hostsettings.json`:

```
{  
  "environment": "Development"  
}
```

Configuração adicional pode ser fornecida com as chaves `applicationName` e `contentRoot`.

Exemplo da configuração `HostBuilder` usando `ConfigureHostConfiguration`:

```
var host = new HostBuilder()  
    .ConfigureHostConfiguration(configHost =>  
    {  
        configHost.SetBasePath(Directory.GetCurrentDirectory());  
        configHost.AddJsonFile("hostsettings.json", optional: true);  
        configHost.AddEnvironmentVariables(prefix: "PREFIX_");  
        configHost.AddCommandLine(args);  
    })
```

## ConfigureAppConfiguration

A configuração de aplicativo é criada chamando `ConfigureAppConfiguration` na implementação `IHostBuilder`. `ConfigureAppConfiguration` usa um `IConfigurationBuilder` para criar um `IConfiguration` para o aplicativo. `ConfigureAppConfiguration` pode ser chamado várias vezes com resultados aditivos. O aplicativo usa a opção que define um valor por último em uma chave determinada. A configuração criada pelo `ConfigureAppConfiguration` está disponível em `HostBuilderContext.Configuration` para operações subsequentes e em `Services`.

A configuração do aplicativo recebe automaticamente a configuração de host fornecida por `ConfigureHostConfiguration`.

Exemplo da configuração de aplicativo usando `ConfigureAppConfiguration`:

```

var host = new HostBuilder()
    .ConfigureAppConfiguration((hostContext, configApp) =>
{
    configApp.SetBasePath(Directory.GetCurrentDirectory());
    configApp.AddJsonFile("appsettings.json", optional: true);
    configApp.AddJsonFile(
        $"appsettings.{hostContext.HostingEnvironment.EnvironmentName}.json",
        optional: true);
    configApp.AddEnvironmentVariables(prefix: "PREFIX_");
    configApp.AddCommandLine(args);
})

```

*appsettings.json:*

```
{
    "Logging": {
        "LogLevel": {
            "Default": "Warning"
        }
    },
    "AllowedHosts": "*"
}
```

*appsettings.Development.json:*

```
{
    "Logging": {
        "LogLevel": {
            "Default": "Debug",
            "System": "Information",
            "Microsoft": "Information"
        }
    }
}
```

*appsettings.Production.json:*

```
{
    "Logging": {
        "LogLevel": {
            "Default": "Error",
            "System": "Information",
            "Microsoft": "Information"
        }
    }
}
```

Para mover arquivos de configurações para o diretório de saída, especifique os arquivos de configurações como [itens de projeto do MSBuild](#) no arquivo de projeto. O aplicativo de exemplo move os arquivos de configurações de aplicativo JSON e *hostsettings.json* com o seguinte item <Content> :

```
<ItemGroup>
    <Content Include="**\*.json" Exclude="bin\**\*;obj\**\*" CopyToOutputDirectory="PreserveNewest" />
</ItemGroup>
```

## ConfigureServices

[ConfigureServices](#) adiciona serviços ao contêiner [injeção de dependência](#) do aplicativo. `ConfigureServices` pode ser chamado várias vezes com resultados aditivos.

Um serviço hospedado é uma classe com lógica de tarefa em segundo plano que implementa a interface [IHostedService](#). Para obter mais informações, consulte [Tarefas em segundo plano com serviços hospedados no ASP.NET Core](#).

O [aplicativo de exemplo](#) usa o método de extensão `AddHostedService` para adicionar um serviço para eventos de tempo de vida, `LifetimeEventsHostedService`, e uma tarefa em segundo plano programada, `TimedHostedService`, para o aplicativo:

```
var host = new HostBuilder()
    .ConfigureServices((hostContext, services) =>
{
    if (hostContext.HostingEnvironment.IsDevelopment())
    {
        // Development service configuration
    }
    else
    {
        // Non-development service configuration
    }

    services.AddHostedService<LifetimeEventsHostedService>();
    services.AddHostedService<TimedHostedService>();
})
```

## ConfigureLogging

[ConfigureLogging](#) adiciona um delegado para configurar o [ILoggingBuilder](#) fornecido. `ConfigureLogging` pode ser chamado várias vezes com resultados aditivos.

```
var host = new HostBuilder()
    .ConfigureLogging((hostContext, configLogging) =>
{
    configLogging.AddConsole();
    configLogging.AddDebug();
})
```

### UseConsoleLifetime

[UseConsoleLifetime](#) escuta `Ctrl+C`/SIGINT ou SIGTERM e chama [StopApplication](#) para iniciar o processo de desligamento. `UseConsoleLifetime` desbloqueia extensões como [RunAsync](#) e [WaitForShutdownAsync](#).

[ConsoleLifetime](#) é previamente registrado como a implementação de tempo de vida padrão. O último tempo de vida registrado é usado.

```
var host = new HostBuilder()
    .UseConsoleLifetime()
```

## Configuração do contêiner

Para dar suporte à conexão de outros contêineres, o host pode aceitar um [IServiceProviderFactory<TContainerBuilder>](#). O fornecimento de um alocador não faz parte do registro de contêiner de injeção de dependência, mas, em vez disso, um host intrínseco é usado para criar o contêiner de DI concreto. [UseServiceProviderFactory\(IServiceProviderFactory<TContainerBuilder>\)](#) substitui o alocador padrão usado para criar o provedor de serviços do aplicativo.

A configuração de contêiner personalizado é gerenciada pelo método `ConfigureContainer`.

`ConfigureContainer`

fornecce uma experiência fortemente tipada para configurar o contêiner sobre a API do host subjacente.

`ConfigureContainer` pode ser chamado várias vezes com resultados aditivos.

Crie um contêiner de serviço para o aplicativo:

```
namespace GenericHostSample
{
    internal class ServiceContainer
    {
    }
}
```

Forneça um alocador de contêiner de serviço:

```
using System;
using Microsoft.Extensions.DependencyInjection;

namespace GenericHostSample
{
    internal class ServiceContainerFactory : IServiceProviderFactory<ServiceContainer>
    {
        public ServiceContainer CreateBuilder(IServiceCollection services)
        {
            return new ServiceContainer();
        }

        public IServiceProvider CreateServiceProvider(ServiceContainer containerBuilder)
        {
            throw new NotImplementedException();
        }
    }
}
```

Use o alocador e configure o contêiner de serviço personalizado para o aplicativo:

```
var host = new HostBuilder()
    .UseServiceProviderFactory<ServiceContainer>(new ServiceContainerFactory())
    .ConfigureContainer<ServiceContainer>((hostContext, container) =>
{
})
```

## Extensibilidade

Extensibilidade de host é executada com métodos de extensão em `IHostBuilder`.

O exemplo a seguir mostra como um método de extensão estende uma implementação do `IHostBuilder` com o exemplo do

[TimedHostedService](#), demonstrado no [Tarefas em segundo plano com serviços hospedados no ASP.NET Core](#).

```
var host = new HostBuilder()
    .UseHostedService<TimedHostedService>()
    .Build();

await host.StartAsync();
```

Um aplicativo estabelece o método de extensão `UseHostedService` para registrar o serviço hospedado passado no `T`:

```
using System;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

public static class Extensions
{
    public static IHostBuilder UseHostedService<T>(this IHostBuilder hostBuilder)
        where T : class, IHostedService, IDisposable
    {
        return hostBuilder.ConfigureServices(services =>
            services.AddHostedService<T>());
    }
}
```

## Gerenciar o host

A implementação [IHost](#) é responsável por iniciar e parar as implementações [IHostedService](#) que estão registradas no contêiner de serviço.

### Executar

[Run](#) executa o aplicativo e bloqueia o thread de chamada até que o host seja desligado:

```
public class Program
{
    public void Main(string[] args)
    {
        var host = new HostBuilder()
            .Build();

        host.Run();
    }
}
```

### RunAsync

[RunAsync](#) executa o aplicativo e retorna um [Task](#) que é concluído quando o token de cancelamento ou o desligamento é disparado:

```
public class Program
{
    public static async Task Main(string[] args)
    {
        var host = new HostBuilder()
            .Build();

        await host.RunAsync();
    }
}
```

### RunConsoleAsync

[RunConsoleAsync](#) habilita o suporte do console, compila e inicia o host e aguarda [Ctrl+C](#) /SIGINT ou SIGTERM desligar.

```
public class Program
{
    public static async Task Main(string[] args)
    {
        var hostBuilder = new HostBuilder();

        await hostBuilder.RunConsoleAsync();
    }
}
```

## Start e StopAsync

[Start](#) inicia o host de forma síncrona.

[StopAsync](#) tenta parar o host dentro do tempo limite fornecido.

```
public class Program
{
    public static async Task Main(string[] args)
    {
        var host = new HostBuilder()
            .Build();

        using (host)
        {
            host.Start();

            await host.StopAsync(TimeSpan.FromSeconds(5));
        }
    }
}
```

## StartAsync e StopAsync

[StartAsync](#) inicia o aplicativo.

[StopAsync](#) interrompe o aplicativo.

```
public class Program
{
    public static async Task Main(string[] args)
    {
        var host = new HostBuilder()
            .Build();

        using (host)
        {
            await host.StartAsync();

            await host.StopAsync();
        }
    }
}
```

## WaitForShutdown

[WaitForShutdown](#) é disparado por meio de [IHostLifetime](#), como [ConsoleLifetime](#) (escuta `Ctrl+C` /SIGINT ou SIGTERM). [WaitForShutdown](#) chama [StopAsync](#).

```
public class Program
{
    public void Main(string[] args)
    {
        var host = new HostBuilder()
            .Build();

        using (host)
        {
            host.Start();

            host.WaitForShutdown();
        }
    }
}
```

### WaitForShutdownAsync

`WaitForShutdownAsync` retorna um `Task` que é concluído quando o desligamento é disparado por meio do token fornecido e chama `StopAsync`.

```
public class Program
{
    public static async Task Main(string[] args)
    {
        var host = new HostBuilder()
            .Build();

        using (host)
        {
            await host.StartAsync();

            await host.WaitForShutdownAsync();
        }
    }
}
```

### Controle externo

O controle externo do host pode ser obtido usando os métodos que podem ser chamados externamente:

```

public class Program
{
    private IHost _host;

    public Program()
    {
        _host = new HostBuilder()
            .Build();
    }

    public async Task StartAsync()
    {
        _host.StartAsync();
    }

    public async Task StopAsync()
    {
        using (_host)
        {
            await _host.StopAsync(TimeSpan.FromSeconds(5));
        }
    }
}

```

`WaitForStartAsync` é chamado no início de `StartAsync`, que aguarda até que ele seja concluído antes de continuar. Isso pode ser usado para atrasar a inicialização até que seja sinalizado por um evento externo.

## Interface `IHostingEnvironment`

`IHostingEnvironment` fornece informações sobre o ambiente de hospedagem do aplicativo. Use a [injeção de construtor](#) para obter o `IHostingEnvironment` para usar suas propriedades e métodos de extensão:

```

public class MyClass
{
    private readonly IHostingEnvironment _env;

    public MyClass(IHostingEnvironment env)
    {
        _env = env;
    }

    public void DoSomething()
    {
        var environmentName = _env.EnvironmentName;
    }
}

```

Para obter mais informações, consulte [Usar vários ambientes no ASP.NET Core](#).

## Interface `IApplicationLifetime`

`IApplicationLifetime` permite atividades pós-inicialização e desligamento, inclusive solicitações de desligamento normal. Três propriedades na interface são tokens de cancelamento usados para registrar métodos `Action` que definem eventos de inicialização e desligamento.

TOKEN DE CANCELAMENTO	ACIONADO QUANDO...
<code>ApplicationStarted</code>	O host foi iniciado totalmente.

TOKEN DE CANCELAMENTO	ACIONADO QUANDO...
ApplicationStopped	O host está concluindo um desligamento normal. Todas as solicitações devem ser processadas. O desligamento é bloqueado até que esse evento seja concluído.
ApplicationStopping	O host está executando um desligamento normal. Solicitações ainda podem estar sendo processadas. O desligamento é bloqueado até que esse evento seja concluído.

O construtor injeta o serviço `IApplicationLifetime` em qualquer classe. O [aplicativo de exemplo](#) usa injeção de construtor em uma classe `LifetimeEventsHostedService` (uma implementação `IHostedService`) para registrar os eventos.

*LifetimeEventsHostedService.cs:*

```

internal class LifetimeEventsHostedService : IHostedService
{
    private readonly ILogger _logger;
    private readonly IApplicationLifetime _appLifetime;

    public LifetimeEventsHostedService(
        ILogger<LifetimeEventsHostedService> logger, IApplicationLifetime appLifetime)
    {
        _logger = logger;
        _appLifetime = appLifetime;
    }

    public Task StartAsync(CancellationToken cancellationToken)
    {
        _appLifetime.ApplicationStarted.Register(OnStarted);
        _appLifetime.ApplicationStopping.Register(OnStopping);
        _appLifetime.ApplicationStopped.Register(OnStopped);

        return Task.CompletedTask;
    }

    public Task StopAsync(CancellationToken cancellationToken)
    {
        return Task.CompletedTask;
    }

    private void OnStarted()
    {
        _logger.LogInformation("OnStarted has been called.");

        // Perform post-startup activities here
    }

    private void OnStopping()
    {
        _logger.LogInformation("OnStopping has been called.");

        // Perform on-stopping activities here
    }

    private void OnStopped()
    {
        _logger.LogInformation("OnStopped has been called.");

        // Perform post-stopped activities here
    }
}

```

[StopApplication](#) solicita o término do aplicativo. A classe a seguir usa [StopApplication](#) para desligar normalmente um aplicativo quando o método [Shutdown](#) da classe é chamado:

```
public class MyClass
{
    private readonly IApplicationLifetime _appLifetime;

    public MyClass(IApplicationLifetime appLifetime)
    {
        _appLifetime = appLifetime;
    }

    public void Shutdown()
    {
        _appLifetime.StopApplication();
    }
}
```

## Recursos adicionais

- [Tarefas em segundo plano com serviços hospedados no ASP.NET Core](#)
- [Hospedagem de exemplos do repositório no GitHub](#)

# Implementações de servidor Web em ASP.NET Core

21/01/2019 • 21 minutes to read • [Edit Online](#)

Por [Tom Dykstra](#), [Steve Smith](#), [Stephen Halter](#) e [Chris Ross](#)

Um aplicativo ASP.NET Core é executado com uma implementação do servidor HTTP em processo. A implementação do servidor escuta solicitações HTTP e apresenta-as para o aplicativo como um conjunto de [recursos de solicitação](#) compostos em um [HttpContext](#).

- [Windows](#)
- [macOS](#)
- [Linux](#)

O ASP.NET Core vem com os seguintes itens:

- O [servidor Kestrel](#) é a implementação padrão do servidor HTTP multiplataforma.
- O servidor HTTP do IIS é um [servidor em processo](#) do IIS.
- O [servidor HTTP.sys](#) é um servidor HTTP somente do Windows com base no [driver do kernel HTTP.sys](#) e na [API do servidor HTTP](#).

Ao usar o [IIS](#) ou o [IIS Express](#), o aplicativo é executado:

- No mesmo processo que o processo de trabalho do IIS (o [modelo de hospedagem em processo](#)) com o [servidor HTTP do IIS](#). *Em processo* é a configuração recomendada.
- Em um processo separado do processo de trabalho do IIS (o [modelo de hospedagem fora do processo](#)) com o [servidor Kestrel](#).

O [módulo do ASP.NET Core](#) é um módulo nativo do IIS que manipula as solicitações nativas do IIS entre o IIS e o servidor HTTP do IIS em processo ou o Kestrel. Para obter mais informações, consulte [Módulo do ASP.NET Core](#).

## Modelos de hospedagem

### Modelo de hospedagem em processo

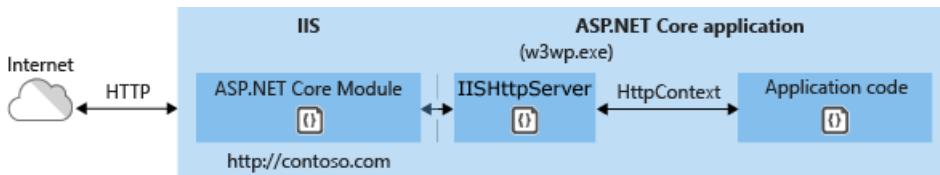
Usando uma hospedagem em processo, um aplicativo ASP.NET Core é executado no mesmo processo que seu processo de trabalho do IIS. A hospedagem em processo oferece desempenho melhor em hospedagem fora do processo porque as solicitações não são transmitidas por proxy pelo adaptador de loopback, um adaptador de rede que retorna o tráfego de rede de saída para o mesmo computador. O IIS manipula o gerenciamento de processos com o [WAS \(Serviço de Ativação de Processos do Windows\)](#).

O Módulo do ASP.NET Core:

- Executa a inicialização do aplicativo.
  - Carrega o [CoreCLR](#).
  - Chama `Program.Main`.
- Manipula o tempo de vida da solicitação nativa do IIS.

Não há suporte para o modelo de hospedagem em processo para aplicativos ASP.NET Core direcionados ao .NET Framework.

O diagrama a seguir ilustra a relação entre o IIS, o Módulo do ASP.NET Core e um aplicativo hospedado em processo:



A solicitação chega da Web para o driver do HTTP.sys no modo kernel. O driver roteia as solicitações nativas ao IIS na porta configurada do site, normalmente, a 80 (HTTP) ou a 443 (HTTPS). O módulo recebe a solicitação nativa e a passa para o Servidor HTTP do IIS (`IISHttpServer`). O servidor HTTP do IIS é uma implementação de servidor em processo do IIS que converte a solicitação de nativa para gerenciada.

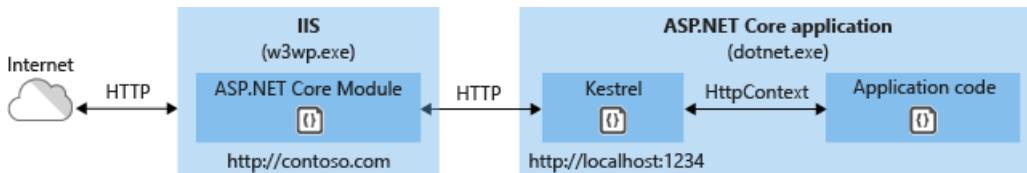
Depois que o Servidor HTTP do IIS processa a solicitação, a solicitação é enviada por push para o pipeline de middleware do ASP.NET Core. O pipeline do middleware manipula a solicitação e a passa como uma instância de `HttpContext` para a lógica do aplicativo. A resposta do aplicativo é retornada ao IIS, que a retorna por push para o cliente que iniciou a solicitação.

A hospedagem em processo é uma opção de aceitação para os aplicativos existentes, mas o padrão dos modelos `dotnet new` é o modelo de hospedagem em processo para todos os cenários do IIS e do IIS Express.

### Modelo de hospedagem de fora do processo

Como os aplicativos ASP.NET Core são executados em um processo separado do processo de trabalho do IIS, o módulo realiza o gerenciamento de processos. O módulo inicia o processo para o aplicativo ASP.NET Core quando a primeira solicitação chega e reinicia o aplicativo se ele é desligado ou falha. Isso é basicamente o mesmo comportamento que o dos aplicativos que são executados dentro do processo e são gerenciados pelo [WAS \(Serviço de Ativação de Processos do Windows\)](#).

O diagrama a seguir ilustra a relação entre o IIS, o Módulo do ASP.NET Core e um aplicativo hospedado de fora de processo:



As solicitações chegam da Web para o driver do HTTP.sys no modo kernel. O driver roteia as solicitações ao IIS na porta configurada do site, normalmente, a 80 (HTTP) ou a 443 (HTTPS). O módulo encaminha as solicitações ao Kestrel em uma porta aleatória do aplicativo, que não seja a porta 80 ou 443.

O módulo especifica a porta por meio de uma variável de ambiente na inicialização e o middleware de integração do IIS configura o servidor para escutar em `http://localhost:{PORT}`. Outras verificações são executadas e as solicitações que não se originam do módulo são rejeitadas. O módulo não é compatível com encaminhamento de HTTPS, portanto, as solicitações são encaminhadas por HTTP, mesmo se recebidas pelo IIS por HTTPS.

Depois que o Kestrel coleta a solicitação do módulo, a solicitação é enviada por push ao pipeline do middleware do ASP.NET Core. O pipeline do middleware manipula a solicitação e a passa como uma instância de `HttpContext` para a lógica do aplicativo. O middleware adicionado pela integração do IIS atualiza o esquema, o IP remoto e pathbase para encaminhar a solicitação para o Kestrel. A resposta do aplicativo é retornada ao IIS, que a retorna por push para o cliente HTTP que iniciou a solicitação.

Para obter as diretrizes de configuração do IIS e do módulo do ASP.NET Core, confira os tópicos a seguir:

- [Hospedar o ASP.NET Core no Windows com o IIS](#)
- [Módulo do ASP.NET Core](#)
- [Windows](#)

- macOS
- Linux

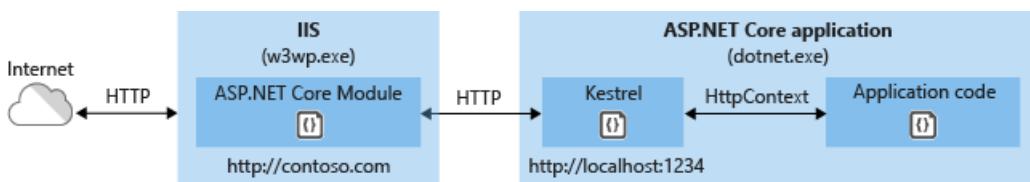
O ASP.NET Core vem com os seguintes itens:

- O servidor **Kestrel** é o servidor HTTP padrão multiplataforma.
- O servidor **HTTP.sys** é um servidor HTTP somente do Windows com base no [driver do kernel HTTP.sys e na API do servidor HTTP](#).

Ao usar o [IIS](#) ou o [IIS Express](#), o aplicativo é executado em um processo separado do processo de trabalho do IIS (*fora do processo*) com o [servidor Kestrel](#).

Como os aplicativos ASP.NET Core são executados em um processo separado do processo de trabalho do IIS, o módulo realiza o gerenciamento de processos. O módulo inicia o processo para o aplicativo ASP.NET Core quando a primeira solicitação chega e reinicia o aplicativo se ele é desligado ou falha. Isso é basicamente o mesmo comportamento que o dos aplicativos que são executados dentro do processo e são gerenciados pelo [WAS \(Serviço de Ativação de Processos do Windows\)](#).

O diagrama a seguir ilustra a relação entre o IIS, o Módulo do ASP.NET Core e um aplicativo hospedado de fora d processo:



As solicitações chegam da Web para o driver do HTTP.sys no modo kernel. O driver roteia as solicitações ao IIS na porta configurada do site, normalmente, a 80 (HTTP) ou a 443 (HTTPS). O módulo encaminha as solicitações ao Kestrel em uma porta aleatória do aplicativo, que não seja a porta 80 ou 443.

O módulo especifica a porta por meio de uma variável de ambiente na inicialização e o middleware de integração do IIS configura o servidor para escutar em `http://localhost:{port}`. Outras verificações são executadas e as solicitações que não se originam do módulo são rejeitadas. O módulo não é compatível com encaminhamento de HTTPS, portanto, as solicitações são encaminhadas por HTTP, mesmo se recebidas pelo IIS por HTTPS.

Depois que o Kestrel coleta a solicitação do módulo, a solicitação é enviada por push ao pipeline do middleware do ASP.NET Core. O pipeline do middleware manipula a solicitação e a passa como uma instância de `HttpContext` para a lógica do aplicativo. O middleware adicionado pela integração do IIS atualiza o esquema, o IP remoto e pathbase para encaminhar a solicitação para o Kestrel. A resposta do aplicativo é retornada ao IIS, que a retorna por push para o cliente HTTP que iniciou a solicitação.

Para obter as diretrizes de configuração do IIS e do módulo do ASP.NET Core, confira os tópicos a seguir:

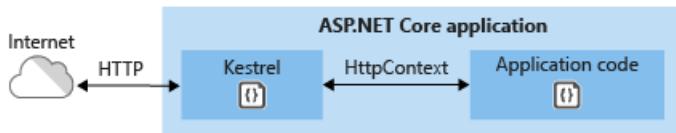
- [Hospedar o ASP.NET Core no Windows com o IIS](#)
- [Módulo do ASP.NET Core](#)

## Kestrel

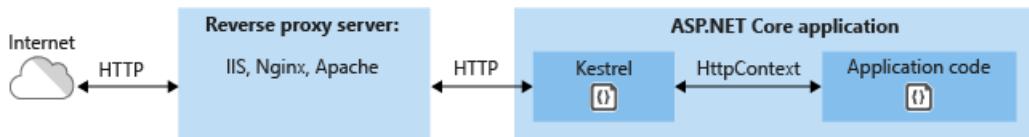
O Kestrel é o servidor Web padrão incluído nos modelos de projeto do ASP.NET Core.

O Kestrel pode ser usado:

- Sozinho, como um servidor de borda que processa solicitações diretamente de uma rede, incluindo a Internet.

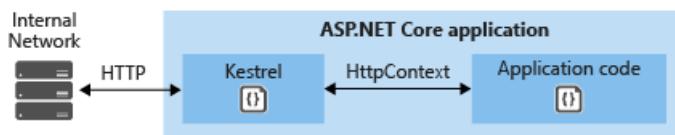


- Com um *servidor proxy reverso* como [IIS \(Serviços de Informações da Internet\)](#), [Nginx](#) ou [Apache](#). Um servidor proxy reverso recebe solicitações HTTP da Internet e encaminha-as para o Kestrel.

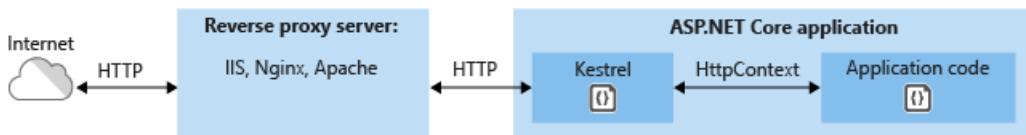


Qualquer configuração de hospedagem (com ou sem um servidor proxy reverso) é compatível com os aplicativos ASP.NET Core 2.1 ou posteriores.

Se o aplicativo aceita somente solicitações de uma rede interna, o Kestrel pode ser usado sozinho.



Se o aplicativo for exposto à Internet, o Kestrel deverá usar um *servidor proxy reverso*, como o [IIS \(Serviços de Informações da Internet\)](#), [Nginx](#) ou [Apache](#). Um servidor proxy reverso recebe solicitações HTTP da Internet e encaminha-as para o Kestrel.



O motivo mais importante para usar um proxy reverso para implantações de servidores de borda voltados para o público que são expostas diretamente na Internet é a segurança. As versões 1.x do Kestrel não incluem recursos de segurança importantes para proteção contra ataques da Internet. Isso inclui, mas não se limita aos tempos limite, limites de tamanho da solicitação e limites de conexões simultâneas apropriados.

Para obter diretrizes de configuração do Kestrel e informações sobre quando usar o Kestrel em uma configuração de proxy reverso, confira [Implementação do servidor Web Kestrel no ASP.NET Core](#).

### Nginx com Kestrel

Para obter informações sobre como usar Nginx no Linux como um servidor proxy reverso para Kestrel, confira [Host ASP.NET Core no Linux com Nginx](#).

### Apache com Kestrel

Para obter informações sobre como usar Apache no Linux como um servidor proxy reverso para Kestrel, confira [Hospedar o ASP.NET Core no Linux com o Apache](#).

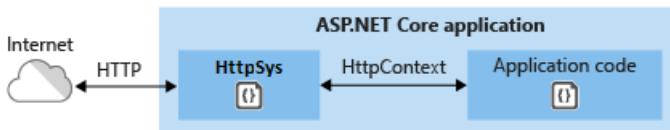
## Servidor HTTP do IIS

O servidor HTTP do IIS é um [servidor em processo](#) do IIS e é necessário para implantações em processo. O [módulo do ASP.NET Core](#) manipula solicitações nativas do IIS entre o IIS e o servidor HTTP do IIS. Para obter mais informações, consulte [Módulo do ASP.NET Core](#).

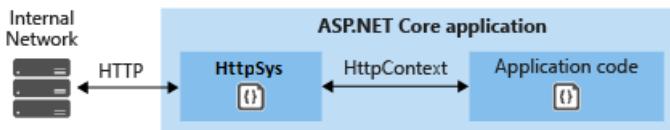
## HTTP.sys

Se os aplicativos ASP.NET Core forem executados no Windows, o HTTP.sys será uma alternativa ao Kestrel. O Kestrel geralmente é recomendado para melhor desempenho. O HTTP.sys pode ser usado em cenários em que o aplicativo é exposto à Internet e as funcionalidades necessárias são compatíveis com HTTP.sys, mas não com

Kestrel. Para obter mais informações, consulte [Implementação do servidor Web HTTP.sys no ASP.NET Core](#).



O HTTP.sys também pode ser usado para aplicativos que são expostos somente a uma rede interna.



Para obter as diretrizes de configuração do HTTP.sys, confira [Implementação do servidor Web HTTP.sys no ASP.NET Core](#).

## Infraestrutura de servidor do ASP.NET Core

O [IApplicationBuilder](#) disponível no método `Startup.Configure` expõe a propriedade [ServerFeatures](#) do tipo [IFeatureCollection](#). O Kestrel e o HTTP.sys expõem apenas um único recurso cada, o [IServerAddressesFeature](#), mas diferentes implementações de servidor podem expor funcionalidades adicionais.

[IServerAddressesFeature](#) pode ser usado para descobrir a qual porta a implementação do servidor se acoplou durante o tempo de execução.

## Servidores personalizados

Se os servidores internos não atenderem aos requisitos do aplicativo, um servidor personalizado poderá ser criado. O [Guia de OWIN \(Open Web Interface para .NET\)](#) demonstra como gravar uma implementação de [IServer](#) com base em [Nowin](#). Somente as interfaces de recurso que o aplicativo usa exigem implementação. Porém, no mínimo [IHttpRequestFeature](#) e [IHttpResponseFeature](#) devem ter suporte.

## Inicialização do servidor

O servidor é iniciado quando o IDE (Ambiente de Desenvolvimento Integrado) ou o editor inicia o aplicativo:

- [Visual Studio](#) – os perfis de inicialização podem ser usados para iniciar o aplicativo e o servidor com o [IIS Express/Módulo do ASP.NET Core](#) ou o console.
- [Visual Studio Code](#) – o aplicativo e o servidor são iniciados pelo [Omnisharp](#), que ativa o depurador CoreCLR.
- [Visual Studio para Mac](#) – o aplicativo e o servidor são iniciados pelo [Depurador de modo suave Mono](#).

Ao iniciar o aplicativo usando um prompt de comando na pasta do projeto, o [dotnet run](#) inicia o aplicativo e o servidor (apenas Kestrel e HTTP.sys). A configuração é especificada pela opção `-c|--configuration`, que é definida como `Debug` (padrão) ou `Release`. Se os perfis de inicialização estiverem presentes em um arquivo `launchSettings.json`, use a opção `--launch-profile <NAME>` para definir o perfil de inicialização (por exemplo, `Development` ou `Production`). Para obter mais informações, confira [dotnet run](#) e [pacote de distribuição do .NET Core](#).

## Compatibilidade com HTTP/2

O [HTTP/2](#) é compatível com ASP.NET Core nos seguintes cenários de implantação:

- [Kestrel](#)
  - Sistema operacional
    - Windows Server 2016/Windows 10 ou posterior†

- Linux com OpenSSL 1.0.2 ou posterior (por exemplo, Ubuntu 16.04 ou posterior)
- O HTTP/2 será compatível com macOS em uma versão futura.
- Estrutura de destino: .NET Core 2.2 ou posterior
- [HTTP.sys](#)
  - Windows Server 2016/Windows 10 ou posterior
  - Estrutura de destino: Não aplicável a implantações do HTTP.sys.
- [IIS \(em processo\)](#)
  - Windows Server 2016/Windows 10 ou posterior; IIS 10 ou posterior
  - Estrutura de destino: .NET Core 2.2 ou posterior
- [IIS \(fora do processo\)](#)
  - Windows Server 2016/Windows 10 ou posterior; IIS 10 ou posterior
  - Conexões de servidor de borda voltadas para o público usam HTTP/2, mas a conexão de proxy reverso para o Kestrel usa HTTP/1.1.
  - Estrutura de destino: Não aplicável a implantações fora do processo do IIS.

†O Kestrel tem suporte limitado para HTTP/2 no Windows Server 2012 R2 e Windows 8.1. O suporte é limitado porque a lista de conjuntos de codificação TLS disponível nesses sistemas operacionais é limitada. Um certificado gerado usando um ECDSA (Algoritmo de Assinatura Digital Curva Elíptica) pode ser necessário para proteger conexões TLS.

- [HTTP.sys](#)
  - Windows Server 2016/Windows 10 ou posterior
  - Estrutura de destino: Não aplicável a implantações do HTTP.sys.
- [IIS \(fora do processo\)](#)
  - Windows Server 2016/Windows 10 ou posterior; IIS 10 ou posterior
  - Conexões de servidor de borda voltadas para o público usam HTTP/2, mas a conexão de proxy reverso para o Kestrel usa HTTP/1.1.
  - Estrutura de destino: Não aplicável a implantações fora do processo do IIS.

Uma conexão HTTP/2 precisa usar [ALPN \(Application-Layer Protocol Negotiation\)](#) e TLS 1.2 ou posterior. Para obter mais informações, consulte os tópicos referentes aos seus cenários de implantação do servidor.

## Recursos adicionais

- [Implementação do servidor Web Kestrel no ASP.NET Core](#)
- [Módulo do ASP.NET Core](#)
- [Hospedar o ASP.NET Core no Windows com o IIS](#)
- [Implantar aplicativos ASP.NET Core no Serviço de Aplicativo do Azure](#)
- [Host ASP.NET Core no Linux com Nginx](#)
- [Hospedar o ASP.NET Core no Linux com o Apache](#)
- [Implementação do servidor Web HTTP.sys no ASP.NET Core](#)

# Iniciar solicitações HTTP

30/01/2019 • 24 minutes to read • [Edit Online](#)

Por [Glenn Condron](#), [Ryan Nowak](#) e [Steve Gordon](#)

É possível registrar e usar um `IHttpClientFactory` para configurar e criar instâncias de `HttpClient` em um aplicativo. Ele oferece os seguintes benefícios:

- Fornece um local central para nomear e configurar instâncias lógicas de `HttpClient`. Por exemplo, um cliente `github` pode ser registrado e configurado para acessar o GitHub. Um cliente padrão pode ser registrado para outras finalidades.
- Codifica o conceito de middleware de saída por meio da delegação de manipuladores no `HttpClient` e fornece extensões para que o middleware baseado no Polly possa aproveitar esse recurso.
- Gerencia o pooling e o tempo de vida das instâncias de `HttpClientMessageHandler` subjacentes para evitar problemas de DNS comuns que ocorrem no gerenciamento manual de tempos de vida de `HttpClient`.
- Adiciona uma experiência de registro em log configurável (via `ILogger`) para todas as solicitações enviadas por meio de clientes criados pelo alocador.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Pré-requisitos

Os projetos direcionados ao .NET Framework exigem a instalação do pacote do NuGet `Microsoft.Extensions.Http`.

Os projetos destinados ao .Net Core e a referência ao [metapacote Microsoft.AspNetCore.App](#) já incluem o pacote `Microsoft.Extensions.Http`.

## Padrões de consumo

Há várias maneiras de usar o `IHttpClientFactory` em um aplicativo:

- [Uso básico](#)
- [Clientes nomeados](#)
- [Clientes com tipo](#)
- [Clientes gerados](#)

Nenhum deles é estritamente superiores ao outro. A melhor abordagem depende das restrições do aplicativo.

### Uso básico

O `IHttpClientFactory` pode ser registrado chamando o método de extensão `AddHttpClient` em `IServiceCollection`, dentro do método `Startup.ConfigureServices`.

```
services.AddHttpClient();
```

Depois de registrado, o código pode aceitar um `IHttpClientFactory` em qualquer lugar em que os serviços possam ser injetados com [injeção de dependência](#). O `IHttpClientFactory` pode ser usado para criar uma instância de `HttpClient`:

```

public class BasicUsageModel : PageModel
{
    private readonly IHttpClientFactory _clientFactory;

    public IEnumerable<GitHubBranch> Branches { get; private set; }

    public bool GetBranchesError { get; private set; }

    public BasicUsageModel(IHttpClientFactory clientFactory)
    {
        _clientFactory = clientFactory;
    }

    public async Task OnGet()
    {
        var request = new HttpRequestMessage(HttpMethod.Get,
            "https://api.github.com/repos/aspnet/docs/branches");
        request.Headers.Add("Accept", "application/vnd.github.v3+json");
        request.Headers.Add("User-Agent", "HttpClientFactory-Sample");

        var client = _clientFactory.CreateClient();

        var response = await client.SendAsync(request);

        if (response.IsSuccessStatusCode)
        {
            Branches = await response.Content
                .ReadAsAsync<IEnumerable<GitHubBranch>>();
        }
        else
        {
            GetBranchesError = true;
            Branches = Array.Empty<GitHubBranch>();
        }
    }
}

```

Usar o `IHttpClientFactory` dessa forma é uma ótima maneira de refatorar um aplicativo existente. Não há nenhum impacto na maneira em que o `HttpClient` é usado. Nos locais em que as instâncias de `HttpClient` são criadas no momento, substitua essas ocorrências por uma chamada a `CreateClient`.

## Clientes nomeados

Quando um aplicativo exige vários usos distintos do `HttpClient`, cada um com uma configuração diferente, uma opção é usar **clientes nomeados**. A configuração de um `HttpClient` nomeado pode ser especificada durante o registro em `Startup.ConfigureServices`.

```

services.AddHttpClient("github", c =>
{
    c.BaseAddress = new Uri("https://api.github.com/");
    // Github API versioning
    c.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
    // Github requires a user-agent
    c.DefaultRequestHeaders.Add("User-Agent", "HttpClientFactory-Sample");
});

```

No código anterior, `AddHttpClient` é chamado, fornecendo o nome *github*. Esse cliente tem algumas configurações padrão aplicadas, ou seja, o endereço básico e dois cabeçalhos necessários para trabalhar com a API do GitHub.

Toda vez que o `CreateClient` é chamado, uma nova instância de `HttpClient` é criada e a ação de configuração é chamada.

Para consumir um cliente nomeado, um parâmetro de cadeia de caracteres pode ser passado para `CreateClient`.

Especifique o nome do cliente a ser criado:

```
public class NamedClientModel : PageModel
{
    private readonly IHttpClientFactory _clientFactory;

    public IEnumerable<GitHubPullRequest> PullRequests { get; private set; }

    public bool GetPullRequestsError { get; private set; }

    public bool HasPullRequests => PullRequests.Any();

    public NamedClientModel(IHttpClientFactory clientFactory)
    {
        _clientFactory = clientFactory;
    }

    public async Task OnGet()
    {
        var request = new HttpRequestMessage(HttpMethod.Get,
            "repos/aspnet/docs/pulls");

        var client = _clientFactory.CreateClient("github");

        var response = await client.SendAsync(request);

        if (response.IsSuccessStatusCode)
        {
            PullRequests = await response.Content
                .ReadAsStringAsync<IEnumerable<GitHubPullRequest>>();
        }
        else
        {
            GetPullRequestsError = true;
            PullRequests = Array.Empty<GitHubPullRequest>();
        }
    }
}
```

No código anterior, a solicitação não precisa especificar um nome do host. Ela pode passar apenas o caminho, pois o endereço básico configurado para o cliente é usado.

## Clientes com tipo

Os clientes com tipo fornecem as mesmas funcionalidade que os clientes nomeados sem a necessidade de usar cadeias de caracteres como chaves. A abordagem de cliente com tipo fornece a ajuda do IntelliSense e do compilador durante o consumo de clientes. Eles fornecem um único local para configurar e interagir com um determinado `HttpClient`. Por exemplo, um único cliente com tipo pode ser usado para um único ponto de extremidade de back-end e encapsular toda a lógica que lida com esse ponto de extremidade. Outra vantagem é que eles funcionam com a DI e podem ser injetados no local necessário no aplicativo.

Um cliente com tipo aceita um parâmetro `HttpClient` em seu construtor:

```
public class GitHubService
{
    public HttpClient Client { get; }

    public GitHubService(HttpClient client)
    {
        client.BaseAddress = new Uri("https://api.github.com/");
        // GitHub API versioning
        client.DefaultRequestHeaders.Add("Accept",
            "application/vnd.github.v3+json");
        // GitHub requires a user-agent
        client.DefaultRequestHeaders.Add("User-Agent",
            "HttpClientFactory-Sample");

        Client = client;
    }

    public async Task<IEnumerable<GitHubIssue>> GetAspNetDocsIssues()
    {
        var response = await Client.GetAsync(
            "/repos/aspnet/docs/issues?state=open&sort=created&direction=desc");

        response.EnsureSuccessStatusCode();

        var result = await response.Content
            .ReadAsAsync<IEnumerable<GitHubIssue>>();

        return result;
    }
}
```

No código anterior, a configuração é movida para o cliente com tipo. O objeto `HttpClient` é exposto como uma propriedade pública. É possível definir métodos específicos da API que expõem a funcionalidade `HttpClient`. O método `GetAspNetDocsIssues` encapsula o código necessário para consultar e analisar os últimos problemas em aberto de um repositório GitHub.

Para registrar um cliente com tipo, o método de extensão `AddHttpClient` genérico pode ser usado em `Startup.ConfigureServices`, especificando a classe do cliente com tipo:

```
services.AddHttpClient<GitHubService>();
```

O cliente com tipo é registrado como transitório com a DI. O cliente com tipo pode ser injetado e consumido diretamente:

```
public class TypedClientModel : PageModel
{
    private readonly GitHubService _gitHubService;

    public IEnumerable<GitHubIssue> LatestIssues { get; private set; }

    public bool HasIssue => LatestIssues.Any();

    public bool GetIssuesError { get; private set; }

    public TypedClientModel(GitHubService gitHubService)
    {
        _gitHubService = gitHubService;
    }

    public async Task OnGet()
    {
        try
        {
            LatestIssues = await _gitHubService.GetAspNetDocsIssues();
        }
        catch(HttpRequestException)
        {
            GetIssuesError = true;
            LatestIssues = Array.Empty<GitHubIssue>();
        }
    }
}
```

Se preferir, a configuração de um cliente com tipo poderá ser especificada durante o registro em `Startup.ConfigureServices` e não no construtor do cliente com tipo:

```
services.AddHttpClient<RepoService>(c =>
{
    c.BaseAddress = new Uri("https://api.github.com/");
    c.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
    c.DefaultRequestHeaders.Add("User-Agent", "HttpClientFactory-Sample");
});
```

É possível encapsular totalmente o `HttpClient` dentro de um cliente com tipo. Em vez de o expor como uma propriedade, é possível fornecer métodos públicos que chamam a instância de `HttpClient` internamente.

```

public class RepoService
{
    // _httpClient isn't exposed publicly
    private readonly HttpClient _httpClient;

    public RepoService(HttpClient client)
    {
        _httpClient = client;
    }

    public async Task<IEnumerable<string>> GetRepos()
    {
        var response = await _httpClient.GetAsync("aspnet/repos");

        response.EnsureSuccessStatusCode();

        var result = await response.Content
            .ReadAsAsync<IEnumerable<string>>();

        return result;
    }
}

```

No código anterior, o `HttpClient` é armazenado como um campo privado. Todo o acesso para fazer chamadas externas passa pelo método `GetRepos`.

## Clientes gerados

O `IHttpClientFactory` pode ser usado em combinação com outras bibliotecas de terceiros, como a [Refit](#). Refit é uma biblioteca REST para .NET. Ela converte APIs REST em interfaces dinâmicas. Uma implementação da interface é gerada dinamicamente pelo `RestService` usando `HttpClient` para fazer as chamadas de HTTP externas.

Uma interface e uma resposta são definidas para representar a API externa e sua resposta:

```

public interface IHelloClient
{
    [Get("/helloworld")]
    Task<Reply> GetMessageAsync();
}

public class Reply
{
    public string Message { get; set; }
}

```

Um cliente com tipo pode ser adicionado usando a Refit para gerar a implementação:

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddHttpClient("hello", c =>
    {
        c.BaseAddress = new Uri("http://localhost:5000");
    })
    .AddTypedClient(c => Refit.RestService.For<IHelloClient>(c));

    services.AddMvc();
}

```

A interface definida pode ser consumida, quando necessário, com a implementação fornecida pela DI e pela Refit:

```
[ApiController]
public class ValuesController : ControllerBase
{
    private readonly IHelloClient _client;

    public ValuesController(IHelloClient client)
    {
        _client = client;
    }

    [HttpGet("/")]
    public async Task<ActionResult<Reply>> Index()
    {
        return await _client.GetMessageAsync();
    }
}
```

## Middleware de solicitação de saída

O `HttpClient` já tem o conceito de delegar manipuladores que podem ser vinculados para solicitações HTTP de saída. O `IHttpClientFactory` facilita a definição dos manipuladores a serem aplicados a cada cliente nomeado. Ele permite o registro e o encadeamento de vários manipuladores para criar um pipeline do middleware de solicitação saída. Cada um desses manipuladores é capaz de executar o trabalho antes e após a solicitação de saída. Esse padrão é semelhante ao pipeline do middleware de entrada no ASP.NET Core. O padrão fornece um mecanismo para gerenciar interesses paralelos em relação às solicitações HTTP, incluindo o armazenamento em cache, o tratamento de erro, a serialização e o registro em log.

Para criar um manipulador, defina uma classe derivando-a de `DelegatingHandler`. Substitua o método `SendAsync` para executar o código antes de passar a solicitação para o próximo manipulador no pipeline:

```
public class ValidateHeaderHandler : DelegatingHandler
{
    protected override async Task<HttpResponseMessage> SendAsync(
        HttpRequestMessage request,
        CancellationToken cancellationToken)
    {
        if (!request.Headers.Contains("X-API-KEY"))
        {
            return new HttpResponseMessage(HttpStatusCode.BadRequest)
            {
                Content = new StringContent(
                    "You must supply an API key header called X-API-KEY")
            };
        }

        return await base.SendAsync(request, cancellationToken);
    }
}
```

O código anterior define um manipulador básico. Ele verifica se um cabeçalho `X-API-KEY` foi incluído na solicitação. Se o cabeçalho estiver ausente, isso poderá evitar a chamada de HTTP e retornar uma resposta adequada.

Durante o registro, um ou mais manipuladores podem ser adicionados à configuração de um `HttpClient`. Essa tarefa é realizada por meio de métodos de extensão no `IHttpClientBuilder`.

```
services.AddTransient<ValidateHeaderHandler>();

services.AddHttpClient("externalservice", c =>
{
    // Assume this is an "external" service which requires an API KEY
    c.BaseAddress = new Uri("https://localhost:5000/");
})
.AddHttpMessageHandler<ValidateHeaderHandler>();
```

No código anterior, o `ValidateHeaderHandler` é registrado com a DI. O `IHttpClientFactory` cria um escopo de injeção de dependência separado para cada manipulador. Os manipuladores podem depender de serviços de qualquer escopo. Os serviços dos quais os manipuladores dependem são descartados quando o manipulador é descartado.

Depois de registrado, o `AddHttpMessageHandler` poderá ser chamado, passando o tipo para o manipulador.

No código anterior, o `ValidateHeaderHandler` é registrado com a DI. O manipulador **deve** ser registrado na injeção de dependência como serviço temporário, mas nunca com escopo. Se o manipulador for registrado como um serviço com escopo e os serviços dos quais ele depende forem descartáveis, esses serviços poderão ser descartados antes que ele saia do escopo. Isso causaria falha no manipulador.

Depois de registrado, o `AddHttpMessageHandler` poderá ser chamado, passando no tipo do manipulador.

Vários manipuladores podem ser registrados na ordem em que eles devem ser executados. Cada manipulador encapsula o próximo manipulador até que o `HttpClientHandler` final execute a solicitação:

```
services.AddTransient<SecureRequestHandler>();
services.AddTransient<requestDataHandler>();

services.AddHttpClient("clientwithhandlers")
    // This handler is on the outside and called first during the
    // request, last during the response.
    .AddHttpMessageHandler<SecureRequestHandler>()
    // This handler is on the inside, closest to the request being
    // sent.
    .AddHttpMessageHandler<requestDataHandler>();
```

Use uma das seguintes abordagens para compartilhar o estado por solicitação com os manipuladores de mensagens:

- Passe os dados pelo manipulador usando `HttpRequestMessage.Properties`.
- Use `IHttpContextAccessor` para acessar a solicitação atual.
- Crie um objeto de armazenamento `AsyncLocal` personalizado para passar os dados.

## Usar manipuladores baseados no Polly

O `IHttpClientFactory` integra-se a uma biblioteca de terceiros popular chamada [Polly](#). O Polly é uma biblioteca abrangente de tratamento de falha transitória e de resiliência para .NET. Ela permite que os desenvolvedores expressem políticas, como Repetição, Disjuntor, Tempo Limite, Isolamento de Bulkhead e Fallback de maneira fluente e thread-safe.

Os métodos de extensão são fornecidos para habilitar o uso de políticas do Polly com instâncias de `HttpClient` configuradas. As extensões do Polly estão disponíveis no pacote do NuGet [Microsoft.Extensions.Http.Polly](#). Esse pacote não está incluído no [metapacote Microsoft.AspNetCore.App](#). Para usar as extensões, um `<PackageReference />` explícito deve ser incluído no projeto.

```

<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp2.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.App" />
    <PackageReference Include="Microsoft.Extensions.Http.Polly" Version="2.1.1" />
  </ItemGroup>

</Project>

```

Depois de restaurar este pacote, os métodos de extensão ficam disponíveis para permitir a adição de manipuladores baseados no Polly aos clientes.

### Tratar falhas transitórias

As falhas mais comuns ocorrem quando as chamadas HTTP externas são transitórias. Um método de extensão conveniente chamado `AddTransientHttpErrorPolicy` é incluído para permitir que uma política seja definida para tratar erros transitórios. As políticas configuradas com esse método de extensão tratam `HttpRequestException`, respostas HTTP 5xx e respostas HTTP 408.

A extensão `AddTransientHttpErrorPolicy` pode ser usada em `Startup.ConfigureServices`. A extensão fornece acesso a um objeto `PolicyBuilder` configurado para tratar erros que representam uma possível falha transitória:

```

services.AddHttpClient<UnreliableEndpointCallerService>()
    .AddTransientHttpErrorPolicy(p =>
        p.WaitAndRetryAsync(3, _ => TimeSpan.FromMilliseconds(600)));

```

No código anterior, uma política `WaitAndRetryAsync` é definida. As solicitações com falha são repetidas até três vezes com um atraso de 600 ms entre as tentativas.

### Selecionar políticas dinamicamente

Existem métodos de extensão adicionais que podem ser usados para adicionar manipuladores baseados no Polly. Uma dessas extensões é a `AddPolicyHandler`, que tem várias sobrecargas. Uma sobrecarga permite que a solicitação seja inspecionada durante a definição da política a ser aplicada:

```

var timeout = Policy.TimeoutAsync<HttpResponseMessage>(
    TimeSpan.FromSeconds(10));
var longTimeout = Policy.TimeoutAsync<HttpResponseMessage>(
    TimeSpan.FromSeconds(30));

services.AddHttpClient("conditionalpolicy")
    // Run some code to select a policy based on the request
    .AddPolicyHandler(request =>
        request.Method == HttpMethod.Get ? timeout : longTimeout);

```

No código anterior, se a solicitação de saída é um GET, é aplicado um tempo limite de 10 segundos. Para qualquer outro método HTTP, é usado um tempo limite de 30 segundos.

### Adicionar vários manipuladores do Polly

É comum aninhar políticas do Polly para fornecer uma funcionalidade avançada:

```
services.AddHttpClient("multiplepolicies")
    .AddTransientHttpErrorPolicy(p => p.RetryAsync(3))
    .AddTransientHttpErrorPolicy(
        p => p.CircuitBreakerAsync(5, TimeSpan.FromSeconds(30)));
```

No exemplo anterior, dois manipuladores são adicionados. O primeiro usa a extensão

`AddTransientHttpErrorPolicy` para adicionar uma política de repetição. As solicitações com falha são repetidas até três vezes. A segunda chamada a `AddTransientHttpErrorPolicy` adiciona uma política de disjuntor. As solicitações externas adicionais são bloqueadas por 30 segundos quando ocorrem cinco tentativas com falha consecutivamente. As políticas de disjuntor são políticas com estado. Todas as chamadas por meio desse cliente compartilham o mesmo estado do circuito.

### Adicionar políticas do registro do Polly

Uma abordagem para gerenciar as políticas usadas com frequência é defini-las uma única vez e registrá-las em um `PolicyRegistry`. É fornecido um método de extensão que permite que um manipulador seja adicionado usando uma política do registro:

```
var registry = services.AddPolicyRegistry();

registry.Add("regular", timeout);
registry.Add("long", longTimeout);

services.AddHttpClient("regulartimeouthandler")
    .AddPolicyHandlerFromRegistry("regular");
```

No código anterior, duas políticas são registradas quando `PolicyRegistry` é adicionado ao `ServiceCollection`. Para usar uma política do registro, o método `AddPolicyHandlerFromRegistry` é usado, passando o nome da política a ser aplicada.

Mais informações sobre o `IHttpClientFactory` e as integrações do Polly podem ser encontradas no [Wiki do Polly](#).

## HttpClient e gerenciamento de tempo de vida

Uma nova instância de `HttpClient` é chamada, sempre que `CreateClient` é chamado na `IHttpClientFactory`. Há um `HttpMessageHandler` por cliente nomeado. O alocador gerencia a vida útil das instâncias do `HttpMessageHandler`.

`IHttpClientFactory` cria pools com as instâncias de `HttpMessageHandler` criadas pelo alocador para reduzir o consumo de recursos. Uma instância de `HttpMessageHandler` poderá ser reutilizada no pool, ao criar uma nova instância de `HttpClient`, se o respectivo tempo de vida não tiver expirado.

O pooling de manipuladores é preferível porque normalmente cada manipulador gerencia as próprias conexões HTTP subjacentes. Criar mais manipuladores do que o necessário pode resultar em atrasos de conexão. Alguns manipuladores também mantêm as conexões abertas indefinidamente, o que pode impedir que o manipulador reaja a alterações de DNS.

O tempo de vida padrão do manipulador é de 2 minutos. O valor padrão pode ser substituído para cada cliente nomeado. Para substituí-lo, chame `SetHandlerLifetime` no `IHttpClientBuilder` que é retornado ao criar o cliente:

```
services.AddHttpClient("extendedhandlerlifetime")
    .SetHandlerLifetime(TimeSpan.FromMinutes(5));
```

Não é necessário descartar o cliente. O descarte cancela as solicitações de saída e garante que a determinada instância de `HttpClient` não seja usada depois de chamar `Dispose`. `IHttpClientFactory` rastreia e descarta

recursos usados pelas instâncias de `HttpClient`. Geralmente, as instâncias de `HttpClient` podem ser tratadas como objetos do .NET e não exigem descarte.

Manter uma única instância de `HttpClient` ativa por uma longa duração é um padrão comum usado, antes do início de `IHttpClientFactory`. Esse padrão se torna desnecessário após a migração para `IHttpClientFactory`.

## Registrando em log

Os clientes criado pelo `IHttpClientFactory` registram mensagens de log para todas as solicitações. Habilite o nível apropriado de informações na configuração de log para ver as mensagens de log padrão. Os registros em log adicionais, como o registro em log dos cabeçalhos de solicitação, estão incluídos somente no nível de rastreamento.

A categoria de log usada para cada cliente inclui o nome do cliente. Um cliente chamado `MyNamedClient`, por exemplo, registra mensagens com uma categoria de `System.Net.Http.HttpClient.MyNamedClient.LogicalHandler`. Mensagens sufixadas com `LogicalHandler` ocorrem fora do pipeline do manipulador de solicitações. Na solicitação, as mensagens são registradas em log antes que qualquer outro manipulador no pipeline a tenha processado. Na resposta, as mensagens são registradas depois que qualquer outro manipulador do pipeline tenha recebido a resposta.

O registro em log também ocorre dentro do pipeline do manipulador de solicitações. No caso do exemplo de `MyNamedClient`, essas mensagens são registradas em log na categoria de log

`System.Net.Http.HttpClient.MyNamedClient.ClientHandler`. Para a solicitação, isso ocorre depois que todos os outros manipuladores são executados e imediatamente antes que a solicitação seja enviada pela rede. Na resposta, esse registro em log inclui o estado da resposta antes que ela seja retornada por meio do pipeline do manipulador.

Habilitar o registro em log dentro e fora do pipeline permite a inspeção das alterações feitas por outros manipuladores do pipeline. Isso pode incluir, por exemplo, alterações nos cabeçalhos de solicitação ou no código de status da resposta.

Incluir o nome do cliente na categoria de log permite a filtragem de log para clientes nomeados específicos, quando necessário.

## Configurar o `HttpMessageHandler`

Talvez seja necessário controlar a configuração do `HttpMessageHandler` interno usado por um cliente.

Um `IHttpClientBuilder` é retornado ao adicionar clientes nomeados ou com tipo. O método de extensão `ConfigurePrimaryHttpMessageHandler` pode ser usado para definir um representante. O representante que é usado para criar e configurar o `HttpMessageHandler` primário usado pelo cliente:

```
services.AddHttpClient("configured-inner-handler")
    .ConfigurePrimaryHttpMessageHandler(() =>
{
    return new HttpClientHandler()
    {
        AllowAutoRedirect = false,
        UseDefaultCredentials = true
    };
});
```

# Introdução a Páginas do Razor no ASP.NET Core

26/12/2018 • 34 minutes to read • [Edit Online](#)

Por [Rick Anderson](#) e [Ryan Nowak](#)

Páginas do Razor é um novo aspecto do ASP.NET Core MVC que torna a codificação de cenários focados em página mais fácil e produtiva.

Se você estiver procurando um tutorial que utiliza a abordagem Modelo-Exibição-Controlador, consulte a [Introdução ao ASP.NET Core MVC](#).

Este documento proporciona uma introdução a páginas do Razor. Este não é um tutorial passo a passo. Se você achar que algumas das seções são muito avançadas, consulte a [Introdução a Páginas do Razor](#). Para obter uma visão geral do ASP.NET Core, consulte a [Introdução ao ASP.NET Core](#).

## Pré-requisitos

### Pré-requisitos

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- [Visual Studio 2017 versão 15.9 ou posterior](#) com a carga de trabalho **ASP.NET e desenvolvimento para a Web**
- [SDK 2.2 ou posterior do .NET Core](#)

## Criar um projeto do Razor Pages

- [Visual Studio](#)
- [Visual Studio para Mac](#)
- [Visual Studio Code](#)

Confira a [Introdução ao Razor Pages](#) para obter instruções detalhadas sobre como criar um projeto do Razor Pages.

## Páginas do Razor

O Páginas do Razor está habilitado em `Startup.cs`:

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        // Includes support for Razor Pages and controllers.
        services.AddMvc();
    }

    public void Configure(IApplicationBuilder app)
    {
        app.UseMvc();
    }
}
```

Considere uma página básica:

```
@page

<h1>Hello, world!</h1>
<h2>The time on the server is @DateTime.Now</h2>
```

O código anterior é muito parecido com um arquivo de exibição do Razor. O que o torna diferentes é a diretiva `@page`. `@page` transforma o arquivo em uma ação do MVC – o que significa que ele trata solicitações diretamente, sem passar por um controlador. `@page` deve ser a primeira diretiva do Razor em uma página. `@page` afeta o comportamento de outros constructos do Razor.

Uma página semelhante, usando uma classe `PageModel`, é mostrada nos dois arquivos a seguir. O arquivo `Pages/Index2.cshtml`:

```
@page
@using RazorPagesIntro.Pages
@model IndexModel2

<h2>Separate page model</h2>
<p>
    @Model.Message
</p>
```

O modelo de página `Pages/Index2.cshtml.cs`:

```
using Microsoft.AspNetCore.Mvc.RazorPages;
using System;

namespace RazorPagesIntro.Pages
{
    public class IndexModel2 : PageModel
    {
        public string Message { get; private set; } = "PageModel in C#";

        public void OnGet()
        {
            Message += $" Server time is { DateTime.Now }";
        }
    }
}
```

Por convenção, o arquivo de classe `PageModel` tem o mesmo nome que o arquivo na Página do Razor com `.cs` acrescentado. Por exemplo, a Página do Razor anterior é `Pages/Index2.cshtml`. O arquivo que contém a classe `PageModel` é chamado `Pages/Index2.cshtml.cs`.

As associações de caminhos de URL para páginas são determinadas pelo local da página no sistema de arquivos. A tabela a seguir mostra um caminho de Página do Razor e a URL correspondente:

CAMINHO E NOME DO ARQUIVO	URL CORRESPONDENTE
/Pages/Index.cshtml	/ ou /Index
/Pages/Contact.cshtml	/Contact
/Pages/Store/Contact.cshtml	/Store/Contact
/Pages/Store/Index.cshtml	/Store ou /Store/Index

Notas:

- O tempo de execução procura arquivos de Páginas do Razor na pasta `Pages` por padrão.
- `Index` é a página padrão quando uma URL não inclui uma página.

## Escrever um formulário básico

Páginas do Razor foi projetado para facilitar a implementação de padrões comuns usados com navegadores da Web ao criar um aplicativo. [Model binding](#), [auxiliares de marcas](#) e auxiliares HTML *funcionam todos apenas* com as propriedades definidas em uma classe de Página do Razor. Considere uma página que implementa um formulário básico "Fale conosco" para o modelo `Contact`:

Para as amostras neste documento, o `DbContext` é inicializado no arquivo [Startup.cs](#).

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using RazorPagesContacts.Data;

namespace RazorPagesContacts
{
    public class Startup
    {
        public IHostingEnvironment HostingEnvironment { get; }

        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<AppDbContext>(options =>
                options.UseInMemoryDatabase("name"));
            services.AddMvc();
        }

        public void Configure(IApplicationBuilder app)
        {
            app.UseMvc();
        }
    }
}
```

O modelo de dados:

```
using System.ComponentModel.DataAnnotations;

namespace RazorPagesContacts.Data
{
    public class Customer
    {
        public int Id { get; set; }

        [Required, StringLength(100)]
        public string Name { get; set; }
    }
}
```

O contexto do banco de dados:

```
using Microsoft.EntityFrameworkCore;

namespace RazorPagesContacts.Data
{
    public class AppDbContext : DbContext
    {
        public AppDbContext(DbContextOptions options)
            : base(options)
        {
        }

        public DbSet<Customer> Customers { get; set; }
    }
}
```

O arquivo de exibição *Pages/Create.cshtml*:

```
@page
@model RazorPagesContacts.Pages.CreateModel
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

<html>
<body>
    <p>
        Enter your name.
    </p>
    <div asp-validation-summary="All"></div>
    <form method="POST">
        <div>Name: <input asp-for="Customer.Name" /></div>
        <input type="submit" />
    </form>
</body>
</html>
```

O modelo de página *Pages/Create.cshtml.cs*:

```

using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using RazorPagesContacts.Data;

namespace RazorPagesContacts.Pages
{
    public class CreateModel : PageModel
    {
        private readonly ApplicationDbContext _db;

        public CreateModel(ApplicationDbContext db)
        {
            _db = db;
        }

        [BindProperty]
        public Customer Customer { get; set; }

        public async Task<IActionResult> OnPostAsync()
        {
            if (!ModelState.IsValid)
            {
                return Page();
            }

            _db.Customers.Add(Customer);
            await _db.SaveChangesAsync();
            return RedirectToPage("/Index");
        }
    }
}

```

Por convenção, a classe `PageModel` é chamada de `<PageName>Model` e está no mesmo namespace que a página.

A classe `PageModel` permite separar a lógica de uma página da respectiva apresentação. Ela define manipuladores para as solicitações enviadas e os dados usados para renderizar a página. Esta separação permite gerenciar as dependências da página por meio de [injeção de dependência](#) e realizar um [teste de unidade](#) nas páginas.

A página tem um *método de manipulador* `OnPostAsync`, que é executado em solicitações `POST` (quando um usuário posta o formulário). Você pode adicionar métodos de manipulador para qualquer verbo HTTP. Os manipuladores mais comuns são:

- `OnGet` para inicializar o estado necessário para a página. Amostra de [OnGet](#).
- `OnPost` para manipular envios de formulário.

O sufixo de nomenclatura `Async` é opcional, mas geralmente é usado por convenção para funções assíncronas. O código `OnPostAsync` no exemplo anterior tem aparência semelhante ao que você normalmente escreve em um controlador. O código anterior é comum para as Páginas do Razor. A maioria dos primitivos MVC como [model binding](#), [validação](#) e resultados da ação são compartilhados.

O método `OnPostAsync` anterior:

```

public async Task<IActionResult> OnPostAsync()
{
    if (!ModelState.IsValid)
    {
        return Page();
    }

    _db.Customers.Add(Customer);
    await _db.SaveChangesAsync();
    return RedirectToPage("/Index");
}

```

O fluxo básico de `OnPostAsync` :

Verifique se há erros de validação.

- Se não houver nenhum erro, salve os dados e redirecione.
- Se houver erros, mostre a página novamente com as mensagens de validação. A validação do lado do cliente é idêntica para aplicativos ASP.NET Core MVC tradicionais. Em muitos casos, erros de validação seriam detectados no cliente e nunca enviados ao servidor.

Quando os dados são inseridos com êxito, o método de manipulador `OnPostAsync` chama o método auxiliar `RedirectToPage` para retornar uma instância de `RedirectPageResult`. `RedirectToPage` é um novo resultado de ação, semelhante a `RedirectToAction` ou `RedirectToRoute`, mas personalizado para páginas. Na amostra anterior, ele redireciona para a página de Índice raiz (`/Index`). `RedirectToPage` é descrito em detalhes na seção [Geração de URLs para páginas](#).

Quando o formulário enviado tem erros de validação (que são passados para o servidor), o método de manipulador `OnPostAsync` chama o método auxiliar `Page`. `Page` retorna uma instância de `PageResult`. Retornar `Page` é semelhante a como as ações em controladores retornam `View`. `PageResult` é o tipo de retorno padrão para um método de manipulador. Um método de manipulador que retorna `void` renderiza a página.

A propriedade `Customer` usa o atributo `[BindProperty]` para aceitar o model binding.

```

public class CreateModel : PageModel
{
    private readonly AppDbContext _db;

    public CreateModel(AppDbContext db)
    {
        _db = db;
    }

    [BindProperty]
    public Customer Customer { get; set; }

    public async Task<IActionResult> OnPostAsync()
    {
        if (!ModelState.IsValid)
        {
            return Page();
        }

        _db.Customers.Add(Customer);
        await _db.SaveChangesAsync();
        return RedirectToPage("/Index");
    }
}

```

Páginas do Razor, por padrão, associam as propriedades somente com verbos não GET. A associação de propriedades pode reduzir a quantidade de código que você precisa escrever. A associação reduz o código usando a mesma propriedade para renderizar os campos de formulário (

```
<input asp-for="Customer.Name" /> ) e aceitar a entrada.
```

#### WARNING

Por motivos de segurança, você deve aceitar associar os dados da solicitação `GET` às propriedades do modelo de página. Verifique a entrada do usuário antes de mapeá-la para as propriedades. Aceitar a associação de `GET` é útil ao lidar com cenários que contam com a cadeia de caracteres de consulta ou com os valores de rota.

Para associar uma propriedade às solicitações `GET`, defina a propriedade `SupportsGet` do atributo `[BindProperty]` como `true` : `[BindProperty(SupportsGet = true)]`

A home page (`/Index.cshtml`):

```
@page
@model RazorPagesContacts.Pages.IndexModel
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

<h1>Contacts</h1>
<form method="post">
    <table class="table">
        <thead>
            <tr>
                <th>ID</th>
                <th>Name</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var contact in Model.Customers)
            {
                <tr>
                    <td>@contact.Id</td>
                    <td>@contact.Name</td>
                    <td>
                        <a asp-page=".Edit" asp-route-id="@contact.Id">edit</a>
                        <button type="submit" asp-page-handler="delete"
                                asp-route-id="@contact.Id">delete</button>
                    </td>
                </tr>
            }
        </tbody>
    </table>

    <a asp-page=".Create">Create</a>
</form>
```

A classe `PageModel` (`/Index.cshtml.cs`) associada:

```

using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using RazorPagesContacts.Data;
using System.Collections.Generic;
using Microsoft.EntityFrameworkCore;

namespace RazorPagesContacts.Pages
{
    public class IndexModel : PageModel
    {
        private readonly AppDbContext _db;

        public IndexModel(AppDbContext db)
        {
            _db = db;
        }

        public IList<Customer> Customers { get; private set; }

        public async Task OnGetAsync()
        {
            Customers = await _db.Customers.AsNoTracking().ToListAsync();
        }

        public async Task<IActionResult> OnPostDeleteAsync(int id)
        {
            var contact = await _db.Customers.FindAsync(id);

            if (contact != null)
            {
                _db.Customers.Remove(contact);
                await _db.SaveChangesAsync();
            }

            return RedirectToPage();
        }
    }
}

```

O arquivo *cshtml* contém a marcação a seguir para criar um link de edição para cada contato:

```
<a asp-page="./Edit" asp-route-id="@contact.Id">edit</a>
```

O auxiliar de marcas de âncora usou o atributo `asp-route-{value}` para gerar um link para a página Edit. O link contém dados de rota com a ID de contato. Por exemplo, `http://localhost:5000/Edit/1`.

O arquivo *Pages/Edit.cshtml*:

```

@page "{id:int}"
@model RazorPagesContacts.Pages.EditModel
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

 @{
    ViewData["Title"] = "Edit Customer";
}

<h1>Edit Customer - @Model.Customer.Id</h1>
<form method="post">
    <div asp-validation-summary="All"></div>
    <input asp-for="Customer.Id" type="hidden" />
    <div>
        <label asp-for="Customer.Name"></label>
        <div>
            <input asp-for="Customer.Name" />
            <span asp-validation-for="Customer.Name" ></span>
        </div>
    </div>

    <div>
        <button type="submit">Save</button>
    </div>
</form>

```

A primeira linha contém a diretiva `@page "{id:int}"`. A restrição de roteamento `"{id:int}"` informa à página para aceitar solicitações para a página que contêm dados da rota `int`. Se uma solicitação para a página não contém dados de rota que podem ser convertidos em um `int`, o tempo de execução retorna um erro HTTP 404 (não encontrado). Para tornar a ID opcional, acrescente `?` à restrição de rota:

```
@page "{id:int?}"
```

O arquivo `Pages/Edit.cshtml.cs`:

```

using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using RazorPagesContacts.Data;

namespace RazorPagesContacts.Pages
{
    public class EditModel : PageModel
    {
        private readonly ApplicationDbContext _db;

        public EditModel(ApplicationDbContext db)
        {
            _db = db;
        }

        [BindProperty]
        public Customer Customer { get; set; }

        public async Task<IActionResult> OnGetAsync(int id)
        {
            Customer = await _db.Customers.FindAsync(id);

            if (Customer == null)
            {
                return RedirectToPage("/Index");
            }

            return Page();
        }

        public async Task<IActionResult> OnPostAsync()
        {
            if (!ModelState.IsValid)
            {
                return Page();
            }

            _db.Attach(Customer).State = EntityState.Modified;

            try
            {
                await _db.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                throw new Exception($"Customer {Customer.Id} not found!");
            }

            return RedirectToPage("/Index");
        }
    }
}

```

O arquivo *Index.cshtml* também contém a marcação para criar um botão de exclusão para cada contato de cliente:

```

<button type="submit" asp-page-handler="delete"
asp-route-id="@contact.Id">delete</button>

```

Quando o botão de exclusão é renderizado em HTML, seu `formaction` inclui parâmetros para:

- A ID de contato do cliente especificada pelo atributo `asp-route-id`.
- O `handler` especificado pelo atributo `asp-page-handler`.

Este é um exemplo de um botão de exclusão renderizado com uma ID de contato do cliente de `1`:

```
<button type="submit" formaction="/?id=1&handler=delete">delete</button>
```

Quando o botão é selecionado, uma solicitação de formulário `POST` é enviada para o servidor. Por convenção, o nome do método do manipulador é selecionado com base no valor do parâmetro `handler` de acordo com o esquema `OnPost[handler]Async`.

Como o `handler` é `delete` neste exemplo, o método do manipulador `OnPostDeleteAsync` é usado para processar a solicitação `POST`. Se o `asp-page-handler` for definido como um valor diferente, como `remove`, um método de manipulador de página com o nome `OnPostRemoveAsync` será selecionado.

```
public async Task<IActionResult> OnPostDeleteAsync(int id)
{
    var contact = await _db.Customers.FindAsync(id);

    if (contact != null)
    {
        _db.Customers.Remove(contact);
        await _db.SaveChangesAsync();
    }

    return RedirectToPage();
}
```

O método `OnPostDeleteAsync`:

- Aceita o `id` da cadeia de caracteres de consulta.
- Consulta o banco de dados para o contato de cliente com `FindAsync`.
- Se o contato do cliente for encontrado, eles serão removidos da lista de contatos do cliente. O banco de dados é atualizado.
- Chama `RedirectToPage` para redirecionar para a página de índice de raiz (`/Index`).

## Marque as propriedades da página conforme necessário

As propriedades em um `PageModel` podem ser decoradas com o atributo `Required`:

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using System.ComponentModel.DataAnnotations;

namespace RazorPagesMovie.Pages.Movies
{
    public class CreateModel : PageModel
    {
        public IActionResult OnGet()
        {
            return Page();
        }

        [BindProperty]
        [Required(ErrorMessage = "Color is required")]
        public string Color { get; set; }

        public IActionResult OnPostAsync()
        {
            if (!ModelState.IsValid)
            {
                return Page();
            }

            // Process color.

            return RedirectToPage("./Index");
        }
    }
}

```

Para obter mais informações, confira [Validação de modelo](#).

## Gerenciar solicitações HEAD com o manipulador OnGet

As solicitações HEAD permitem recuperar os cabeçalhos de um recurso específico. Ao contrário das solicitações GET, as solicitações HEAD não retornam um corpo de resposta.

Geralmente, um manipulador HEAD é criado e chamado para solicitações HEAD:

```

public void OnHead()
{
    HttpContext.Response.Headers.Add("HandledBy", "Handled by OnHead!");
}

```

Caso nenhum manipulador HEAD (`OnHead`) seja definido, as Páginas Razor voltam a chamar o manipulador de página GET (`OnGet`) no ASP.NET Core 2.1 ou posterior. No ASP.NET Core 2.1 e 2.2, esse comportamento ocorre com o `SetCompatibilityVersion` em `Startup.Configure`:

```

services.AddMvc()
    .SetCompatibilityVersion(Microsoft.AspNetCore.Mvc.CompatibilityVersion.Version_2_1);

```

Os modelos padrão geram a chamada `SetCompatibilityVersion` no ASP.NET Core 2.1 e 2.2.

`SetCompatibilityVersion` define de forma eficiente a opção de Páginas Razor `AllowMappingHeadRequestsToGetHandler` como `true`.

Em vez de aceitar todos os 2.1 comportamentos com `SetCompatibilityVersion`, você pode explicitamente participar de comportamentos específicos. O código a seguir aceita as solicitações de mapeamento HEAD

para o manipulador GET.

```
services.AddMvc()
    .AddRazorPagesOptions(options =>
{
    options.AllowMappingHeadRequestsToGetHandler = true;
});
```

## XSRF/CSRF e Páginas do Razor

Você não precisa escrever nenhum código para [validação antifalsificação](#). Validação e geração de token antifalsificação são automaticamente incluídas nas Páginas do Razor.

## Usando Layouts, parciais, modelos e auxiliares de marcas com Páginas do Razor

As Páginas funcionam com todos os recursos do mecanismo de exibição do Razor. Layouts, parciais, modelos, auxiliares de marcas, `_ViewStart.cshtml` e `_ViewImports.cshtml` funcionam da mesma forma que funcionam exibições convencionais do Razor.

Organizaremos essa página aproveitando alguns desses recursos.

Adicione uma [página de layout](#) a `Pages/Shared/_Layout.cshtml`:

Adicione uma [página de layout](#) a `Pages/_Layout.cshtml`:

```
<!DOCTYPE html>
<html>
<head>
    <title>Razor Pages Sample</title>
</head>
<body>
    <a asp-page="/Index">Home</a>
    @RenderBody()
    <a asp-page="/Customers/Create">Create</a> <br />
</body>
</html>
```

O [Layout](#):

- Controla o layout de cada página (a menos que a página opte por não usar o layout).
- Importa estruturas HTML como JavaScript e folhas de estilo.

Veja [página de layout](#) para obter mais informações.

A propriedade [Layout](#) é definida em `Pages/_ViewStart.cshtml`:

```
@{
    Layout = "_Layout";
}
```

O layout está na pasta `Pages/Shared`. As páginas buscam outras exibições (layouts, modelos, parciais) hierarquicamente, iniciando na mesma pasta que a página atual. Um layout na pasta `Pages/Shared` pode ser usado em qualquer página do Razor na pasta `Pages`.

O arquivo de layout deve entrar na pasta `Pages/Shared`.

O layout está na pasta `Pages`. As páginas buscam outras exibições (layouts, modelos, parciais)

hierarquicamente, iniciando na mesma pasta que a página atual. Um layout na pasta *Pages* pode ser usado em qualquer Página do Razor na pasta *Pages*.

Recomendamos que você **não** coloque o arquivo de layout na pasta *Views/Shared*. *Views/Shared* é um padrão de exibições do MVC. As Páginas do Razor devem confiar na hierarquia de pasta e não nas convenções de caminho.

A pesquisa de modo de exibição de uma Página do Razor inclui a pasta *Pages*. Os layouts, modelos e parciais que você está usando com controladores MVC e exibições do Razor convencionais *apenas funcionam*.

Adicione um arquivo *Pages/\_ViewImports.cshtml*:

```
@namespace RazorPagesContacts.Pages  
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

`@namespace` é explicado posteriormente no tutorial. A diretiva `@addTagHelper` coloca os [auxiliares de marcas internos](#) em todas as páginas na pasta *Pages*.

Quando a diretiva `@namespace` é usada explicitamente em uma página:

```
@page  
@namespace RazorPagesIntro.Pages.Customers  
  
@model NameSpaceModel  
  
<h2>Name space</h2>  
<p>  
    @Model.Message  
</p>
```

A diretiva define o namespace da página. A diretiva `@model` não precisa incluir o namespace.

Quando a diretiva `@namespace` está contida em *\_ViewImports.cshtml*, o namespace especificado fornece o prefixo do namespace gerado na página que importa a diretiva `@namespace`. O restante do namespace gerado (a parte do sufixo) é o caminho relativo separado por ponto entre a pasta que contém *\_ViewImports.cshtml* e a pasta que contém a página.

Por exemplo, a classe `PageModel` *Pages/Customers/Edit.cshtml.cs* define explicitamente o namespace:

```
namespace RazorPagesContacts.Pages  
{  
    public class EditModel : PageModel  
    {  
        private readonly AppDbContext _db;  
  
        public EditModel(AppDbContext db)  
        {  
            _db = db;  
        }  
  
        // Code removed for brevity.  
    }  
}
```

O arquivo *Pages/\_ViewImports.cshtml* define o namespace a seguir:

```
@namespace RazorPagesContacts.Pages  
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

O namespace gerado para o Razor Pages *Pages/Customers/Edit.cshtml* é o mesmo que a classe `PageModel`.

`@namespace` também funciona com exibições do Razor convencionais.

O arquivo de exibição *Pages/Create.cshtml* original:

```
@page
@model RazorPagesContacts.Pages.CreateModel
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

<html>
<body>
    <p>
        Enter your name.
    </p>
    <div asp-validation-summary="All"></div>
    <form method="POST">
        <div>Name: <input asp-for="Customer.Name" /></div>
        <input type="submit" />
    </form>
</body>
</html>
```

O arquivo de exibição *Pages/Create.cshtml* atualizado:

```
@page
@model CreateModel

<html>
<body>
    <p>
        Enter your name.
    </p>
    <div asp-validation-summary="All"></div>
    <form method="POST">
        <div>Name: <input asp-for="Customer.Name" /></div>
        <input type="submit" />
    </form>
</body>
</html>
```

O projeto inicial de Páginas do Razor contém o *Pages/\_ValidationScriptsPartial.cshtml*, que conecta a validação do lado do cliente.

Para obter mais informações sobre exibições parciais, consulte [Exibições parciais no ASP.NET Core](#).

## Geração de URL para Páginas

A página `Create`, exibida anteriormente, usa `RedirectToPage`:

```
public async Task<IActionResult> OnPostAsync()
{
    if (!ModelState.IsValid)
    {
        return Page();
    }

    _db.Customers.Add(Customer);
    await _db.SaveChangesAsync();
    return RedirectToPage("/Index");
}
```

O aplicativo tem a estrutura de arquivos/pastas a seguir:

- `/Pages`
  - `Index.cshtml`
  - `/Clientes`
    - `Create.cshtml`
    - `Edit.cshtml`
    - `Index.cshtml`

As páginas `Pages/Customers/Create.cshtml` e `Pages/Customers/Edit.cshtml` redirecionam para o `Pages/Index.cshtml` após êxito. A cadeia de caracteres `/Index` faz parte do URI para acessar a página anterior. A cadeia de caracteres `/Index` pode ser usada para gerar URLs para a página `Pages/Index.cshtml`. Por exemplo:

- `Url.Page("/Index", ...)`
- `<a asp-page="/Index">My Index Page</a>`
- `RedirectToPage("/Index")`

O nome da página é o caminho para a página da pasta raiz `/Pages`, incluindo um `/` à direita (por exemplo, `/Index`). Os exemplos anteriores de geração de URL oferecem opções avançadas e recursos funcionais para codificar uma URL. A geração de URL usa [roteamento](#) e pode gerar e codificar parâmetros de acordo com o modo como a rota é definida no caminho de destino.

A Geração de URL para páginas dá suporte a nomes relativos. A tabela a seguir mostra qual página de Índice é selecionada com diferentes parâmetros `RedirectToPage` de `Pages/Customers/Create.cshtml`:

REDIRECTTOPAGE(X)	PÁGINA
<code>RedirectToPage("/Index")</code>	<code>Pages/Index</code>
<code>RedirectToPage("./Index");</code>	<code>Pages/Customers/Index</code>
<code>RedirectToPage("../Index")</code>	<code>Pages/Index</code>
<code>RedirectToPage("Index")</code>	<code>Pages/Customers/Index</code>

`RedirectToPage("Index")` , `RedirectToPage("./Index")` e `RedirectToPage("../Index")` são *nomes relativos*. O parâmetro `RedirectToPage` é combinado com o caminho da página atual para calcular o nome da página de destino.

Vinculação de nome relativo é útil ao criar sites com uma estrutura complexa. Se você usar nomes relativos para vincular entre páginas em uma pasta, você poderá renomear essa pasta. Todos os links ainda funcionarão (porque eles não incluirão o nome da pasta).

## Atributo ViewData

Os dados podem ser passados para uma página com [ViewDataAttribute](#). As propriedades nos controladores ou nos modelos da Página Razor decoradas com `[ViewData]` têm seus valores armazenados e carregados em [ViewDataDictionary](#).

No exemplo a seguir, o `AboutModel` contém uma propriedade `Title` decorada com `[ViewData]`. A propriedade `Title` está definida como o título da página Sobre:

```
public class AboutModel : PageModel
{
    [ViewData]
    public string Title { get; } = "About";

    public void OnGet()
    {
    }
}
```

Na página Sobre, accesse a propriedade `Title` como uma propriedade de modelo:

```
<h1>@Model.Title</h1>
```

No layout, o título é lido a partir do dicionário ViewData:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>@ViewData["Title"] - WebApplication</title>
    ...

```

## TempData

O ASP.NET Core expõe a propriedade `TempData` em um [controlador](#). Essa propriedade armazena dados até eles serem lidos. Os métodos `Keep` e `Peek` podem ser usados para examinar os dados sem exclusão.

`TempData` é útil para redirecionamento nos casos em que os dados são necessários para mais de uma única solicitação.

O atributo `[TempData]` é novo no ASP.NET Core 2.0 e tem suporte em controladores e páginas.

Os conjuntos de código a seguir definem o valor de `Message` usando  `TempData`:

```
public class CreateDotModel : PageModel
{
    private readonly AppDbContext _db;

    public CreateDotModel(AppDbContext db)
    {
        _db = db;
    }

    [TempData]
    public string Message { get; set; }

    [BindProperty]
    public Customer Customer { get; set; }

    public async Task<IActionResult> OnPostAsync()
    {
        if (!ModelState.IsValid)
        {
            return Page();
        }

        _db.Customers.Add(Customer);
        await _db.SaveChangesAsync();
        Message = $"Customer {Customer.Name} added";
        return RedirectToPage("./Index");
    }
}
```

A marcação a seguir no arquivo *Pages/Customers/Index.cshtml* exibe o valor de `Message` usando `[TempData]`

```
<h3>Msg: @Model.Message</h3>
```

O modelo de página *Pages/Customers/Index.cshtml.cs* aplica o atributo `[TempData]` à propriedade `Message`.

```
[TempData]
public string Message { get; set; }
```

Para obter mais informações, confira  [TempData](#).

## Vários manipuladores por página

A página a seguir gera marcação para dois manipuladores de página usando o auxiliar de marcas `asp-page-handler`:

```
@page
@model CreateFATHModel

<html>
<body>
    <p>
        Enter your name.
    </p>
    <div asp-validation-summary="All"></div>
    <form method="POST">
        <div>Name: <input asp-for="Customer.Name" /></div>
        <input type="submit" asp-page-handler="JoinList" value="Join" />
        <input type="submit" asp-page-handler="JoinListUC" value="JOIN UC" />
    </form>
</body>
</html>
```

O formulário no exemplo anterior tem dois botões de envio, cada um usando o `FormActionTagHelper` para enviar para uma URL diferente. O atributo `asp-page-handler` é um complemento para `asp-page`.

`asp-page-handler` gera URLs que enviam para cada um dos métodos de manipulador definidos por uma página. `asp-page` não foi especificado porque a amostra está vinculando à página atual.

O modelo de página:

```

using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using RazorPagesContacts.Data;

namespace RazorPagesContacts.Pages.Customers
{
    public class CreateFATHModel : PageModel
    {
        private readonly ApplicationDbContext _db;

        public CreateFATHModel(ApplicationDbContext db)
        {
            _db = db;
        }

        [BindProperty]
        public Customer Customer { get; set; }

        public async Task<IActionResult> OnPostJoinListAsync()
        {
            if (!ModelState.IsValid)
            {
                return Page();
            }

            _db.Customers.Add(Customer);
            await _db.SaveChangesAsync();
            return RedirectToPage("/Index");
        }

        public async Task<IActionResult> OnPostJoinListUCAsync()
        {
            if (!ModelState.IsValid)
            {
                return Page();
            }
            Customer.Name = Customer.Name?.ToUpper();
            return await OnPostJoinListAsync();
        }
    }
}

```

O código anterior usa *métodos de manipulador nomeados*. Métodos de manipulador nomeados são criados colocando o texto no nome após `On<HTTP Verb>` e antes de `Async` (se houver). No exemplo anterior, os métodos de página são `OnPostJoinListAsync` e `OnPostJoinListUCAsync`. Com `OnPost` e `Async` removidos, os nomes de manipulador são `JoinList` e `JoinListUC`.

```

<input type="submit" asp-page-handler="JoinList" value="Join" />
<input type="submit" asp-page-handler="JoinListUC" value="JOIN UC" />

```

Usando o código anterior, o caminho da URL que envia a `OnPostJoinListAsync` é `http://localhost:5000/Customers/CreateFATH?handler=JoinList`. O caminho da URL que envia a `OnPostJoinListUCAsync` é `http://localhost:5000/Customers/CreateFATH?handler=JoinListUC`.

## Rotas personalizadas

Use a diretiva `@page` para:

- Especifique uma rota personalizada para uma página. Por exemplo, a rota para a página Sobre pode ser definida como `/Some/Other/Path` com `@page "/Some/Other/Path"`.

- Acrescente segmentos à rota padrão de uma página. Por exemplo, um segmento de "item" pode ser adicionado à rota padrão da página com `@page "item"`.
- Acrescente parâmetros à rota padrão de uma página. Por exemplo, um parâmetro de ID, `id`, pode ser necessário para uma página com `@page "{id}"`.

Há suporte para um caminho relativo à raiz designado por um til (`~`) no início do caminho. Por exemplo, `@page "~/Some/Other/Path"` é o mesmo que `@page "/Some/Other/Path"`.

Você pode alterar a cadeia de caracteres de consulta `?handler=JoinList` na URL para um segmento de rota `/JoinList` ao especificar o modelo de rota `@page "{handler?}"`.

Se você não deseja a cadeia de consulta `?handler=JoinList` na URL, você pode alterar a rota para colocar o nome do manipulador na parte do caminho da URL. Você pode personalizar a rota adicionando um modelo de rota entre aspas duplas após a diretiva `@page`.

```
@page "{handler?}"
@model CreateRouteModel

<html>
<body>
    <p>
        Enter your name.
    </p>
    <div asp-validation-summary="All"></div>
    <form method="POST">
        <div>Name: <input asp-for="Customer.Name" /></div>
        <input type="submit" asp-page-handler="JoinList" value="Join" />
        <input type="submit" asp-page-handler="JoinListUC" value="JOIN UC" />
    </form>
</body>
</html>
```

Usando o código anterior, o caminho da URL que envia a `OnPostJoinListAsync` é `http://localhost:5000/Customers/CreateFATH/JoinList`. O caminho da URL que envia a `OnPostJoinListUCAsync` é `http://localhost:5000/Customers/CreateFATH/JoinListUC`.

O `?` após `handler` significa que o parâmetro de rota é opcional.

## Configuração e definições

Para configurar opções avançadas, use o método de extensão `AddRazorPagesOptions` no construtor de MVC:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc()
        .AddRazorPagesOptions(options =>
    {
        options.RootDirectory = "/MyPages";
        options.Conventions.AuthorizeFolder("/MyPages/Admin");
    });
}
```

No momento, você pode usar o `RazorPagesOptions` para definir o diretório raiz para páginas ou adicionar as convenções de modelo de aplicativo para páginas. Permitiremos mais extensibilidade dessa maneira no futuro.

Para pré-compilar exibições, consulte [Compilação de exibição do Razor](#).

[Baixar ou exibir código de exemplo](#).

Consulte a [Introdução a Páginas do Razor](#), que se baseia nesta introdução.

### Especificar que as Páginas Razor estão na raiz do conteúdo

Por padrão, as Páginas Razor estão na raiz do diretório `/Pages`. Adicione [WithRazorPagesAtContentRoot](#) em [AddMvc](#) para especificar que as Páginas Razor estão na raiz do conteúdo ([ContentRootPath](#)) do aplicativo:

```
services.AddMvc()
    .AddRazorPagesOptions(options =>
{
    ...
})
    .WithRazorPagesAtContentRoot();
```

### Especificar que as Páginas Razor estão em um diretório raiz personalizado

Adicione [WithRazorPagesRoot](#) em [AddMvc](#) para especificar que as Páginas Razor estão em um diretório raiz personalizado no aplicativo (forneça um caminho relativo):

```
services.AddMvc()
    .AddRazorPagesOptions(options =>
{
    ...
})
    .WithRazorPagesRoot("/path/to/razor/pages");
```

## Recursos adicionais

- [Introdução ao ASP.NET Core](#)
- [Referência da sintaxe Razor para ASP.NET Core](#)
- [Tutorial: Introdução às Páginas do Razor no ASP.NET Core](#)
- [Convenções de autorização de páginas do Razor no ASP.NET Core](#)
- [Convenções de rota e aplicativo das Páginas do Razor no ASP.NET Core](#)
- [Testes de unidade de páginas do Razor no ASP.NET Core](#)
- [Exibições parciais no ASP.NET Core](#)

# Tutorial: Criar um aplicativo Web de Páginas do Razor com o ASP.NET Core

04/02/2019 • 2 minutes to read • [Edit Online](#)

Esta série de tutoriais explica as noções básicas sobre a criação de um aplicativo Web Razor Pages.

Para obter uma introdução mais avançada direcionada a desenvolvedores experientes, confira [Introdução a Razor Pages](#).

Esta série inclui os seguintes tutoriais:

1. [Criar um aplicativo Web de Páginas do Razor](#)
2. [Adicionar um modelo a um aplicativo de Páginas do Razor](#)
3. [Gerar páginas do Razor por scaffolding](#)
4. [Trabalhar com um banco de dados](#)
5. [Atualizar páginas do Razor](#)
6. [Adicionar pesquisa](#)
7. [Adicionar um novo campo](#)
8. [Adicionar validação](#)

No final, você terá um aplicativo que pode exibir e gerenciar um banco de dados de filmes.

The screenshot shows a browser window with the title "Index - Movie". The address bar contains the URL "https://localhost:5001/Movies?SearchString=ghost". The page content includes a navigation bar with links for "RpMovie", "Home", and "Privacy". Below this is a section titled "Index" with a "Create New" link. A search/filter input field contains the value "ghost", with a "Filter" button next to it. A table lists movie data with the following rows:

Title	Release Date	Genre	Price	
Ghostbusters	3/13/1984	Comedy	8.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Ghostbusters 2	2/23/1986	Comedy	9.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

At the bottom of the page, there is a copyright notice: "© 2019 - RazorPagesMovie - Privacy".

# Tutorial: Introdução às Páginas do Razor no ASP.NET Core

04/02/2019 • 8 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

Este é o primeiro tutorial de uma série. [A série](#) ensina as noções básicas de criação de um aplicativo Web do Razor Pages do ASP.NET Core.

Para obter uma introdução mais avançada direcionada a desenvolvedores experientes, confira [Introdução a Razor Pages](#).

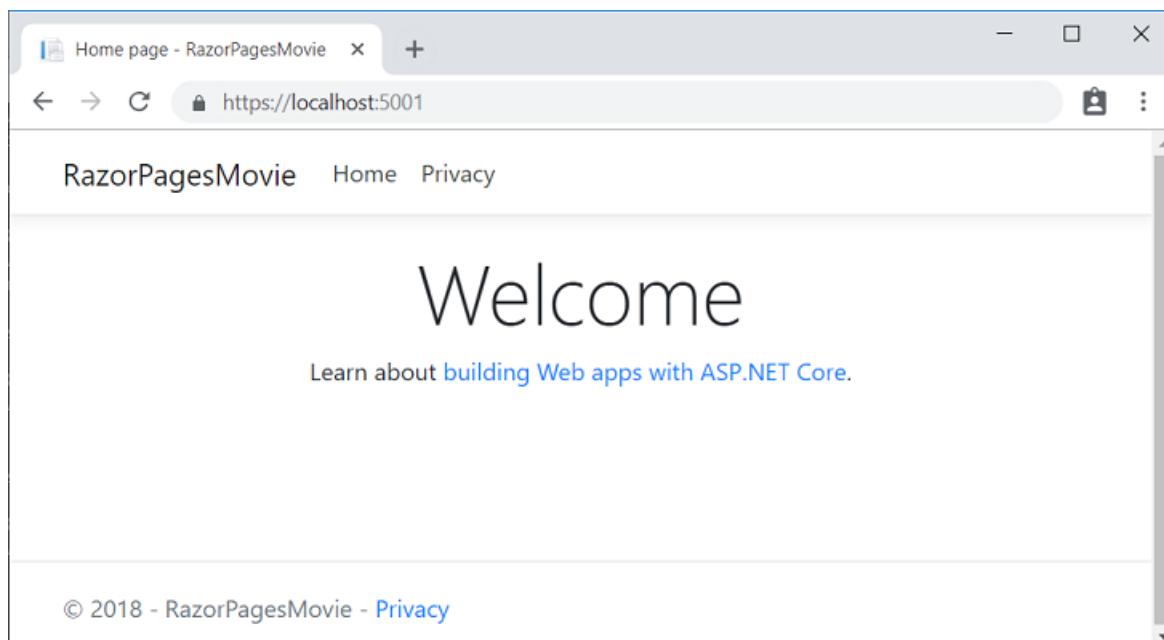
No final da série, você terá um aplicativo que gerencia um banco de dados de filmes.

[View or download sample code \(how to download\)](#).

Neste tutorial, você:

- Criar um aplicativo Web do Razor Pages.
- Execute o aplicativo.
- Examinar os arquivos de projeto.

No final deste tutorial, você terá um aplicativo Web em funcionamento do Razor Pages que você criará em tutoriais posteriores.

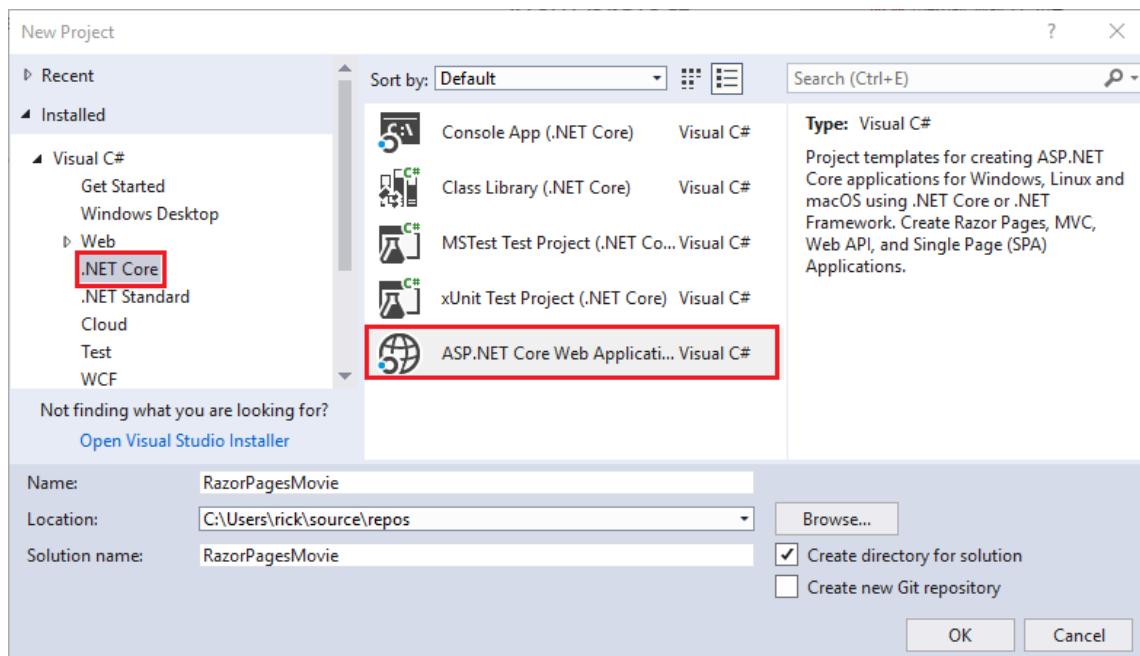


## Pré-requisitos

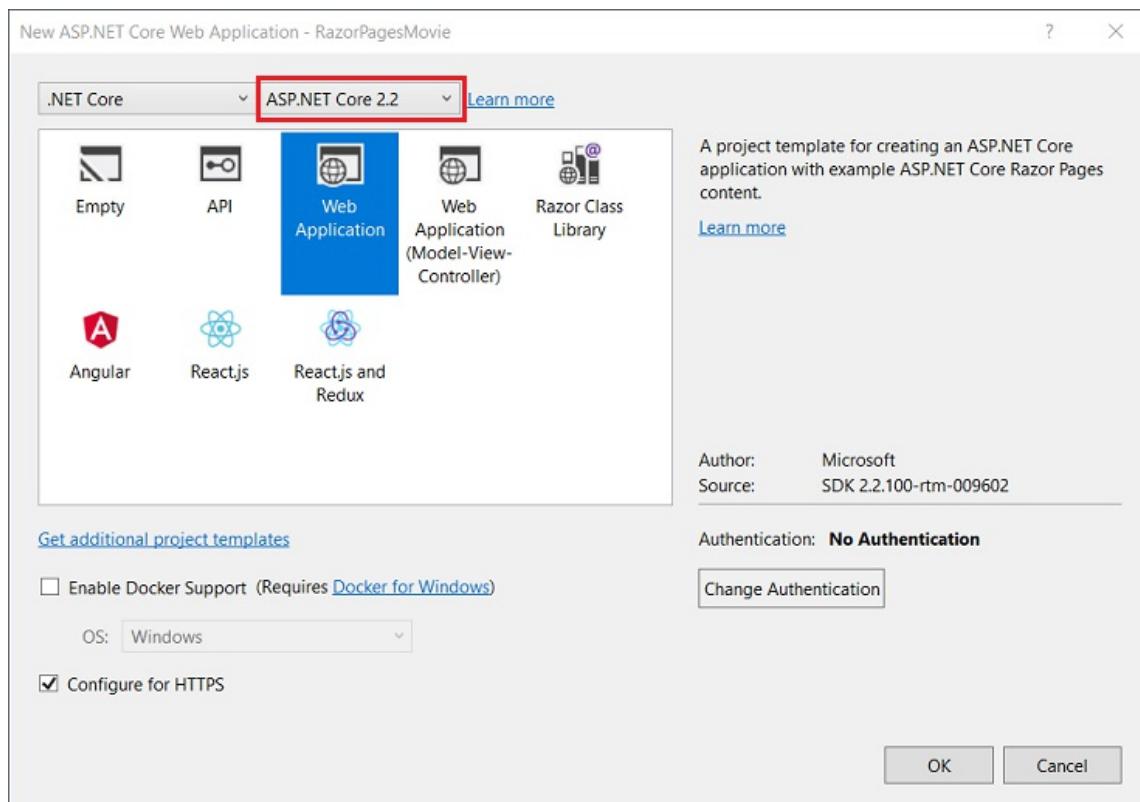
- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- [Visual Studio 2017 versão 15.9 ou posterior](#) com a carga de trabalho **ASP.NET e desenvolvimento para a Web**
- [SDK 2.2 ou posterior do .NET Core](#)

# Criar um aplicativo Web das Páginas do Razor

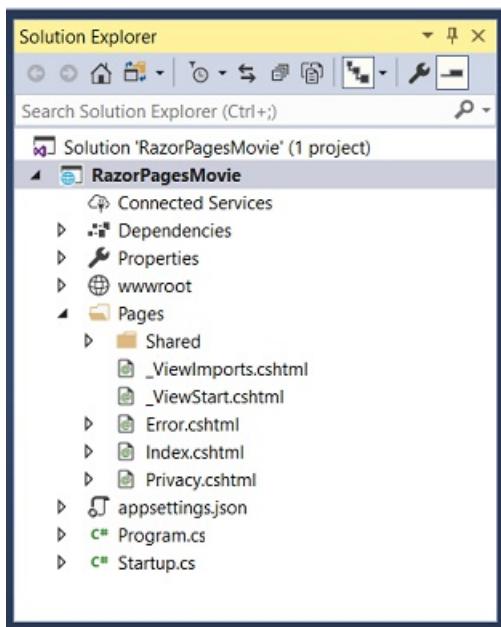
- Visual Studio
  - Visual Studio Code
  - Visual Studio para Mac
- No menu **Arquivo** do Visual Studio, selecione **Novo > Projeto**.
  - Crie um novo Aplicativo Web ASP.NET Core. Nomeie o projeto **RazorPagesMovie**. É importante nomear o projeto *RazorPagesMovie* de modo que os namespaces façam a correspondência quando você copiar e colar o código.



- Selecione **ASP.NET Core 2.2** na lista suspensa e, em seguida, selecione **Aplicativo Web**.



O seguinte projeto inicial é criado:



## Executar o aplicativo Web

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- Pressione Ctrl + F5 para execução sem o depurador.

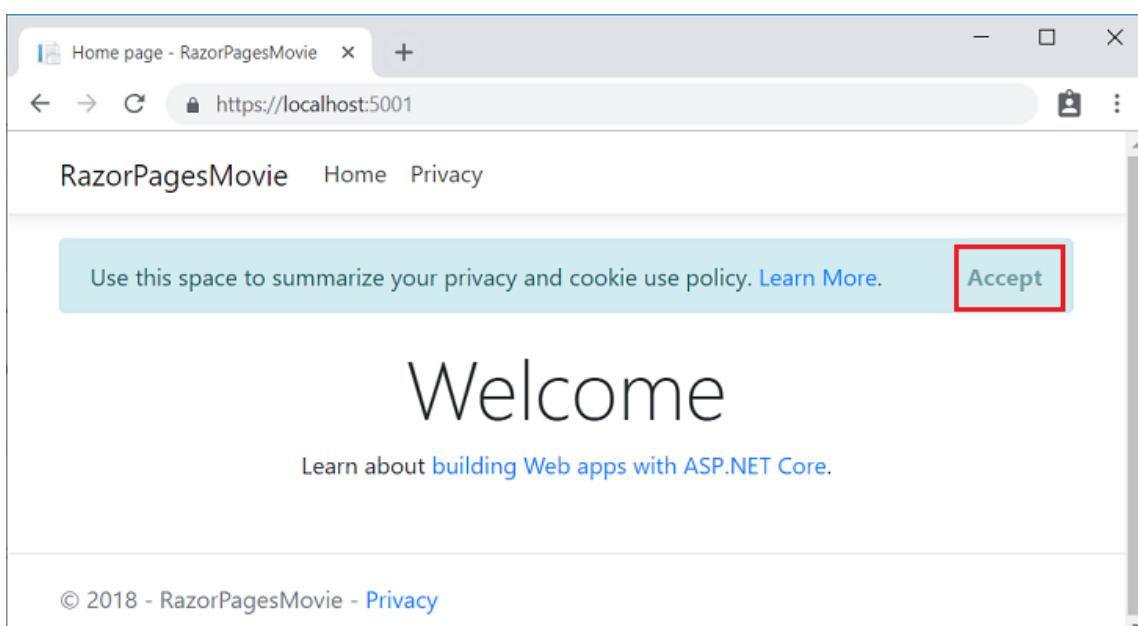
O Visual Studio inicia o [IIS Express](#) e executa o aplicativo. A barra de endereços mostra

`localhost:port#` e não algo como `example.com`. Isso ocorre porque `localhost` é o nome do host padrão do computador local. Localhost serve somente solicitações da Web do computador local.

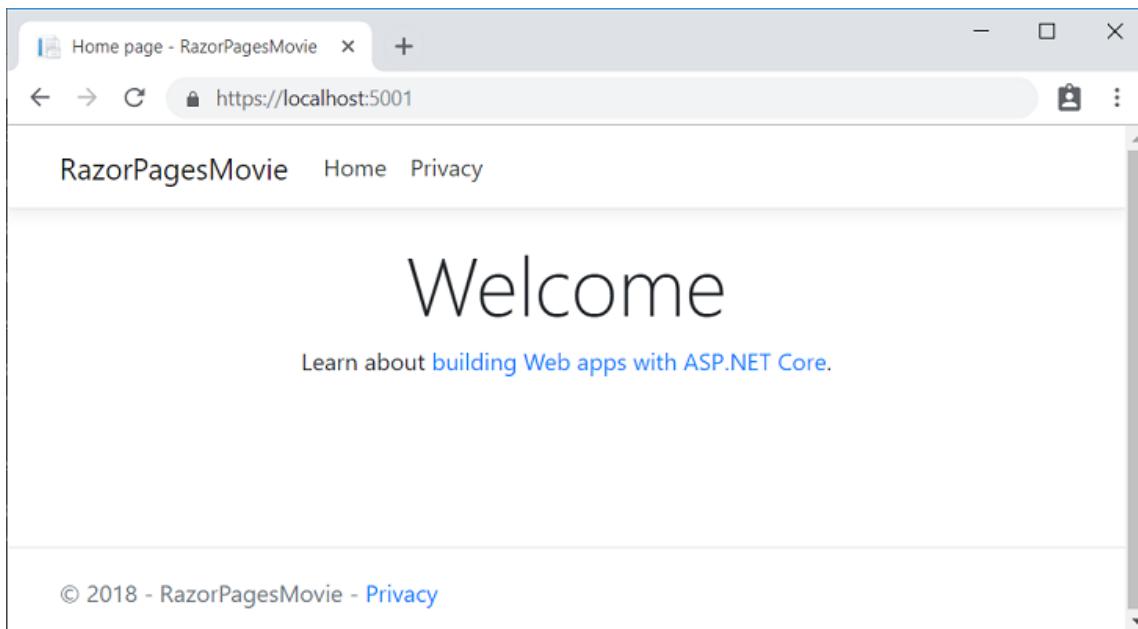
Quando o Visual Studio cria um projeto Web, uma porta aleatória é usada para o servidor Web. Na imagem anterior, o número da porta é 5001. Quando você executar o aplicativo, verá um número de porta diferente.

- Na página inicial do aplicativo, selecione **Aceitar** para dar consentimento de acompanhamento.

Este aplicativo não rastreia informações pessoais, mas o modelo de projeto inclui o recurso de consentimento no caso de você precisar estar em conformidade com o [RGPD \(Regulamento Geral sobre a Proteção de Dados\)](#) da União Europeia.



A imagem a seguir mostra o aplicativo depois de consentir o acompanhamento:



## Examinar os arquivos de projeto

Aqui está uma visão geral das pastas do projeto principal e os arquivos que você usará para trabalhar em tutoriais posteriores.

### Pasta Páginas

Contém as Razor Pages e os arquivos de suporte. Cada Razor Page é um par de arquivos:

- Um arquivo `.cshtml` que contém a marcação HTML com o código C# usando a sintaxe Razor.
- Um arquivo `.cshtml.cs` que contém o código C# que manipula os eventos de página.

Arquivos de suporte têm nomes que começam com um sublinhado. Por exemplo, o arquivo `_Layout.cshtml` configura os elementos de interface do usuário comuns a todas as páginas. Esse arquivo define o menu de navegação na parte superior da página e a notificação de direitos autorais na parte inferior da página. Para obter mais informações, consulte [Layout no ASP.NET Core](#).

### Pasta wwwroot

Contém os arquivos estáticos, como arquivos HTML, arquivos JavaScript e arquivos CSS. Para obter mais informações, consulte [Arquivos estáticos no ASP.NET Core](#).

### appSettings.json

Contém dados de configuração, como cadeias de conexão. Para obter mais informações, consulte [Configuração no ASP.NET Core](#).

### Module.vb

Contém o ponto de entrada para o programa. Para obter mais informações, consulte [Host da Web do ASP.NET Core](#).

### Startup.cs

Contém o código que configura o comportamento do aplicativo, como se ele requer o consentimento para cookies. Para obter mais informações, consulte [Inicialização de aplicativo no ASP.NET Core](#).

## Próximas etapas

Neste tutorial, você:

- Criou um aplicativo Web do Razor Pages.
- Executou o aplicativo.
- Examinou os arquivos de projeto.

Vá para o próximo tutorial da série:

ADICIONAR UM  
MÓDULO

# Adicionar um modelo a um aplicativo Páginas Razor no ASP.NET Core

28/01/2019 • 21 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

[View or download sample code \(how to download\)](#).

Nesta seção, classes são adicionadas para o gerenciamento de filmes em um banco de dados. Essas classes são usadas com o [EF Core](#) (Entity Framework Core) para trabalhar com um banco de dados. O EF Core é uma estrutura ORM (mapeamento relacional de objetos) que simplifica o código de acesso a dados.

As classes de modelo são conhecidas como classes POCO (de "objetos CLR básicos") porque não têm nenhuma dependência do EF Core. Elas definem as propriedades dos dados que são armazenados no banco de dados.

[Exiba ou baixe](#) a amostra.

## Adicionar um modelo de dados

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)

Clique com o botão direito do mouse no projeto **RazorPagesMovie** > **Adicionar** > **Nova Pasta**. Nomeie a pasta *Models*.

Clique com o botão direito do mouse na pasta *Modelos*. Selecione **Adicionar** > **Classe**. Dê à classe o nome **Movie**.

Adicione as seguintes propriedades à classe `Movie`:

```
using System;
using System.ComponentModel.DataAnnotations;

namespace RazorPagesMovie.Models
{
    public class Movie
    {
        public int ID { get; set; }
        public string Title { get; set; }

        [DataType(DataType.Date)]
        public DateTime ReleaseDate { get; set; }
        public string Genre { get; set; }
        public decimal Price { get; set; }
    }
}
```

A classe `Movie` contém:

- O campo `ID` é necessário para o banco de dados para a chave primária.
- `[DataType(DataType.Date)]`: O atributo [DataType](#) especifica o tipo de dados (Data). Com esse atributo:
  - O usuário não precisa inserir informações de tempo no campo de data.

- Somente a data é exibida, não as informações de tempo.

[DataAnnotations](#) são abordados em um tutorial posterior.

Crie o projeto para verificar se não há erros de compilação.

## Fazer scaffold do modelo de filme

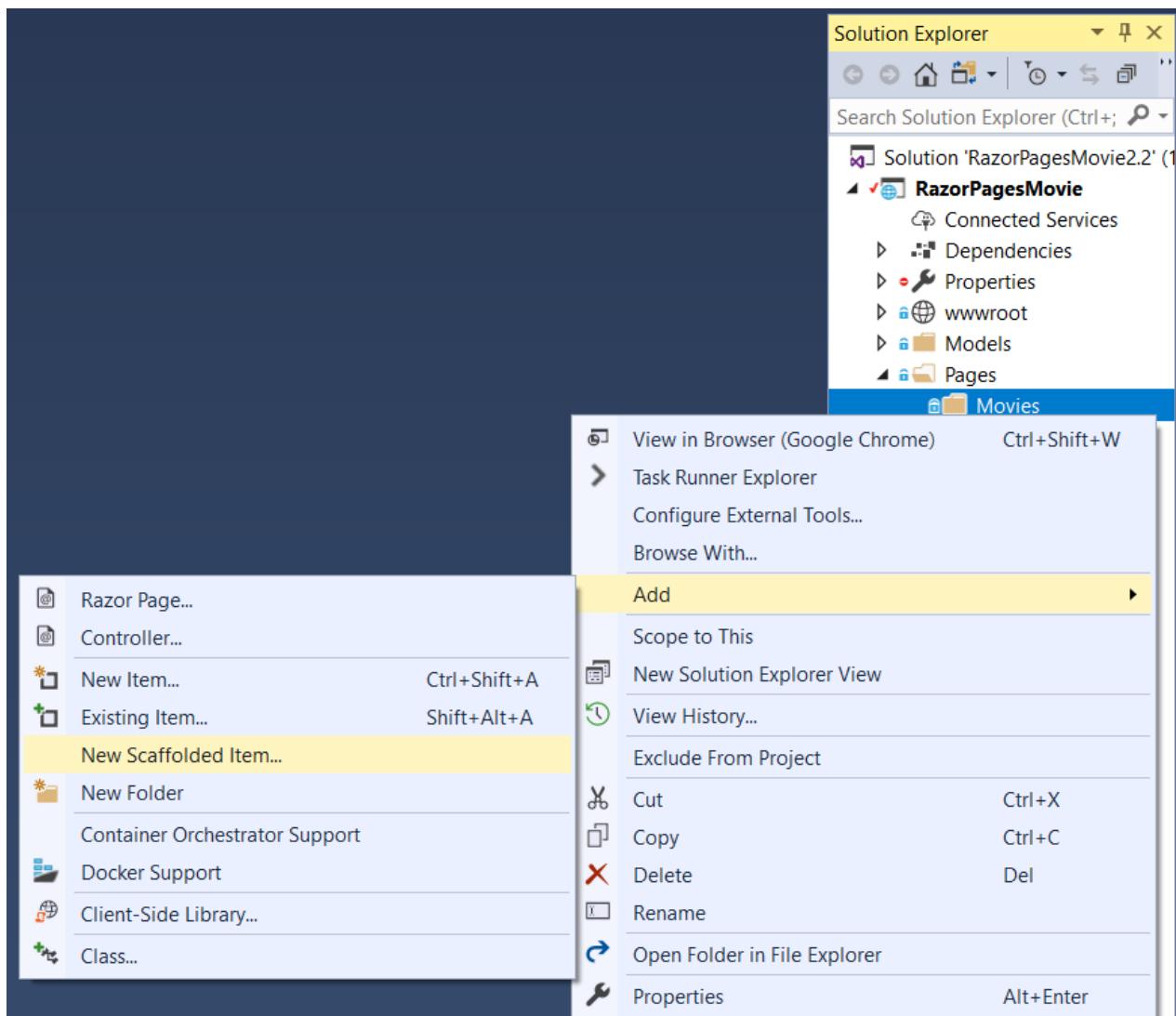
Nesta seção, é feito o scaffold do modelo de filme. Ou seja, a ferramenta de scaffolding gera páginas para operações de CRUD (Criar, Ler, Atualizar e Excluir) para o modelo do filme.

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)

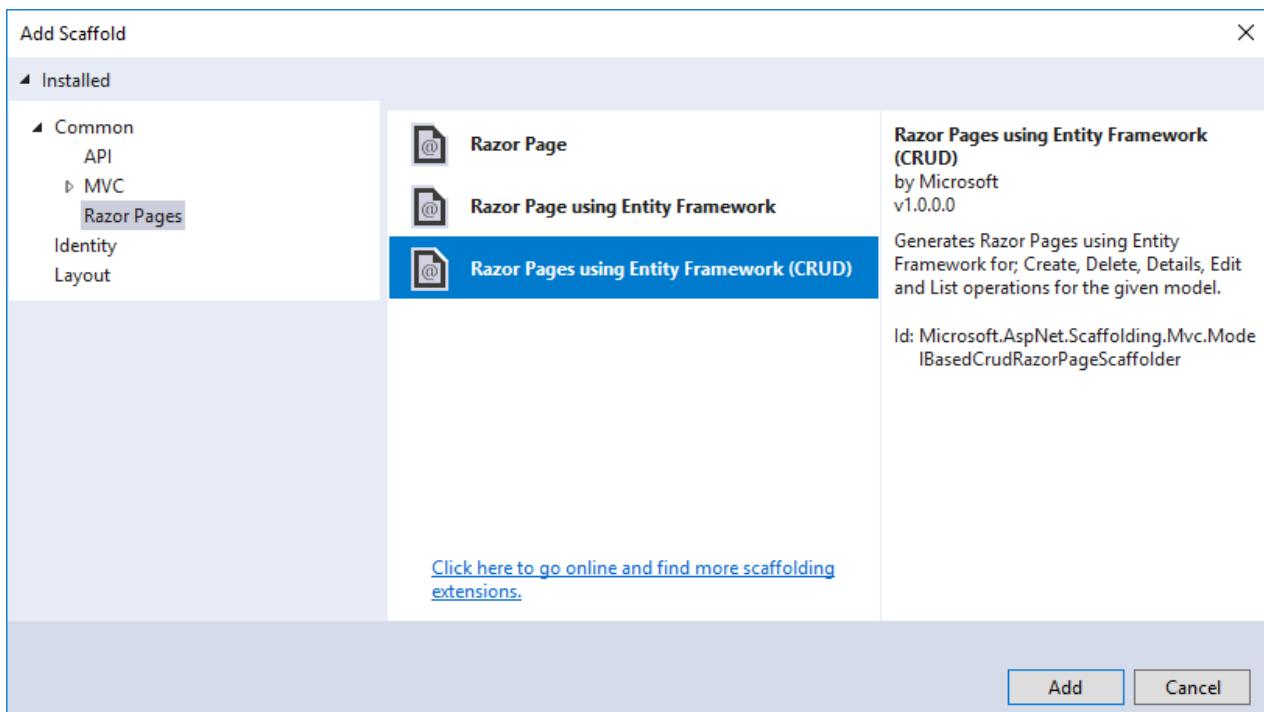
Crie uma pasta *Pages/Movies*:

- Clique com o botão direito do mouse na pasta *Pages* > **Adicionar** > **Nova Pasta**.
- Dê à pasta o nome *Movies*

Clique com o botão direito do mouse na pasta *Pages/Movies* > **Adicionar** > **Novo Item Gerado por Scaffold**.

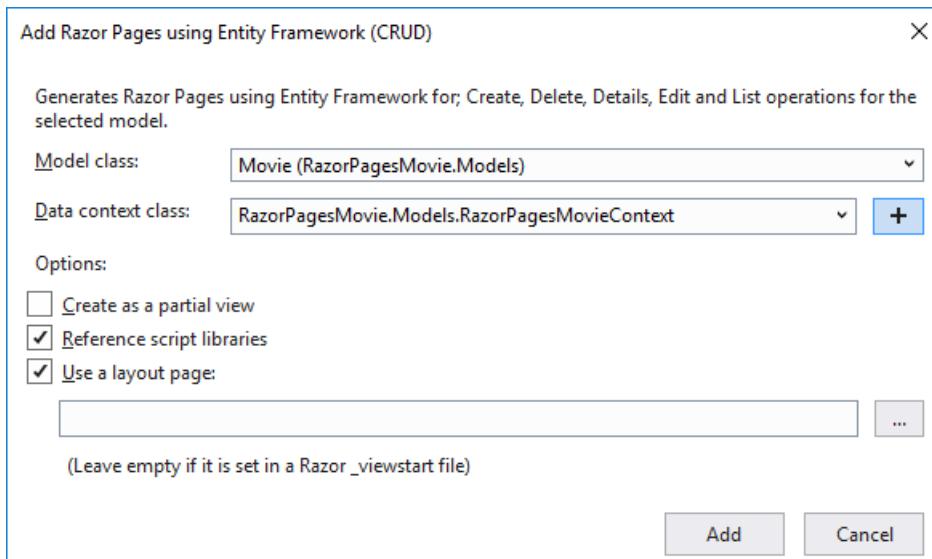


Na caixa de diálogo **Adicionar Scaffold**, selecione **Razor Pages usando o Entity Framework (CRUD)** > **Adicionar**.



Conclua a caixa de diálogo **Adicionar Razor Pages usando o Entity Framework (CRUD)**:

- Na lista suspensa **Classe de modelo**, selecione **Filme (RazorPagesMovie.Models)**.
- Na linha **Classe de contexto de dados**, selecione o sinal + (+) e aceite o nome gerado **RazorPagesMovie.Models.RazorPagesMovieContext**.
- Selecione **Adicionar**.



O arquivo *appsettings.json* é atualizado com a cadeia de conexão usada para se conectar a um banco de dados local.

Os comandos anteriores geram o seguinte aviso: "Nenhum tipo foi especificado para a coluna decimal "Preço" no tipo de entidade "Filme". Isso fará com que valores sejam truncados silenciosamente se não couberem na precisão e na escala padrão. Especifique explicitamente o tipo de coluna do SQL Server que pode acomodar todos os valores usando 'HasColumnType()'."

Você pode ignorar esse aviso, ele será corrigido em um tutorial posterior.

O processo de scaffold cria e atualiza os arquivos a seguir:

#### Arquivos criados

- *Pages/Movies*: Criar, Excluir, Detalhes, Editar e Índice.

- *Data/RazorPagesMovieContext.cs*

## Arquivo atualizado

- *Startup.cs*

Os arquivos criados e atualizados são explicados na próxima seção.

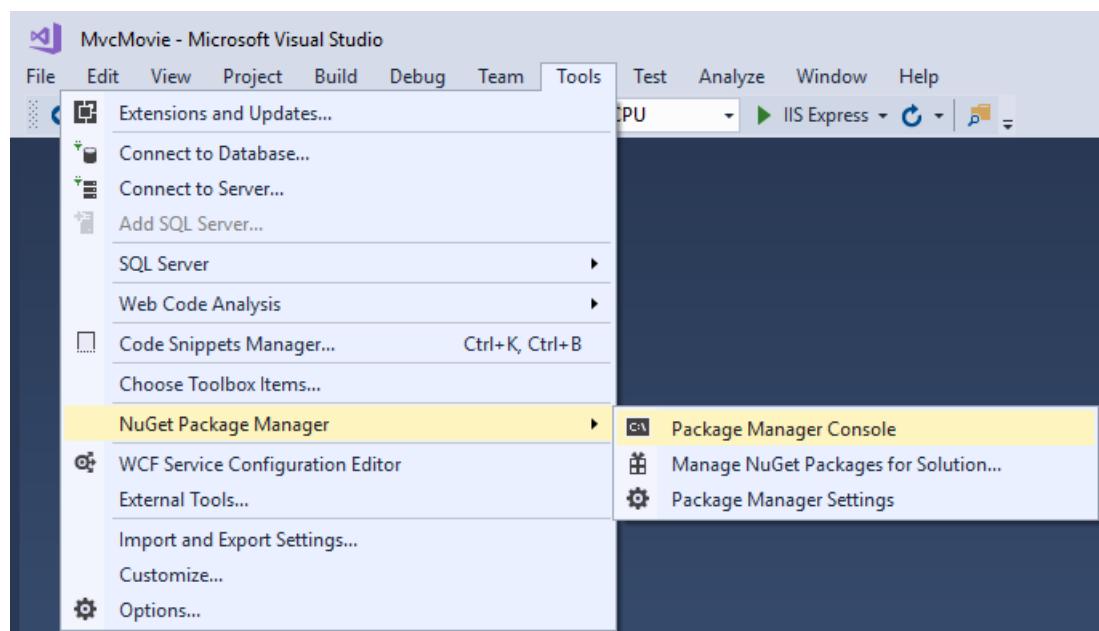
## Migração inicial

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)

Nesta seção, o PMC (Console de Gerenciador de Pacotes) é usado para:

- Adicione uma migração inicial.
- Atualize o banco de dados com a migração inicial.

No menu **Ferramentas**, selecione **Gerenciador de pacotes NuGet > Console do Gerenciador de pacotes**.



No PMC, insira os seguintes comandos:

```
Add-Migration Initial
Update-Database
```

O comando `ef migrations add InitialCreate` gera código para criar o esquema de banco de dados inicial. O esquema é baseado no modelo especificado no `DbContext` (no arquivo *RazorPagesMovieContext.cs*). O argumento `InitialCreate` é usado para nomear as migrações. Qualquer nome pode ser usado, mas, por convenção, um nome que descreve a migração é selecionado.

O comando `ef database update` executa o método `Up` no arquivo *Migrations/<time-stamp>\_InitialCreate.cs*. O método `Up` cria o banco de dados.

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)

## Examinar o contexto registrado com a injeção de dependência

O ASP.NET Core é construído com a [injeção de dependência](#). Serviços (como o contexto de BD do EF Core) são registrados com injeção de dependência durante a inicialização do aplicativo. Os componentes que exigem esses serviços (como as Páginas do Razor) recebem esses serviços por meio de parâmetros do construtor. O código de construtor que obtém uma instância de contexto do BD será mostrado mais adiante no tutorial.

A ferramenta de scaffolding criou automaticamente um contexto de BD e o registrou no contêiner da injeção de dependência.

Examine o método `Startup.ConfigureServices`. A linha destacada foi adicionada pelo scaffold:

```
// This method gets called by the runtime.  
// Use this method to add services to the container.  
public void ConfigureServices(IServiceCollection services)  
{  
    services.Configure<CookiePolicyOptions>(options =>  
    {  
        // This lambda determines whether user consent for non-essential cookies is  
        // needed for a given request.  
        options.CheckConsentNeeded = context => true;  
        options.MinimumSameSitePolicy = SameSiteMode.None;  
    });  
  
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);  
  
    services.AddDbContext<RazorPagesMovieContext>(options =>  
        options.UseSqlServer(  
            Configuration.GetConnectionString("RazorPagesMovieContext")));  
}
```

O `RazorPagesMovieContext` coordena a funcionalidade do EF Core (Criar, Ler, Atualizar, Excluir etc.) para o modelo `Movie`. O contexto de dados (`RazorPagesMovieContext`) deriva de [Microsoft.EntityFrameworkCore.DbContext](#). O contexto de dados especifica quais entidades são incluídas no modelo de dados.

```
using Microsoft.EntityFrameworkCore;  
  
namespace RazorPagesMovie.Models  
{  
    public class RazorPagesMovieContext : DbContext  
    {  
        public RazorPagesMovieContext (DbContextOptions<RazorPagesMovieContext> options)  
            : base(options)  
        {  
        }  
  
        public DbSet<RazorPagesMovie.Models.Movie> Movie { get; set; }  
    }  
}
```

O código anterior cria uma propriedade `DbSet<Movie>` para o conjunto de entidades. Na terminologia do Entity Framework, um conjunto de entidades normalmente corresponde a uma tabela de banco de dados. Uma entidade corresponde a uma linha da tabela.

O nome da cadeia de conexão é passado para o contexto com a chamada de um método em um objeto `DbContextOptions`. Para o desenvolvimento local, o [sistema de configuração do ASP.NET Core](#) lê a cadeia de conexão do arquivo `appsettings.json`.

O comando `Add-Migration` gera código para criar o esquema de banco de dados inicial. O esquema é baseado no modelo especificado no `RazorPagesMovieContext` (no arquivo `Data/RazorPagesMovieContext.cs`). O argumento `Initial` é usado para nomear as migrações. Qualquer nome pode ser usado, mas, por convenção, um nome que

descreve a migração é usado. Consulte [Introdução às migrações](#) para obter mais informações.

O comando `Update-Database` executa o método `Up` no arquivo `Migrations/{time-stamp}_InitialCreate.cs`, que cria o banco de dados.

## Testar o aplicativo

- Executar o aplicativo e acrescentar `/Movies` à URL no navegador (`http://localhost:port/movies`).

Se você obtiver o erro:

```
SqlException: Cannot open database "RazorPagesMovieContext-GUID" requested by the login. The login failed.  
Login failed for user 'User-name'.
```

Você perdeu a [etapa de migrações](#).

- Teste o link **Criar**.

The screenshot shows a browser window titled "Create - RazorPagesMovie". The address bar displays the URL `https://localhost:5001/Movies/Create`. The main content area is titled "Create" and "Movie". It contains four input fields: "Title" with the value "The Good, the bad, and the ugly", "ReleaseDate" with the value "11/30/2018", "Genre" with the value "Western", and "Price" with the value "1.19". Below these fields is a blue "Create" button. At the bottom of the form, there is a link "Back to List" and a footer with the text "© 2019 - RazorPagesMovie - [Privacy](#)".

### NOTE

Talvez você não consiga inserir casas decimais ou vírgulas no campo `Price`. Para dar suporte à [validação do jQuery](#) para localidades com idiomas diferentes do inglês que usam uma vírgula (",") para um ponto decimal e formatos de data diferentes do inglês dos EUA, o aplicativo precisa ser globalizado. Para obter instruções sobre a globalização, consulte [esse problema no GitHub](#).

- Teste os links **Editar**, **Detalhes** e **Excluir**.

O tutorial a seguir explica os arquivos criados por scaffolding.

**ANTERIOR:**  
**INTRODUÇÃO**

**PRÓXIMO: RAZOR PAGES GERADAS POR**  
**SCAFFOLDING**

# Páginas do Razor geradas por scaffolding no ASP.NET Core

28/01/2019 • 14 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

Este tutorial examina as Páginas do Razor criadas por scaffolding no tutorial anterior.

[Exiba ou baixe](#) a amostra.

## As páginas Create, Delete, Details e Edit

Examine o Modelo de Página, *Pages/Movies/Index.cshtml.cs*:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using RazorPagesMovie.Models;

namespace RazorPagesMovie.Pages.Movies
{
    public class IndexModel : PageModel
    {
        private readonly RazorPagesMovie.Models.RazorPagesMovieContext _context;

        public IndexModel(RazorPagesMovie.Models.RazorPagesMovieContext context)
        {
            _context = context;
        }

        public IList<Movie> Movie { get; set; }

        public async Task OnGetAsync()
        {
            Movie = await _context.Movie.ToListAsync();
        }
    }
}
```

As Páginas do Razor são derivadas de `PageModel`. Por convenção, a classe derivada de `PageModel` é chamada de `<PageName>Model`. O construtor usa [injeção de dependência](#) para adicionar o `RazorPagesMovieContext` à página. Todas as páginas geradas por scaffolding seguem esse padrão. Consulte [Código assíncrono](#) para obter mais informações sobre a programação assíncrona com o Entity Framework.

Quando uma solicitação é feita à página, o método `OnGetAsync` retorna uma lista de filmes para a Página do Razor. `OnGetAsync` ou `OnGet` é chamado em uma Página do Razor para inicializar o estado da página. Nesse caso, `OnGetAsync` obtém uma lista de filmes e os exibe.

Quando `OnGet` retorna `void` ou `OnGetAsync` retorna `Task`, então nenhum método de retorno é usado. Quando o tipo de retorno for `IActionResult` ou `Task<IActionResult>`, é necessário fornecer uma instrução de retorno. Por exemplo, o método `OnPostAsync` do arquivo *Pages/Movies/Create.cshtml.cs*:

```
public async Task<IActionResult> OnPostAsync()
{
    if (!ModelState.IsValid)
    {
        return Page();
    }

    _context.Movie.Add(Movie);
    await _context.SaveChangesAsync();

    return RedirectToPage("./Index");
}
```

Examine a Página do Razor *Pages/Movies/Index.cshtml*:

```

@page
@model RazorPagesMovie.Pages.Movies.IndexModel

 @{
     ViewData["Title"] = "Index";
 }

<h1>Index</h1>

<p>
    <a asp-page="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Movie[0].Title)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Movie[0].ReleaseDate)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Movie[0].Genre)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Movie[0].Price)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model.Movie) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Title)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.ReleaseDate)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Genre)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Price)
            </td>
            <td>
                <a asp-page=".Edit" asp-route-id="@item.ID">Edit</a> |
                <a asp-page=".Details" asp-route-id="@item.ID">Details</a> |
                <a asp-page=".Delete" asp-route-id="@item.ID">Delete</a>
            </td>
        </tr>
}
    </tbody>
</table>

```

O Razor pode fazer a transição do HTML em C# ou em marcação específica do Razor. Quando um símbolo `@` é seguido por uma **palavra-chave reservada do Razor**, ele faz a transição para marcação específica do Razor, caso contrário, ele faz a transição para C#.

A diretiva do Razor `@page` transforma o arquivo em uma ação do MVC, o que significa que ele pode manipular solicitações. `@page` deve ser a primeira diretiva do Razor em uma página. `@page` é um exemplo de transição para a marcação específica do Razor. Consulte [Sintaxe Razor](#) para obter mais informações.

Examine a expressão lambda usada no auxiliar HTML a seguir:

```
@Html.DisplayNameFor(model => model.Movie[0].Title)
```

O auxiliar HTML `DisplayNameFor` inspeciona a propriedade `Title` referenciada na expressão lambda para determinar o nome de exibição. A expressão lambda é inspecionada em vez de avaliada. Isso significa que não há nenhuma violação de acesso quando `model`, `model.Movie` ou `model.Movie[0]` são `null` ou vazios. Quando a expressão lambda é avaliada (por exemplo, com `@Html.DisplayFor(modelItem => item.Title)`), os valores de propriedade do modelo são avaliados.

## A diretiva @model

```
@page  
@model RazorPagesMovie.Pages.Movies.IndexModel
```

A diretiva `@model` especifica o tipo de modelo passado para a Página do Razor. No exemplo anterior, a linha `@model` torna a classe derivada de `PageModel` disponível para a Página do Razor. O modelo é usado nos auxiliares HTML `@Html.DisplayNameFor` e `@Html.DisplayFor` na página.

## A página de layout

Selecione os links de menu (**RazorPagesMovie**, **Página Inicial** e **Privacidade**). Cada página mostra o mesmo layout de menu. O layout de menu é implementado no arquivo `Pages/Shared/_Layout.cshtml`. Abra o arquivo `Pages/Shared/_Layout.cshtml`.

Os modelos de `layout` permitem especificar o layout de contêiner HTML do site em um lugar e, em seguida, aplicá-lo a várias páginas do site. Localize a linha `@RenderBody()`. `RenderBody` é um espaço reservado em que todas as visualizações específicas da página criadas são mostradas, *encapsuladas* na página de layout. Por exemplo, se você selecionar o link **Privacidade**, a exibição **Pages/Privacy.cshtml** será renderizada dentro do método `RenderBody`.

## ViewData e layout

Considere o seguinte código do arquivo `Pages/Movies/Index.cshtml`:

```
@page  
@model RazorPagesMovie.Pages.Movies.IndexModel  
  
{@  
    ViewData["Title"] = "Index";  
}
```

O código realçado anterior é um exemplo de transição do Razor para C#. Os caracteres `{` e `}` circunscrevem um bloco de código C#.

A classe base `PageModel` tem uma propriedade de dicionário `ViewData` que pode ser usada para adicionar os dados que você deseja passar para uma exibição. Você adiciona objetos ao dicionário `ViewData` usando um padrão de chave/valor. No exemplo anterior, a propriedade "Title" é adicionada ao dicionário `ViewData`.

A propriedade "Título" é usada no arquivo `Pages/_Layout.cshtml`. A marcação a seguir mostra as primeiras linhas do arquivo `Pages/_Layout.cshtml`.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ ViewData["Title"] - RazorPagesMovie</title>

    /* Markup removed for brevity.*@
```

A linha `/* Markup removed for brevity.*@` é um comentário do Razor que não é exibida no arquivo de layout. Ao contrário de comentários HTML (`<!-- -->`), comentários do Razor não são enviados ao cliente.

## Atualizar o layout

Altere o elemento `<title>` no arquivo `Pages/Shared/_Layout.cshtml` para exibir **Movie**, em vez de **RazorPagesMovie**.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ ViewData["Title"] - Movie</title>
```

Localizar o elemento de âncora a seguir no arquivo `Pages/Shared/_Layout.cshtml`.

```
<a class="navbar-brand" asp-area="" asp-page="/Index">RazorPagesMovie</a>
```

Substitua o elemento anterior pela marcação a seguir.

```
<a class="navbar-brand" asp-page="/Movies/Index">RpMovie</a>
```

O elemento de âncora anterior é um [Auxiliar de Marcas](#). Nesse caso, ele é o [Auxiliar de Marcas de Âncora](#). O atributo e valor do auxiliar de marcas `asp-page="/Movies/Index"` cria um link para a Página do Razor `/Movies/Index`. O valor do atributo `asp-area` está vazio e, portanto, a área não é usada no link. Confira [Áreas](#) para obter mais informações.

Salve suas alterações e teste o aplicativo clicando no link **RpMovie**. Confira o arquivo `_Layout.cshtml` no GitHub caso tenha problemas.

Teste os outros links (**Home**, **RpMovie**, **Create**, **Edit** e **Delete**). Cada página define o título, que pode ser visto na guia do navegador. Quando você coloca um indicador em uma página, o título é usado para o indicador.

### NOTE

Talvez você não consiga inserir casas decimais ou vírgulas no campo `Price`. Para dar suporte à [validação do jQuery](#) para localidades de idiomas diferentes do inglês que usam uma vírgula (",") para um ponto decimal e formatos de data diferentes do inglês dos EUA, você deve tomar medidas para globalizar o aplicativo. Veja [Problema 4076 do GitHub](#) para obter instruções sobre como adicionar casas decimais.

A propriedade `Layout` é definida no arquivo `Pages/_ViewStart.cshtml`:

```
@{
    Layout = "_Layout";
}
```

A marcação anterior define o arquivo de layout como *Pages/Shared/\_Layout.cshtml* para todos os arquivos do Razor na pasta *Pages*. Veja [Layout](#) para obter mais informações.

## O modelo Criar página

Examine o modelo de página *Pages/Movies/Create.cshtml.cs*:

```
// Unused usings removed.
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using RazorPagesMovie.Models;
using System;
using System.Threading.Tasks;

namespace RazorPagesMovie.Pages.Movies
{
    public class CreateModel : PageModel
    {
        private readonly RazorPagesMovie.Models.RazorPagesMovieContext _context;

        public CreateModel(RazorPagesMovie.Models.RazorPagesMovieContext context)
        {
            _context = context;
        }

        public IActionResult OnGet()
        {
            return Page();
        }

        [BindProperty]
        public Movie Movie { get; set; }

        public async Task<IActionResult> OnPostAsync()
        {
            if (!ModelState.IsValid)
            {
                return Page();
            }

            _context.Movie.Add(Movie);
            await _context.SaveChangesAsync();

            return RedirectToPage("./Index");
        }
    }
}
```

O método `OnGet` inicializa qualquer estado necessário para a página. A página Criar não tem nenhum estado para inicializar, assim, `Page` é retornado. Mais adiante no tutorial, você verá o estado de inicialização do método `OnGet`. O método `Page` cria um objeto `PageResult` que renderiza a página *Create.cshtml*.

A propriedade `Movie` usa o atributo `[BindProperty]` para aceitar o [model binding](#). Quando o formulário Criar posta os valores de formulário, o tempo de execução do ASP.NET Core associa os valores postados ao modelo `Movie`.

O método `OnPostAsync` é executado quando a página posta dados de formulário:

```
public async Task<IActionResult> OnPostAsync()
{
    if (!ModelState.IsValid)
    {
        return Page();
    }

    _context.Movie.Add(Movie);
    await _context.SaveChangesAsync();

    return RedirectToPage("./Index");
}
```

Se há algum erro de modelo, o formulário é reexibido juntamente com quaisquer dados de formulário postados. A maioria dos erros de modelo podem ser capturados no lado do cliente antes do formulário ser enviado. Um exemplo de um erro de modelo é postar, para o campo de data, um valor que não pode ser convertido em uma data. A validação do lado do cliente e a validação de modelo são abordadas mais adiante no tutorial.

Se não há nenhum erro de modelo, os dados são salvos e o navegador é redirecionado à página Índice.

## A Página do Razor Criar

Examine o arquivo na Página do Razor *Pages/Movies/Create.cshtml*:

```

@page
@model RazorPagesMovie.Pages.Movies.CreateModel

 @{
     ViewData["Title"] = "Create";
 }

<h1>Create</h1>

<h4>Movie</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Movie.Title" class="control-label"></label>
                <input asp-for="Movie.Title" class="form-control" />
                <span asp-validation-for="Movie.Title" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Movie.ReleaseDate" class="control-label"></label>
                <input asp-for="Movie.ReleaseDate" class="form-control" />
                <span asp-validation-for="Movie.ReleaseDate" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Movie.Genre" class="control-label"></label>
                <input asp-for="Movie.Genre" class="form-control" />
                <span asp-validation-for="Movie.Genre" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Movie.Price" class="control-label"></label>
                <input asp-for="Movie.Price" class="form-control" />
                <span asp-validation-for="Movie.Price" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>

<div>
    <a asp-page="Index">Back to List</a>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}

```

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)

O Visual Studio exibe a marca `<form method="post">` em uma fonte em negrito diferente usada para Auxiliares de Marcas:

```
1 @page
2 @model RazorPagesMovie.Pages_Movie.CreateModel
3
4 @{
5     ViewData["Title"] = "Create";
6 }
7
8 <h2>Create</h2>
9
10 <h4>Movie</h4>
11 <hr />
12 <div class="row">
13     <div class="col-md-4">
14         <form method="post">
15
16             The form element represents a collection of form-associated elements, some of which can represent editable values that can be submitted to a server for processing.
17
18             Learn more \(F1\)
19
20
21             * Microsoft.AspNetCore.Mvc.TagHelpers.FormTagHelper
22                 Microsoft.AspNetCore.Razor.TagHelpers.ITagHelper implementation targeting <form> elements.
23
24                 <span asp-validation-for="Movie.ReleaseDate" class="text-danger">
25                     </div>
26                     <div class="form-group">
27                         <label asp-for="Movie.Genre" class="control-label"></label>
28                         <input asp-for="Movie.Genre" class="form-control" />
29                         <span asp-validation-for="Movie.Genre" class="text-danger"></span>
30                     </div>
31                     <div class="form-group">
32                         <label asp-for="Movie.Price" class="control-label"></label>
33                         <input asp-for="Movie.Price" class="form-control" />
34                         <span asp-validation-for="Movie.Price" class="text-danger"></span>
35                     </div>
36                     <div class="form-group">
37                         <input type="submit" value="Create" class="btn btn-default" />
38                     </div>
39                 </form>
40             </div>

```

O elemento `<form method="post">` é um [auxiliar de marcas de formulário](#). O auxiliar de marcas de formulário inclui automaticamente um [token antifalsificação](#).

O mecanismo de scaffolding cria marcação do Razor para cada campo no modelo (exceto a ID) semelhante ao seguinte:

```
<div asp-validation-summary="ModelOnly" class="text-danger"></div>
<div class="form-group">
    <label asp-for="Movie.Title" class="control-label"></label>
    <input asp-for="Movie.Title" class="form-control" />
    <span asp-validation-for="Movie.Title" class="text-danger"></span>
</div>
```

Os [auxiliares de marcas de validação](#) (`<div asp-validation-summary>` e `<span asp-validation-for>`) exibem erros de validação. A validação será abordada em mais detalhes posteriormente nesta série.

O [auxiliar de marcas de rótulo](#) (`<label asp-for="Movie.Title" class="control-label"></label>`) gera a legenda do rótulo e o atributo `for` para a propriedade `Title`.

O [auxiliar de marcas de entrada](#) (`<input asp-for="Movie.Title" class="form-control" />`) usa os atributos `DataAnnotations` e produz os atributos HTML necessários para validação jQuery no lado do cliente.



# Trabalhar com um banco de dados e o ASP.NET Core

13/12/2018 • 10 minutes to read • [Edit Online](#)

Por [Rick Anderson](#) e [Joe Audette](#)

[View or download sample code \(how to download\).](#)

O objeto `RazorPagesMovieContext` cuida da tarefa de se conectar ao banco de dados e mapear objetos `Movie` para registros do banco de dados. O contexto de banco de dados é registrado com o contêiner [Injeção de Dependência](#) no método `ConfigureServices` em `Startup.cs`:

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)

```
// This method gets called by the runtime.  
// Use this method to add services to the container.  
public void ConfigureServices(IServiceCollection services)  
{  
    services.Configure<CookiePolicyOptions>(options =>  
    {  
        // This lambda determines whether user consent for non-essential cookies is  
        // needed for a given request.  
        options.CheckConsentNeeded = context => true;  
        options.MinimumSameSitePolicy = SameSiteMode.None;  
    });  
  
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);  
  
    services.AddDbContext<RazorPagesMovieContext>(options =>  
        options.UseSqlServer(  
            Configuration.GetConnectionString("RazorPagesMovieContext")));  
}
```

Para obter mais informações sobre os métodos usados em `ConfigureServices`, veja:

- [Suporte ao RGPD \(Regulamento Geral sobre a Proteção de Dados\) da UE no ASP.NET Core](#) para `CookiePolicyOptions`.
- [SetCompatibilityVersion](#)

O sistema de [Configuração](#) do ASP.NET Core lê a `ConnectionString`. Para o desenvolvimento local, ele obtém a cadeia de conexão do arquivo `appsettings.json`.

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)

O valor do nome do banco de dados (`Database={Database name}`) será diferente para o seu código gerado. O valor do nome é arbitrário.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "RazorPagesMovieContext": "Server=(localdb)\\mssqllocaldb;Database=RazorPagesMovieContext-1234;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}
```

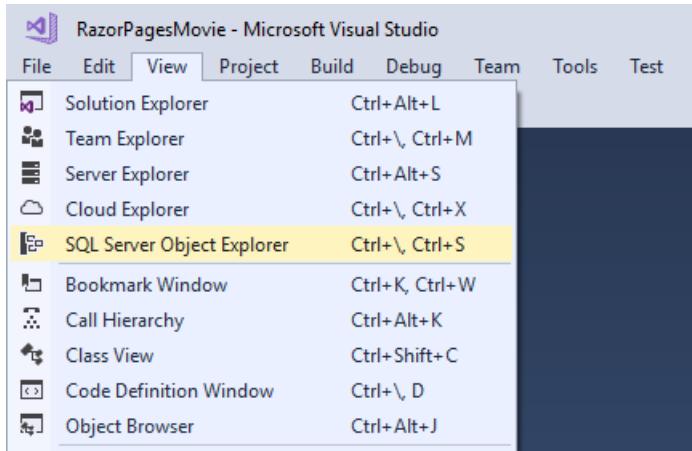
Quando o aplicativo é implantado em um servidor de teste ou de produção, uma variável de ambiente pode ser usada para definir a cadeia de conexão como um servidor de banco de dados real. Consulte [Configuração](#) para obter mais informações.

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)

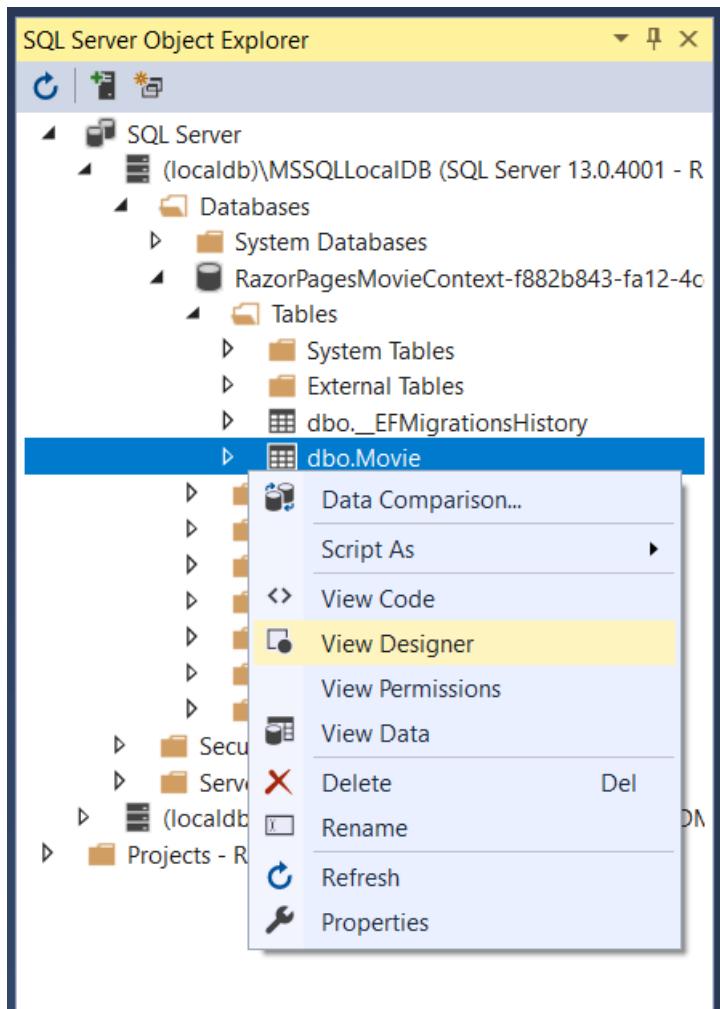
## SQL Server Express LocalDB

O LocalDB é uma versão leve do mecanismo de banco de dados do SQL Server Express direcionada para o desenvolvimento de programas. O LocalDB é iniciado sob demanda e executado no modo de usuário e, portanto, não há nenhuma configuração complexa. Por padrão, o banco de dados LocalDB cria arquivos `*.mdf` no diretório `C:/Users/<user/>`.

- No menu **Exibir**, abra **SSOX** (Pesquisador de Objetos do SQL Server).



- Clique com o botão direito do mouse na tabela `Movie` e selecione **Designer de exibição**:



The screenshot shows the 'dbo.Movie [Design]' window. The table structure is displayed with columns: ID, Genre, Price, ReleaseDate, and Title. The 'ID' column is defined as int, primary key, clustered index. The 'Genre' column is nvarchar(MAX) with Allow Nulls checked. The 'Price' column is decimal(18,2). The 'ReleaseDate' column is datetime2(7). The 'Title' column is nvarchar(MAX) with Allow Nulls checked. On the right side, there are sections for Keys (1), Check Constraints (0), Indexes (0), Foreign Keys (0), and Triggers (0). Below the table structure, a T-SQL tab shows the CREATE TABLE statement:

```
1 CREATE TABLE [dbo].[Movie] (
2     [ID] INT IDENTITY (1, 1) NOT NULL,
3     [Genre] NVARCHAR (MAX) NULL,
4     [Price] DECIMAL (18, 2) NOT NULL,
5     [ReleaseDate] DATETIME2 (7) NOT NULL,
6     [Title] NVARCHAR (MAX) NULL,
7     CONSTRAINT [PK_Movie] PRIMARY KEY CLUSTERED ([ID] ASC)
8 );
```

Observe o ícone de chave ao lado de `ID`. Por padrão, o EF cria uma propriedade chamada `ID` para a chave primária.

- Clique com o botão direito do mouse na tabela `Movie` e selecione **Exibir dados**:

	ID	Genre	Price	ReleaseDate	Title
	3	Action	1.99	1/1/0001 12:00:...	Conan
	2003	Comedy	1.99	8/4/2017 6:27:3...	Back to the Future
▶	2004	Western	1.19	8/4/2017 7:08:3...	The Good, the bad, and t...
*	NULL	NULL	NULL	NULL	NULL

## Propagar o banco de dados

Crie uma classe chamada `SeedData` na pasta *Models* com o seguinte código:

```

using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using System;
using System.Linq;

namespace RazorPagesMovie.Models
{
    public static class SeedData
    {
        public static void Initialize(IServiceProvider serviceProvider)
        {
            using (var context = new RazorPagesMovieContext(
                serviceProvider.GetRequiredService<
                    DbContextOptions<RazorPagesMovieContext>>()))
            {
                // Look for any movies.
                if (context.Movie.Any())
                {
                    return; // DB has been seeded
                }

                context.Movie.AddRange(
                    new Movie
                    {
                        Title = "When Harry Met Sally",
                        ReleaseDate = DateTime.Parse("1989-2-12"),
                        Genre = "Romantic Comedy",
                        Price = 7.99M
                    },
                    new Movie
                    {
                        Title = "Ghostbusters ",
                        ReleaseDate = DateTime.Parse("1984-3-13"),
                        Genre = "Comedy",
                        Price = 8.99M
                    },
                    new Movie
                    {
                        Title = "Ghostbusters 2",
                        ReleaseDate = DateTime.Parse("1986-2-23"),
                        Genre = "Comedy",
                        Price = 9.99M
                    },
                    new Movie
                    {
                        Title = "Rio Bravo",
                        ReleaseDate = DateTime.Parse("1959-4-15"),
                        Genre = "Western",
                        Price = 3.99M
                    }
                );
                context.SaveChanges();
            }
        }
    }
}

```

Se houver um filme no BD, o inicializador de semeadura será retornado e nenhum filme será adicionado.

```
if (context.Movie.Any())
{
    return; // DB has been seeded.
}
```

## Adicionar o inicializador de semeadura

Em `Program.cs`, modifique o método `Main` para fazer o seguinte:

- Obtenha uma instância de contexto de BD do contêiner de injeção de dependência.
- Chame o método de semente passando a ele o contexto.
- Descarte o contexto quando o método de semente for concluído.

O código a seguir mostra o arquivo `Program.cs` atualizado.

```
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using RazorPagesMovie.Models;
using System;
using Microsoft.EntityFrameworkCore;

namespace RazorPagesMovie
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var host = CreateWebHostBuilder(args).Build();

            using (var scope = host.Services.CreateScope())
            {
                var services = scope.ServiceProvider;

                try
                {
                    var context=services.
                        GetRequiredService<RazorPagesMovieContext>();
                    context.Database.Migrate();
                    SeedData.Initialize(services);
                }
                catch (Exception ex)
                {
                    var logger = services.GetRequiredService<ILogger<Program>>();
                    logger.LogError(ex, "An error occurred seeding the DB.");
                }
            }

            host.Run();
        }

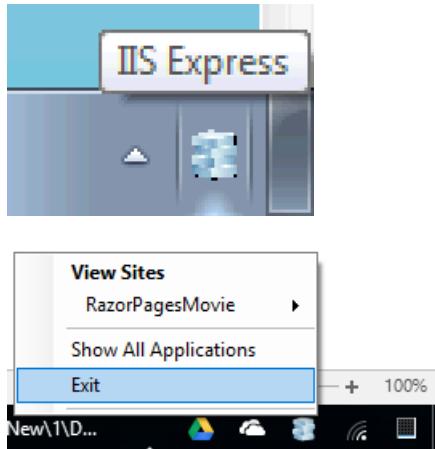
        public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>();
    }
}
```

Um aplicativo de produção não chamaria `Database.Migrate`. Ele é adicionado ao código anterior para evitar a exceção a seguir quando `Update-Database` não foi executado:

`SqlException: não pode abrir o banco de dados "RazorPagesMovieContext-21" solicitado pelo logon. O logon falhou. O logon falhou para o usuário 'user name'.`

## Testar o aplicativo

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- Exclua todos os registros no BD. Faça isso com os links Excluir no navegador ou no [SSOX](#)
- Force o aplicativo a ser inicializado (chame os métodos na classe `Startup`) para que o método de sementeada seja executado. Para forçar a inicialização, o IIS Express deve ser interrompido e reiniciado. Faça isso com uma das seguintes abordagens:
  - Clique com botão direito do mouse no ícone de bandeja do sistema do IIS Express na área de notificação e toque em **Sair** ou **Parar site**:



- Se você estiver executando o VS no modo sem depuração, pressione F5 para executar no modo de depuração.
- Se você estiver executando o VS no modo de depuração, pare o depurador e pressione F5.

O aplicativo mostra os dados propagados:

The screenshot shows a Microsoft Edge browser window. The address bar displays 'Index - Movie' and 'https://localhost:5001/Movies'. The page content is the 'Index' view of the 'RpMovie' application. It features a navigation bar with 'RpMovie', 'Home', and 'Privacy' links. Below the navigation is a heading 'Index' and a 'Create New' button. A table lists five movies with the following data:

Title	ReleaseDate	Genre	Price	
When Harry Met Sally	2/12/1989	Romantic Comedy	7.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Ghostbusters	3/13/1984	Comedy	8.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Ghostbusters 2	2/23/1986	Comedy	9.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Rio Bravo	4/15/1959	Western	3.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
The Good, the bad, and the ugly	12/1/2018	Western	1.19	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

At the bottom of the page, there is a footer with the text '© 2019 - RazorPagesMovie - [Privacy](#)'.

O próximo tutorial limpará a apresentação dos dados.

ANTERIOR: PÁGINAS DO RAZOR GERADAS POR  
SCAFFOLDING

PRÓXIMO: ATUALIZAÇÃO DAS  
PÁGINAS

# Atualizar as páginas geradas em um aplicativo ASP.NET Core

10/01/2019 • 8 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

O aplicativo de filme gerado por scaffolding tem um bom começo, mas a apresentação não é ideal. **ReleaseDate** deveria ser **Data de Lançamento** (duas palavras).

Title	ReleaseDate	Genre	Price	
When Harry Met Sally	2/12/1989	Romantic Comedy	7.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Ghostbusters	3/13/1984	Comedy	8.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Ghostbusters 2	2/23/1986	Comedy	9.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Rio Bravo	4/15/1959	Western	3.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
The Good, the bad, and the ugly	12/1/2018	Western	1.19	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

© 2019 - RazorPagesMovie - [Privacy](#)

## Atualize o código gerado

Abra o arquivo *Models/Movie.cs* e adicione as linhas realçadas mostradas no seguinte código:

```

using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace RazorPagesMovie.Models
{
    public class Movie
    {
        public int ID { get; set; }
        public string Title { get; set; }

        [Display(Name = "Release Date")]
        [DataType(DataType.Date)]
        public DateTime ReleaseDate { get; set; }
        public string Genre { get; set; }

        [Column(TypeName = "decimal(18, 2)")]
        public decimal Price { get; set; }
    }
}

```

A anotação de dados `[Column(TypeName = "decimal(18, 2)")]` permite que o Entity Framework Core mapeie o `Price` corretamente para a moeda no banco de dados. Para obter mais informações, veja [Tipos de Dados](#).

[DataAnnotations](#) é abordado no próximo tutorial. O atributo `Display` especifica o que deve ser exibido no nome de um campo (neste caso, "Release Date" em vez de "ReleaseDate"). O atributo `DataType` especifica o tipo de dados (Data) e, portanto, as informações de hora armazenadas no campo não são exibidas.

Procure Pages/Movies e focalize um link **Editar** para ver a URL de destino.

Title	Release Date	Genre	Price	
When Harry Met Sally	2/12/1989	Romantic Comedy	7.99	Edit   Details   Delete
Ghostbusters	3/13/1984	Comedy	8.99	Edit   Details   Delete
Ghostbusters 2	2/23/1986	Comedy	9.99	Edit   Details   Delete
Rio Bravo	4/15/1959	Western	3.99	Edit   Details   Delete

© 2019 - RazorPagesMovie - [Privacy](#)  
<https://localhost:5001/Movies/Edit?id=3>

Os links **Editar**, **Detalhes** e **Excluir** são gerados pelo [Auxiliar de Marcação de Âncora](#) no arquivo `Pages/Movies/Index.cshtml`.

```

@foreach (var item in Model.Movie) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Title)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.ReleaseDate)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Genre)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Price)
        </td>
        <td>
            <a asp-page=".Edit" asp-route-id="@item.ID">Edit</a> | 
            <a asp-page=".Details" asp-route-id="@item.ID">Details</a> | 
            <a asp-page=".Delete" asp-route-id="@item.ID">Delete</a>
        </td>
    </tr>
}
</tbody>
</table>

```

Os [Auxiliares de Marcação](#) permitem que o código do servidor participe da criação e renderização de elementos HTML em arquivos do Razor. No código anterior, o `AnchorTagHelper` gera dinamicamente o valor do atributo `href` HTML da página Razor (a rota é relativa), o `asp-page` e a ID da rota (`asp-route-id`). Consulte [Geração de URL para Páginas](#) para obter mais informações.

Use **Exibir Código-fonte** em seu navegador favorito para examinar a marcação gerada. Uma parte do HTML gerado é mostrada abaixo:

```

<td>
    <a href="/Movies/Edit?id=1">Edit</a> | 
    <a href="/Movies/Details?id=1">Details</a> | 
    <a href="/Movies/Delete?id=1">Delete</a>
</td>

```

Os links gerados dinamicamente passam a ID de filme com uma cadeia de consulta (por exemplo, o `?id=1` em <https://localhost:5001/Movies/Details?id=1>).

Atualize as Páginas Editar, Detalhes e Excluir do Razor para que elas usem o modelo de rota `{id:int}`. Altere a diretiva de página de cada uma dessas páginas de `@page` para `@page "{id:int}"`. Execute o aplicativo e, em seguida, exiba o código-fonte. O HTML gerado adiciona a ID à parte do caminho da URL:

```

<td>
    <a href="/Movies/Edit/1">Edit</a> | 
    <a href="/Movies/Details/1">Details</a> | 
    <a href="/Movies/Delete/1">Delete</a>
</td>

```

Uma solicitação para a página com o modelo de rota `{id:int}` que **não** inclui o inteiro retornará um erro HTTP 404 (não encontrado). Por exemplo, <http://localhost:5000/Movies/Details> retornará um erro 404. Para tornar a ID opcional, acrescente `?` à restrição de rota:

```

@page "{id:int?}"

```

Para testar o comportamento de `@page "{id:int?}"`:

- defina a diretiva de página em `Pages/Movies/Details.cshtml` como `@page "{id:int?}"`.
- Defina um ponto de interrupção em `public async Task<IActionResult> OnGetAsync(int? id)` (em `Pages/Movies/Details.cshtml.cs`).
- Navegue para `https://localhost:5001/Movies/Details/`.

Com a diretiva `@page "{id:int}"`, o ponto de interrupção nunca é atingido. O mecanismo de roteamento retorna HTTP 404. Usando `@page "{id:int?}"`, o método `OnGetAsync` retorna `NotFound` (HTTP 404).

Embora não seja recomendado, você pode escrever o método `OnGetAsync` (em `Pages/Movies/Delete.cshtml.cs`) como:

```
public async Task<IActionResult> OnGetAsync(int? id)
{
    if (id == null)
    {
        Movie = await _context.Movie.FirstOrDefaultAsync();
    }
    else
    {
        Movie = await _context.Movie.FirstOrDefaultAsync(m => m.ID == id);
    }

    if (Movie == null)
    {
        return NotFound();
    }
    return Page();
}
```

Teste o código anterior:

- Selecione um link de **exclusão**.
- Remova a ID da URL. Por exemplo, altere `https://localhost:5001/Movies/Delete/8` para `https://localhost:5001/Movies/Delete`.
- Execute o código em etapas no depurador.

### **Examinar o tratamento de exceção de simultaneidade**

Examine o método `OnPostAsync` no arquivo `Pages/Movies/Edit.cshtml.cs`:

```

public async Task<IActionResult> OnPostAsync()
{
    if (!ModelState.IsValid)
    {
        return Page();
    }

    _context.Attach(Movie).State = EntityState.Modified;

    try
    {
        await _context.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!MovieExists(Movie.ID))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }

    return RedirectToPage("./Index");
}

private bool MovieExists(int id)
{
    return _context.Movie.Any(e => e.ID == id);
}

```

O código anterior detecta exceções de simultaneidade quando um cliente exclui o filme e o outro cliente posta alterações no filme.

Para testar o bloco `catch` :

- Definir um ponto de interrupção em `catch (DbUpdateConcurrencyException)`
- Selecione **Editar** para um filme, faça alterações, mas não insira **Salvar**.
- Em outra janela do navegador, selecione o link **Excluir** do mesmo filme e, em seguida, exclua o filme.
- Na janela do navegador anterior, poste as alterações no filme.

O código de produção talvez deseje detectar conflitos de simultaneidade. Confira [Lidar com conflitos de simultaneidade](#) para obter mais informações.

### Análise de postagem e associação

Examine o arquivo *Pages/Movies/Edit.cshtml.cs*:

```

public class EditModel : PageModel
{
    private readonly RazorPagesMovieContext _context;

    public EditModel(RazorPagesMovieContext context)
    {
        _context = context;
    }

    [BindProperty]
    public Movie Movie { get; set; }

    public async Task<IActionResult> OnGetAsync(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        Movie = await _context.Movie.SingleOrDefaultAsync(m => m.ID == id);

        if (Movie == null)
        {
            return NotFound();
        }
        return Page();
    }

    public async Task<IActionResult> OnPostAsync()
    {
        if (!ModelState.IsValid)
        {
            return Page();
        }

        _context.Attach(Movie).State = EntityState.Modified;

        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!_context.Movie.Any(e => e.ID == Movie.ID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }

        return RedirectToPage("./Index");
    }
}

```

Quando uma solicitação HTTP GET é feita para a página Movies/Edit (por exemplo,

`http://localhost:5000/Movies/Edit/2`):

- O método `OnGetAsync` busca o filme do banco de dados e retorna o método `Page`.
- O método `Page` renderiza a página `Razor Pages/Movies/Edit.cshtml`. O arquivo `Pages/Movies/Edit.cshtml` contém a diretiva de modelo (`@model RazorPagesMovie.Pages.Movies.EditModel`), que disponibiliza o modelo de filme na página.

- O formulário Editar é exibido com os valores do filme.

Quando a página Movies/Edit é postada:

- Os valores de formulário na página são associados à propriedade `Movie`. O atributo `[BindProperty]` habilita o [Model binding](#).

```
[BindProperty]  
public Movie Movie { get; set; }
```

- Se houver erros no estado do modelo (por exemplo, `ReleaseDate` não pode ser convertido em uma data), o formulário será mostrado com os valores enviados.
- Se não houver erros do modelo, o filme será salvo.

Os métodos HTTP GET nas páginas Índice, Criar e Excluir do Razor seguem um padrão semelhante. O método `OnPostAsync` HTTP POST na página Criar do Razor segue um padrão semelhante ao método `OnPostAsync` na página Editar do Razor.

A pesquisa é adicionada no próximo tutorial.

[ANTERIOR: TRABALHANDO COM UM BANCO DE  
DADOS](#)

[PRÓXIMO: ADICIONAR  
PESQUISA](#)

# Adicionar a pesquisa às Páginas Razor do ASP.NET Core

10/01/2019 • 8 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

[View or download sample code \(how to download\).](#)

Nas seções a seguir, a pesquisa de filmes por *gênero* ou *nome* é adicionada.

Adicione as seguintes propriedades realçadas em *Pages/Movies/Index.cshtml.cs*:

```
public class IndexModel : PageModel
{
    private readonly RazorPagesMovie.Models.RazorPagesMovieContext _context;

    public IndexModel(RazorPagesMovie.Models.RazorPagesMovieContext context)
    {
        _context = context;
    }

    public IList<Movie> Movie { get; set; }
    [BindProperty(SupportsGet = true)]
    public string SearchString { get; set; }
    // Requires using Microsoft.AspNetCore.Mvc.Rendering;
    public SelectList Genres { get; set; }
    [BindProperty(SupportsGet = true)]
    public string MovieGenre { get; set; }
```

- `SearchString`: contém o texto que os usuários inserem na caixa de texto de pesquisa. `SearchString` está decorada com o atributo `[BindProperty]`. `[BindProperty]` associa valores de formulário e cadeias de consulta ao mesmo nome da propriedade. `(SupportsGet = true)` é necessário para a associação em solicitações GET.
- `Genres`: contém a lista de gêneros. `Genres` permite que o usuário selecione um gênero na lista. `SelectList` exige `using Microsoft.AspNetCore.Mvc.Rendering;`
- `MovieGenre`: contém o gênero específico que o usuário seleciona (por exemplo, "Oeste").
- `Genres` e `MovieGenre` são abordados mais adiante neste tutorial.

## WARNING

Por motivos de segurança, você deve aceitar associar os dados da solicitação `GET` às propriedades do modelo de página. Verifique a entrada do usuário antes de mapeá-la para as propriedades. Aceitar a associação de `GET` é útil ao lidar com cenários que contam com a cadeia de caracteres de consulta ou com os valores de rota.

Para associar uma propriedade às solicitações `GET`, defina a propriedade `SupportsGet` do atributo `[BindProperty]` como `true` : `[BindProperty(SupportsGet = true)]`

Atualize o método `OnGetAsync` da página de Índice pelo seguinte código:

```
public async Task OnGetAsync()
{
    var movies = from m in _context.Movie
                 select m;
    if (!string.IsNullOrEmpty(SearchString))
    {
        movies = movies.Where(s => s.Title.Contains(SearchString));
    }

    Movie = await movies.ToListAsync();
}
```

A primeira linha do método `OnGetAsync` cria uma consulta [LINQ](#) para selecionar os filmes:

```
// using System.Linq;
var movies = from m in _context.Movie
             select m;
```

A consulta é *somente* definida neste ponto; ela **não** foi executada no banco de dados.

Se a propriedade `SearchString` não é nula nem vazia, a consulta de filmes é modificada para filtrar a cadeia de pesquisa:

```
if (!string.IsNullOrEmpty(SearchString))
{
    movies = movies.Where(s => s.Title.Contains(SearchString));
}
```

O código `s => s.Title.Contains()` é uma [Expressão Lambda](#). Lambdas são usados em consultas [LINQ](#) baseadas em método como argumentos para métodos de operadores de consulta padrão, como o método [Where](#) ou [Contains](#) (usado no código anterior). As consultas LINQ não são executadas quando são definidas ou quando são modificadas com uma chamada a um método (como `Where`, `Contains` ou `OrderBy`). Em vez disso, a execução da consulta é adiada. Isso significa que a avaliação de uma expressão é atrasada até que seu valor realizado seja iterado ou o método `ToListAsync` seja chamado. Consulte [Execução de consulta](#) para obter mais informações.

**Observação:** o método [Contains](#) é executado no banco de dados, não no código C#. A diferenciação de maiúsculas e minúsculas na consulta depende do banco de dados e da ordenação. No SQL Server, `Contains` é mapeado para [SQL LIKE](#), que não diferencia maiúsculas de minúsculas. No SQLite, com a ordenação padrão, ele diferencia maiúsculas de minúsculas.

Navegue para a página Movies e acrescente uma cadeia de consulta, como `?searchString=Ghost`, à URL (por exemplo, <https://localhost:5001/Movies?searchString=Ghost>). Os filmes filtrados são exibidos.

Index

Create New

Title	Release Date	Genre	Price
Ghostbusters	3/13/1984	Comedy	8.99
Ghostbusters 2	2/23/1986	Comedy	9.99

© 2019 - RazorPagesMovie - [Privacy](#)

Se o modelo de rota a seguir for adicionado à página de Índice, a cadeia de caracteres de pesquisa poderá ser passada como um segmento de URL (por exemplo, `https://localhost:5001/Movies/Ghost`).

```
@page "{searchString?"}
```

A restrição da rota anterior permite a pesquisa do título como dados de rota (um segmento de URL), em vez de como um valor de cadeia de caracteres de consulta. O `{}` em `"{searchString?}"` significa que esse é um parâmetro de rota opcional.

Index

Create New

Title	Release Date	Genre	Price
Ghostbusters	3/13/1984	Comedy	8.99
Ghostbusters 2	2/23/1986	Comedy	9.99

© 2019 - RazorPagesMovie - [Privacy](#)

O tempo de execução do ASP.NET Core usa o [model binding](#) para definir o valor da propriedade `SearchString` na cadeia de consulta (`?searchString=Ghost`) ou nos dados de rota (`https://localhost:5001/Movies/Ghost`). O model binding não diferencia maiúsculas de minúsculas.

No entanto, você não pode esperar que os usuários modifiquem a URL para pesquisar um filme. Nesta etapa, a interface do usuário é adicionada para filtrar filmes. Se você adicionou a restrição de rota `"{searchString?}"`, remova-a.

Abra o arquivo `Pages/Movies/Index.cshtml` e, em seguida, adicione a marcação `<form>` realçada no seguinte código:

```
@page
@model RazorPagesMovie.Pages.Movies.IndexModel

 @{
    ViewData["Title"] = "Index";
}

<h1>Index</h1>

<p>
    <a asp-page="Create">Create New</a>
</p>

<form>
    <p>
        Title: <input type="text" asp-for="SearchString" />
        <input type="submit" value="Filter" />
    </p>
</form>

<table class="table">
    @*Markup removed for brevity.*@
```

A marca `<form>` HTML usa os seguintes [Auxiliares de Marcas](#):

- [Auxiliar de Marcas de Formulário](#). Quando o formulário é enviado, a cadeia de caracteres de filtro é enviada para a página `Pages/Movies/Index` por meio da cadeia de consulta.
- [Auxiliar de marcação de entrada](#)

Salve as alterações e teste o filtro.

Index - Movie

RpMovie Home Privacy

# Index

Create New

Title:  Filter

Title	Release Date	Genre	Price	
Ghostbusters	3/13/1984	Comedy	8.99	<a>Edit</a>   <a>Details</a>   <a>Delete</a>
Ghostbusters 2	2/23/1986	Comedy	9.99	<a>Edit</a>   <a>Details</a>   <a>Delete</a>

© 2019 - RazorPagesMovie - Privacy

## Pesquisar por gênero

Atualize o método `OnGetAsync` pelo seguinte código:

```
public async Task OnGetAsync()
{
    // Use LINQ to get list of genres.
    IQueryable<string> genreQuery = from m in _context.Movie
                                         orderby m.Genre
                                         select m.Genre;

    var movies = from m in _context.Movie
                 select m;

    if (!string.IsNullOrEmpty(SearchString))
    {
        movies = movies.Where(s => s.Title.Contains(SearchString));
    }

    if (!string.IsNullOrEmpty(MovieGenre))
    {
        movies = movies.Where(x => x.Genre == MovieGenre);
    }

    Genres = new SelectList(await genreQuery.Distinct().ToListAsync());
    Movie = await movies.ToListAsync();
}
```

O código a seguir é uma consulta LINQ que recupera todos os gêneros do banco de dados.

```
// Use LINQ to get list of genres.  
IQueryable<string> genreQuery = from m in _context.Movies  
                                orderby m.Genre  
                                select m.Genre;
```

O `SelectList` de gêneros é criado com a projeção dos gêneros distintos.

```
Genres = new SelectList(await genreQuery.Distinct().ToListAsync());
```

## Adicionar pesquisa por gênero ao Razor Page

Atualize `Index.cshtml` da seguinte maneira:

```
@page
@model RazorPagesMovie.Pages.Movies.IndexModel

 @{
    ViewData["Title"] = "Index";
}

<h1>Index</h1>

<p>
    <a asp-page="Create">Create New</a>
</p>

<form>
    <p>
        <select asp-for="MovieGenre" asp-items="Model.Genres">
            <option value="">All</option>
        </select>
        Title: <input type="text" asp-for="SearchString" />
        <input type="submit" value="Filter" />
    </p>
</form>

<table class="table">
    @*Markup removed for brevity.*@
```

Teste o aplicativo pesquisando por gênero, título do filme e por ambos.

ANTERIOR: ATUALIZANDO AS  
PÁGINAS

PRÓXIMO: ADICIONANDO UM NOVO  
CAMPO

# Adicionar um novo campo em uma página Razor no ASP.NET Core

10/01/2019 • 10 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

[View or download sample code \(how to download\).](#)

Nesta seção, as Migrações do [Entity Framework](#) Code First são usadas para:

- Adicionar um novo campo ao modelo.
- Migrar a nova alteração de esquema de campo para o banco de dados.

Ao usar o Code First do EF para criar automaticamente um banco de dados, o Code First:

- Adiciona uma tabela ao banco de dados para acompanhar se o esquema do banco de dados está sincronizado com as classes de modelo das quais ele foi gerado.
- Se as classes de modelo não estiverem em sincronia com o banco de dados, o EF gerará uma exceção.

Verificação automática de esquema/modelo em sincronia torna mais fácil encontrar problemas de código/banco de dados inconsistente.

## Adicionando uma propriedade de classificação ao modelo de filme

Abra o arquivo *Models/Movie.cs* e adicione uma propriedade `Rating`:

```
public class Movie
{
    public int ID { get; set; }
    public string Title { get; set; }

    [Display(Name = "Release Date")]
    [DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }
    public string Genre { get; set; }

    [Column(TypeName = "decimal(18, 2)")]
    public decimal Price { get; set; }
    public string Rating { get; set; }
}
```

Crie o aplicativo.

Edite *Pages/Movies/Index.cshtml* e adicione um campo `Rating`:

```
@page
@model RazorPagesMovie.Pages.Movies.IndexModel

 @{
     ViewData["Title"] = "Index";
 }

<h1>Index</h1>

<p>
    <a asp-page="Create">Create New</a>

```

```

</p>

<form>
    <p>
        <select asp-for="MovieGenre" asp-items="Model.Genres">
            <option value="">All</option>
        </select>
        Title: <input type="text" asp-for="SearchString" />
        <input type="submit" value="Filter" />
    </p>
</form>

<table class="table">

    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Movie[0].Title)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Movie[0].ReleaseDate)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Movie[0].Genre)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Movie[0].Price)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Movie[0].Rating)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model.Movie)
        {
            <tr><td>
                @Html.DisplayFor(modelItem => item.Title)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.ReleaseDate)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Genre)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Price)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Rating)
            </td>
            <td>
                <a asp-page=".Edit" asp-route-id="@item.ID">Edit</a> |
                <a asp-page=".Details" asp-route-id="@item.ID">Details</a> |
                <a asp-page=".Delete" asp-route-id="@item.ID">Delete</a>
            </td>
        </tr>
        }
    </tbody>
</table>

```

Atualize as seguintes páginas:

- Adicione o campo `Rating` às páginas Excluir e Detalhes.
- Atualize `Create.cshtml` com um campo `Rating`.

- Adicione o campo `Rating` à página Editar.

O aplicativo não funcionará até que o BD seja atualizado para incluir o novo campo. Se for executado agora, o aplicativo gerará uma `SqlException`:

```
SqlException: Invalid column name 'Rating'.
```

Esse erro é causado devido à classe de modelo `Movie` atualizada ser diferente do esquema da tabela `Movie` do banco de dados. (Não há nenhuma coluna `Rating` na tabela de banco de dados.)

Existem algumas abordagens para resolver o erro:

1. Faça com que o Entity Framework remova automaticamente e recrie o banco de dados usando o novo esquema de classe de modelo. Essa abordagem é conveniente no início do ciclo de desenvolvimento; ela permite que você desenvolva rapidamente o modelo e o esquema de banco de dados juntos. A desvantagem é que você perde os dados existentes no banco de dados. Não use essa abordagem em um banco de dados de produção. A remoção do BD em alterações de esquema e o uso de um inicializador para propagar automaticamente o banco de dados com os dados de teste é muitas vezes uma maneira produtiva de desenvolver um aplicativo.
2. Modifique explicitamente o esquema do banco de dados existente para que ele corresponda às classes de modelo. A vantagem dessa abordagem é que você mantém os dados. Faça essa alteração manualmente ou criando um script de alteração de banco de dados.
3. Use as Migrações do Code First para atualizar o esquema de banco de dados.

Para este tutorial, use as Migrações do Code First.

Atualize a classe `SeedData` para que ela forneça um valor para a nova coluna. Uma alteração de amostra é mostrada abaixo, mas é recomendável fazer essa alteração em cada bloco `new Movie`.

```
context.Movie.AddRange(  
    new Movie  
    {  
        Title = "When Harry Met Sally",  
        ReleaseDate = DateTime.Parse("1989-2-12"),  
        Genre = "Romantic Comedy",  
        Price = 7.99M,  
        Rating = "R"  
    },
```

Consulte o [arquivo SeedData.cs concluído](#).

Compile a solução.

- [Visual Studio](#)
- [Visual Studio Code/Visual Studio para Mac](#)

### Adicionar uma migração para o campo de classificação

No menu **Ferramentas**, selecione **Gerenciador de Pacotes NuGet > Console do Gerenciador de Pacotes**.

No PMC, insira os seguintes comandos:

```
Add-Migration Rating  
Update-Database
```

O comando `Add-Migration` informa à estrutura:

- Compare o modelo `Movie` com o esquema de BD `Movie`.

- Crie um código para migrar o esquema de BD para o novo modelo.

O nome “Classificação” é arbitrário e é usado para nomear o arquivo de migração. É útil usar um nome significativo para o arquivo de migração.

O comando `Update-Database` informa à estrutura para aplicar as alterações de esquema no banco de dados.

Se você excluir todos os registros no BD, o inicializador propagará o BD e incluirá o campo `Rating`. Faça isso com os links Excluir no navegador ou no [SSOX](#) (Pesquisador de Objetos do SQL Server).

Outra opção é excluir o banco de dados e usar as migrações para recriar o banco de dados. Para excluir o banco de dados no SSOX:

- Selecione o banco de dados no SSOX.
- Clique com o botão direito do mouse no banco de dados e selecione *Excluir*.
- Marque **Fechar conexões existentes**.
- Selecione **OK**.
- No [PMC](#), atualize o banco de dados:

```
Update-Database
```

Execute o aplicativo e verifique se você pode criar/editar/exibir filmes com um campo `Rating`. Se o banco de dados não for propagado, defina um ponto de interrupção no método `SeedData.Initialize`.

[ANTERIOR: ADICIONAR](#)

[PESQUISA](#)

[PRÓXIMO: ADICIONAR](#)

[VALIDAÇÃO](#)

# Adicionar validação a uma Página Razor do ASP.NET Core

06/02/2019 • 15 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

Nesta seção, a lógica de validação é adicionada para o modelo `Movie`. As regras de validação são impostas sempre que um usuário cria ou edita um filme.

## Validação

Um princípio-chave do desenvolvimento de software é chamado **DRY** ("Don't Repeat Yourself"). O Razor Pages incentiva o desenvolvimento quando a funcionalidade é especificada uma vez e ela é refletida em todo o aplicativo. O DRY pode ajudar a:

- Reduzir a quantidade de código em um aplicativo.
- Fazer com que o código seja menos propenso a erros e mais fácil de ser testado e mantido.

O suporte de validação fornecido pelo Razor Pages e pelo Entity Framework é um bom exemplo do princípio DRY. As regras de validação são especificadas de forma declarativa em um único lugar (na classe de modelo) e as regras são impostas em qualquer lugar no aplicativo.

### Adicionando regras de validação ao modelo de filme

Abra o arquivo `Models/Movie.cs`. [DataAnnotations](#) fornece um conjunto interno de atributos de validação que são aplicados de forma declarativa a uma classe ou propriedade. DataAnnotations também contém atributos de formatação como `DataType`, que ajudam com a formatação e não fornecem validação.

Atualize a classe `Movie` para aproveitar os atributos de validação `Required`, `StringLength`, `RegularExpression` e `Range`.

```

public class Movie
{
    public int ID { get; set; }

    [StringLength(60, MinimumLength = 3)]
    [Required]
    public string Title { get; set; }

    [Display(Name = "Release Date")]
    [DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }

    [Range(1, 100)]
    [DataType(DataType.Currency)]
    [Column(TypeName = "decimal(18, 2)")]
    public decimal Price { get; set; }

    [RegularExpression(@"^([A-Z][a-zA-Z\s-]*$")]
    [Required]
    [StringLength(30)]
    public string Genre { get; set; }

    [RegularExpression(@"^([A-Z][a-zA-Z0-9\s-]*$")]
    [StringLength(5)]
    [Required]
    public string Rating { get; set; }
}

```

Os atributos de validação especificam o comportamento que é imposto nas propriedades do modelo:

- Os atributos `Required` e `MinimumLength` indicam que uma propriedade deve ter um valor. No entanto, nada impede que um usuário digite espaços em branco para atender à restrição de validação para um tipo de permitir valor nulo. Os [tipos de valor](#) que não permitem valores nulos (como `decimal`, `int`, `float` e `DateTime`) são inherentemente necessários e não precisam do atributo `Required`.
- O atributo `RegularExpression` limita os caracteres que o usuário pode inserir. No código anterior, `Genre` precisa começar com uma ou mais letras maiúsculas e seguir com zero ou mais letras, aspas simples ou duplas, caracteres de espaço em branco ou traço. `Rating` precisa começar com uma ou mais letras maiúsculas e seguir com zero ou mais letras, números, aspas simples ou duplas, caracteres de espaço em branco ou traço.
- O atributo `Range` restringe um valor a um intervalo especificado.
- O atributo `StringLength` define o tamanho máximo de uma cadeia de caracteres e, opcionalmente, o tamanho mínimo.

Ter as regras de validação automaticamente impostas pelo ASP.NET Core ajuda a tornar um aplicativo mais robusto. A validação automática em modelos ajuda a proteger o aplicativo porque você não precisa se lembrar de aplicá-los quando um novo código é adicionado.

### Interface do usuário do erro de validação no Razor Pages

Execute o aplicativo e navegue para Pages/Movies.

Selecione o link **Criar Novo**. Preencha o formulário com alguns valores inválidos. Quando a validação do lado do cliente do jQuery detecta o erro, ela exibe uma mensagem de erro.

Create - Movie

RpMovie Home Privacy

## Create

### Movie

Title

The field Title must be a string with a minimum length of 3 and a maximum length of 60.

Release Date

The Release Date field is required.

Genre

The field Genre must match the regular expression '^ [A-Z]+[a-zA-Z\s-]\*\$'.

Price

The field Price must be a number.

Rating

The field Rating must match the regular expression '^ [A-Z]+[a-zA-Z0-9\s-]\*\$'.

[Create](#)

[Back to List](#)

© 2019 - RazorPagesMovie - [Privacy](#)

This screenshot shows a browser window with a 'Create Movie' form. The 'Title' field contains 'a', which is too short. The 'Release Date' field is empty and required. The 'Genre' field contains 'a', which does not match the regular expression. The 'Price' field contains 'Dog', which is not a number. The 'Rating' field contains 'z', which does not match the regular expression. A 'Create' button is visible at the bottom left, and a 'Back to List' link is at the bottom right. The status bar at the bottom shows copyright information and a 'Privacy' link.

#### NOTE

Talvez você não consiga inserir vírgulas decimais em campos decimais. Para dar suporte à [validação do jQuery](#) para localidades de idiomas diferentes do inglês que usam uma vírgula (",") para um ponto decimal e formatos de data diferentes do inglês dos EUA, você deve tomar medidas para globalizar o aplicativo. Veja [Problema 4076 do GitHub](#) para obter instruções sobre como adicionar casas decimais.

Observe como o formulário renderizou automaticamente uma mensagem de erro de validação em cada campo que contém um valor inválido. Os erros são impostos no lado do cliente (usando o JavaScript e o jQuery) e no lado do servidor (quando um usuário tem o JavaScript desabilitado).

Uma vantagem significativa é que **nenhuma** alteração de código foi necessária nas páginas Criar ou Editar. Depois que DataAnnotations foi aplicado ao modelo, a interface do usuário de validação foi habilitada. O Razor Pages criado neste tutorial selecionou automaticamente as regras de validação (usando os atributos de validação nas propriedades da classe do modelo `Movie`). Validação do teste usando a página Editar: a mesma validação é aplicada.

Os dados de formulário não serão postados no servidor enquanto houver erros de validação do lado do cliente. Verifique se os dados de formulário não são postados por uma ou mais das seguintes abordagens:

- Coloque um ponto de interrupção no método `OnPostAsync`. Envie o formulário (selecione **Criar** ou **Salvar**). O ponto de interrupção nunca é atingido.

- Use a [ferramenta Fiddler](#).
- Use as ferramentas do desenvolvedor do navegador para monitorar o tráfego de rede.

## Validação do servidor

Quando o JavaScript está desabilitado no navegador, o envio do formulário com erros será postado no servidor.

(Opcional) Teste a validação do servidor:

- Desabilite o JavaScript no navegador. É possível fazer isso usando as ferramentas para desenvolvedores do navegador. Se você não conseguir desabilitar o JavaScript no navegador, tente outro navegador.
- Defina um ponto de interrupção no método `OnPostAsync` da página Criar ou Editar.
- Envie um formulário com erros de validação.
- Verifique se o estado do modelo é inválido:

```
if (!ModelState.IsValid)
{
    return Page();
}
```

O código a seguir mostra uma parte da página `Create.cshtml` gerada por scaffolding anteriormente no tutorial. Ele é usado pelas páginas Criar e Editar para exibir o formulário inicial e exibir o formulário novamente, em caso de erro.

```
<form method="post">
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
    <div class="form-group">
        <label asp-for="Movie.Title" class="control-label"></label>
        <input asp-for="Movie.Title" class="form-control" />
        <span asp-validation-for="Movie.Title" class="text-danger"></span>
    </div>
```

O [Auxiliar de Marcação de Entrada](#) usa os atributos de [DataAnnotations](#) e produz os atributos HTML necessários para a Validação do jQuery no lado do cliente. O [Auxiliar de Marcação de Validação](#) exibe erros de validação. Consulte [Validação](#) para obter mais informações.

As páginas Criar e Editar não têm nenhuma regra de validação. As regras de validação e as cadeias de caracteres de erro são especificadas somente na classe `Movie`. Essas regras de validação são aplicadas automaticamente ao Razor Page que edita o modelo `Movie`.

Quando a lógica de validação precisa ser alterada, ela é feita apenas no modelo. A validação é aplicada de forma consistente em todo o aplicativo (a lógica de validação é definida em um único lugar). A validação em um único lugar ajuda a manter o código limpo e facilita sua manutenção e atualização.

## Usando atributos DataType

Examine a classe `Movie`. O namespace `System.ComponentModel.DataAnnotations` fornece atributos de formatação, além do conjunto interno de atributos de validação. O atributo `DataType` é aplicado às propriedades `ReleaseDate` e `Price`.

```
[Display(Name = "Release Date")]
[DataType(DataType.Date)]
public DateTime ReleaseDate { get; set; }

[Range(1, 100)]
[DataType(DataType.Currency)]
public decimal Price { get; set; }
```

Os atributos `DataType` fornecem apenas dicas para que o mecanismo de exibição formate os dados (e fornece atributos como `<a>` para as URLs e `<a href="mailto:EmailAddress.com">` para o email). Use o atributo `RegularExpression` para validar o formato dos dados. O atributo `DataType` é usado para especificar um tipo de dados mais específico do que o tipo intrínseco de banco de dados. Os atributos `DataType` não são atributos de validação. No aplicativo de exemplo, apenas a data é exibida, sem a hora.

A Enumeração `DataType` fornece muitos tipos de dados, como Date, Time, PhoneNumber, Currency, EmailAddress e muito mais. O atributo `DataType` também pode permitir que o aplicativo forneça automaticamente recursos específicos a um tipo. Por exemplo, um link `mailto:` pode ser criado para `DataType.EmailAddress`. Um seletor de data pode ser fornecido para `DataType.Date` em navegadores que dão suporte a HTML5. Os atributos `DataType` emitem atributos `data-` HTML 5 (pronunciados "data dash") que são consumidos pelos navegadores HTML 5. Os atributos `DataType` **não** fornecem nenhuma validação.

`DataType.Date` não especifica o formato da data exibida. Por padrão, o campo de dados é exibido de acordo com os formatos padrão com base nas `CultureInfo` do servidor.

A anotação de dados `[Column(TypeName = "decimal(18, 2)")]` é necessária para que o Entity Framework Core possa mapear corretamente o `Price` para a moeda no banco de dados. Para obter mais informações, veja [Tipos de Dados](#).

O atributo `DisplayFormat` é usado para especificar explicitamente o formato de data:

```
[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
public DateTime ReleaseDate { get; set; }
```

A configuração `ApplyFormatInEditMode` especifica que a formatação deve ser aplicada quando o valor é exibido para edição. Não é recomendável ter esse comportamento em alguns campos. Por exemplo, em valores de moeda, você provavelmente não deseja que o símbolo de moeda seja exibido na interface do usuário de edição.

O atributo `DisplayFormat` pode ser usado por si só, mas geralmente é uma boa ideia usar o atributo `DataType`. O atributo `DataType` transmite a semântica dos dados, ao invés de apresentar como renderizá-lo em uma tela e oferece os seguintes benefícios que você não obtém com `DisplayFormat`:

- O navegador pode habilitar os recursos do HTML5 (por exemplo, mostrar um controle de calendário, o símbolo de moeda apropriado à localidade, links de email, etc.)
- Por padrão, o navegador renderizará os dados usando o formato correto de acordo com a localidade.
- O atributo `DataType` pode permitir que a estrutura ASP.NET Core escolha o modelo de campo correto para renderizar os dados. O `DisplayFormat`, se usado por si só, usa o modelo de cadeia de caracteres.

Observação: a validação do jQuery não funciona com os atributos `Range` e `DateTime`. Por exemplo, o seguinte código sempre exibirá um erro de validação do lado do cliente, mesmo quando a data estiver no intervalo especificado:

```
[Range(typeof(DateTime), "1/1/1966", "1/1/2020")]
```

Geralmente, não é uma boa prática compilar datas rígidas nos modelos e, portanto, o uso do atributo `Range` e de

`DateTime` não é recomendado.

O seguinte código mostra como combinar atributos em uma linha:

```
public class Movie
{
    public int ID { get; set; }

    [StringLength(60, MinimumLength = 3)]
    public string Title { get; set; }

    [Display(Name = "Release Date"), DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z''"\s-]*$"), Required, StringLength(30)]
    public string Genre { get; set; }

    [Range(1, 100), DataType(DataType.Currency)]
    [Column(TypeName = "decimal(18, 2)")]
    public decimal Price { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z0-9'''\s-]*$"), StringLength(5)]
    public string Rating { get; set; }
}
```

A [Introdução ao Razor Pages e ao EF Core](#) mostra operações mais avançadas do EF Core com o Razor Pages.

### Publicar no Azure

Para obter informações sobre como implantar no Azure, consulte [Tutorial: Compilar um aplicativo ASP.NET no Azure com o Banco de Dados SQL](#). Essas instruções são para um aplicativo ASP.NET, não um aplicativo ASP.NET Core, mas as etapas são as mesmas.

Obrigado por concluir esta introdução ao Razor Pages. A [Introdução ao Razor Pages e ao EF Core](#) é um excelente acompanhamento para este tutorial.

## Recursos adicionais

- [Auxiliares de marca em formulários no ASP.NET Core](#)
- [Globalização e localização no ASP.NET Core](#)
- [Auxiliares de Marca no ASP.NET Core](#)
- [Auxiliares de marca de autor no ASP.NET Core](#)

[ANTERIOR: ADICIONANDO UM NOVO](#)

[CAMPO](#)

# Métodos de filtro para Páginas Razor no ASP.NET Core

30/10/2018 • 7 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

Os filtros de página Razor [IPageFilter](#) e [IAsyncPageFilter](#) permitem que as Páginas Razor executem código antes e depois a execução de um manipulador de Página Razor. Os filtros de página Razor são semelhantes aos [filtros de ação de MVC do ASP.NET Core](#), exceto que eles não podem ser aplicados aos métodos do manipulador de cada página individual.

Filtros de página Razor:

- Executam o código depois que um método do manipulador é selecionado, mas antes que o model binding ocorra.
- Executam o código antes que o método do manipulador seja executado, após a conclusão do model binding.
- Executam o código após a execução do método do manipulador.
- Podem ser implementados em uma única página ou globalmente.
- Não podem ser aplicados a métodos do manipulador de uma página específica.

O código pode ser executado antes que um método do manipulador seja executado usando o construtor de página ou o middleware, mas somente os filtros de página Razor têm acesso a [HttpContext](#). Os filtros têm um parâmetro derivado [FilterContext](#), que fornece acesso a [HttpContext](#). Por exemplo, a amostra [Implementar um atributo de filtro](#) adiciona um cabeçalho à resposta, algo que não pode ser feito com construtores nem middlewares.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

Os filtros de página Razor fornecem os métodos a seguir, que podem ser aplicados globalmente ou no nível da página:

- Métodos síncronos:
  - [OnPageHandlerSelected](#): chamado depois que um método do manipulador é selecionado, mas antes que o model binding ocorra.
  - [OnPageHandlerExecuting](#): chamado antes que o método do manipulador seja executado, após a conclusão do model binding.
  - [OnPageHandlerExecuted](#): chamado depois que o método do manipulador é executado, antes do resultado da ação.
- Métodos assíncronos:
  - [OnPageHandlerSelectionAsync](#): chamado de forma assíncrona depois que o método do manipulador é selecionado, mas antes que o model binding ocorra.
  - [OnPageHandlerExecutionAsync](#): chamado de forma assíncrona, antes que o método do manipulador seja invocado, após a conclusão do model binding.

#### NOTE

Implemente **ou** a versão assíncrona ou a versão síncrona de uma interface de filtro, não ambas. Primeiro, a estrutura verifica se o filtro implementa a interface assíncrona e, se for esse o caso, a chama. Caso contrário, ela chama os métodos da interface síncrona. Se ambas as interfaces forem implementadas, somente os métodos assíncronos serão chamados. A mesma regra aplica-se para substituições em páginas. Implemente a versão síncrona ou a assíncrona da substituição, não ambas.

## Implementar filtros de página Razor globalmente

O código a seguir implementa `IAsyncPageFilter` :

```
using Microsoft.AspNetCore.Mvc.Filters;
using Microsoft.Extensions.Logging;
using System.Threading.Tasks;

namespace PageFilter.Filters
{
    public class SampleAsyncPageFilter : IAsyncPageFilter
    {
        private readonly ILogger _logger;

        public SampleAsyncPageFilter(ILogger logger)
        {
            _logger = logger;
        }

        public async Task OnPageHandlerSelectionAsync(
            PageHandlerSelectedContext context)
        {
            _logger.LogDebug("Global OnPageHandlerSelectionAsync called.");
            await Task.CompletedTask;
        }

        public async Task OnPageHandlerExecutionAsync(
            PageHandlerExecutingContext context,
            PageHandlerExecutionDelegate next)
        {
            _logger.LogDebug("Global OnPageHandlerExecutionAsync called.");
            await next.Invoke();
        }
    }
}
```

No código anterior, `ILogger` não é necessário. Ele é usado na amostra para fornecer informações de rastreamento do aplicativo.

O código a seguir habilita o `SampleAsyncPageFilter` na classe `Startup` :

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(options =>
    {
        options.Filters.Add(new SampleAsyncPageFilter(_logger));
    });
}
```

O código a seguir mostra a classe `Startup` completa:

```

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using PageFilter.Filters;

namespace PageFilter
{
    public class Startup
    {
        ILogger _logger;
        public Startup	ILoggerFactory loggerFactory, IConfiguration configuration)
        {
            _logger = loggerFactory.CreateLogger<GlobalFiltersLogger>();
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        public void ConfigureServices(IServiceCollection services)
        {
            services.AddMvc(options =>
            {
                options.Filters.Add(new SampleAsyncPageFilter(_logger));
            });
        }

        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Error");
                app.UseHsts();
            }

            app.UseHttpsRedirection();
            app.UseStaticFiles();
            app.UseCookiePolicy();

            app.UseMvc();
        }
    }
}

```

O código a seguir chama `AddFolderApplicationModelConvention` para aplicar o `SampleAsyncPageFilter` somente às páginas em `/subFolder`:

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc()
        .AddRazorPagesOptions(options =>
    {
        options.Conventions.AddFolderApplicationModelConvention(
            "/subFolder",
            model => model.Filters.Add(new SampleAsyncPageFilter(_logger)));
    });
}

```

O código a seguir implementa o `IPageFilter` síncrono:

```

using Microsoft.AspNetCore.Mvc.Filters;
using Microsoft.Extensions.Logging;

namespace PageFilter.Filters
{
    public class SamplePageFilter : IPageFilter
    {
        private readonly ILogger _logger;

        public SamplePageFilter(ILogger logger)
        {
            _logger = logger;
        }

        public void OnPageHandlerSelected(PageHandlerSelectedContext context)
        {
            _logger.LogDebug("Global sync OnPageHandlerSelected called.");
        }

        public void OnPageHandlerExecuting(PageHandlerExecutingContext context)
        {
            _logger.LogDebug("Global sync PageHandlerExecutingContext called.");
        }

        public void OnPageHandlerExecuted(PageHandlerExecutedContext context)
        {
            _logger.LogDebug("Global sync OnPageHandlerExecuted called.");
        }
    }
}

```

O código a seguir habilita o `SamplePageFilter`:

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(options =>
    {
        options.Filters.Add(new SamplePageFilter(_logger));
    });
}

```

## Implementar filtros de página Razor substituindo os métodos de filtro

O código a seguir substitui os filtros de página Razor síncronos:

```

using Microsoft.AspNetCore.Mvc.Filters;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Logging;

namespace PageFilter.Pages
{
    public class IndexModel : PageModel
    {
        private readonly ILogger _logger;

        public IndexModel(ILogger<IndexModel> logger)
        {
            _logger = logger;
        }
        public string Message { get; set; }

        public void OnGet()
        {
            _logger.LogDebug("IndexModel/OnGet");
        }

        public override void OnPageHandlerSelected(
            PageHandlerSelectedContext context)
        {
            _logger.LogDebug("IndexModel/OnPageHandlerSelected");
        }

        public override void OnPageHandlerExecuting(
            PageHandlerExecutingContext context)
        {
            Message = "Message set in handler executing";
            _logger.LogDebug("IndexModel/OnPageHandlerExecuting");
        }

        public override void OnPageHandlerExecuted(
            PageHandlerExecutedContext context)
        {
            _logger.LogDebug("IndexModel/OnPageHandlerExecuted");
        }
    }
}

```

## Implementar um atributo de filtro

O filtro baseado em atributo interno [OnResultExecutionAsync](#) pode tornar-se uma subclasse. O filtro a seguir adiciona um cabeçalho à resposta:

```

using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc.Filters;

namespace PageFilter.Filters
{
    public class AddHeaderAttribute : ResultFilterAttribute
    {
        private readonly string _name;
        private readonly string _value;

        public AddHeaderAttribute (string name, string value)
        {
            _name = name;
            _value = value;
        }

        public override void OnResultExecuting(ResultExecutingContext context)
        {
            context.HttpContext.Response.Headers.Add(_name, new string[] { _value });
        }
    }
}

```

O código a seguir se aplica ao atributo `AddHeader` :

```

[AddHeader("Author", "Rick")]
public class ContactModel : PageModel
{
    private readonly ILogger _logger;

    public ContactModel(ILogger<ContactModel> logger)
    {
        _logger = logger;
    }
    public string Message { get; set; }

    public async Task OnGetAsync()
    {
        Message = "Your contact page.";
        _logger.LogDebug("Contact/OnGet");
        await Task.CompletedTask;
    }
}

```

Confira [Substituindo a ordem padrão](#) para obter instruções sobre a substituição da ordem.

Confira [Cancelamento e curto-circuito](#) para obter instruções para causar um curto-circuito no pipeline do filtro por meio de um filtro.

## Autorizar o atributo de filtro

O atributo [Authorize](#) pode ser aplicado a um `PageModel` :

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace PageFilter.Pages
{
    [Authorize]
    public class ModelWithAuthFilterModel : PageModel
    {
        public IActionResult OnGet() => Page();
    }
}
```

# Criar a interface do usuário reutilizável usando o projeto de biblioteca de classes Razor no ASP.NET Core

09/12/2018 • 10 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

Modos de exibição do Razor, páginas, controladores, modelos de página, [Componentes de exibição](#) e modelos de dados podem ser inseridos em uma RCL (Biblioteca de Classes Razor). A RCL pode ser empacotada e reutilizada. Os aplicativos podem incluir a RCL e substituir as exibições e as páginas que ela contém. Quando uma exibição, uma exibição parcial ou uma página Razor for encontrada no aplicativo Web e na RCL, a marcação Razor (arquivo .cshtml) no aplicativo Web terá precedência.

Este recurso requer [SDK do .NET Core 2.1 ou posteriores](#)

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Criar uma biblioteca de classes contendo a interface do usuário do Razor

- [Visual Studio](#)
  - [CLI do .NET Core](#)
- No menu **Arquivo** do Visual Studio, selecione **Novo > Projeto**.
  - Selecione **Aplicativo Web ASP.NET Core**.
  - Dê um nome à biblioteca (por exemplo, "RazorClassLib") > **OK**. Para evitar uma colisão de nome de arquivo com a biblioteca de exibição gerada, verifique se o nome da biblioteca não termina em `.Views`.
  - Verifique se o **ASP.NET Core 2.1** ou posterior está selecionado.
  - Selecione **Biblioteca de Classes Razor** > **OK**.

Uma biblioteca de classes Razor tem o seguinte arquivo de projeto:

```
<Project Sdk="Microsoft.NET.Sdk.Razor">

<PropertyGroup>
    <TargetFramework>netstandard2.0</TargetFramework>
</PropertyGroup>

<ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Mvc" Version="2.1.2" />
</ItemGroup>

</Project>
```

Adicione arquivos Razor na RCL.

Os modelos do ASP.NET Core, suponha que o conteúdo da RCL está no `áreas` pasta. Ver [layout de páginas RCL](#) para criar uma RCL que expõe conteúdo em `~/Pages` em vez de `~/Areas/Pages`.

## Referenciando o conteúdo da Biblioteca de Classes Razor

A RCL pode ser referenciada por:

- Pacote do NuGet. Confira [Criando pacotes do NuGet](#), `dotnet add package` e [Criar e publicar um pacote do NuGet](#).
- `{ProjectName}.csproj`. Confira [dotnet-add reference](#).

## Passo a passo: Criar um projeto de biblioteca de classes Razor e usar um projeto de Páginas Razor

Você pode baixar o [projeto completo](#) e testá-lo em vez de criá-lo. O download de exemplo contém um código adicional e links que facilitam o teste do projeto. Você pode deixar comentários [nesse problema do GitHub](#) com suas opiniões sobre os exemplos de download em relação às instruções passo a passo.

### Testar o aplicativo de download

Se você não tiver baixado o aplicativo concluído e preferir criar o projeto do passo a passo, vá para a [próxima seção](#).

- [Visual Studio](#)
- [CLI do .NET Core](#)

Abra o arquivo `.sln` no Visual Studio. Execute o aplicativo.

Siga as instruções em [Testar WebApp1](#)

## Criar uma Biblioteca de Classes Razor

Nesta seção, uma RCL (Biblioteca de Classes Razor) é criada. Arquivos Razor são adicionados à RCL.

- [Visual Studio](#)
- [CLI do .NET Core](#)

Crie o projeto da RCL:

- No menu **Arquivo** do Visual Studio, selecione **Novo > Projeto**.
- Selecione **Aplicativo Web ASP.NET Core**.
- Nomeie o aplicativo **RazorUIClassLib > Okey**.
- Verifique se o **ASP.NET Core 2.1** ou posterior está selecionado.
- Selecione **Biblioteca de Classes Razor > OK**.
- Adicione um arquivo Razor de exibição parcial chamado `RazorUIClassLib\Areas\MyFeature\Pages\Shared\_Message.cshtml`.

### Adicionar pastas e arquivos Razor ao projeto

- Substitua a marcação em `RazorUIClassLib\Areas\MyFeature\Pages\Shared\_Message.cshtml` pelo código a seguir:

```
<h3>_Message.cshtml partial view.</h3>
<p>RazorUIClassLib\Areas\MyFeature\Pages\Shared\_Message.cshtml</p>
```

- Substitua a marcação em `RazorUIClassLib\Areas\MyFeature\Pages\Page1.cshtml` pelo código a seguir:

```
@page
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

<h2>Hello from a Razor UI class library!</h2>
<p> From RazorUIClassLib\Areas\MyFeature\Pages\Page1.cshtml</p>

<partial name="_Message" />
```

`@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers` é necessário para usar a exibição parcial (`<partial name="_Message" />`). Em vez de incluir a diretiva `@addTagHelper`, você pode adicionar um arquivo `_ViewImports.cshtml`. Por exemplo:

```
dotnet new viewimports -o RazorUIClassLib/Areas/MyFeature/Pages
```

Para obter mais informações sobre `viewimports.cshtml`, consulte [importando diretivas compartilhadas](#)

- Crie a biblioteca de classes para verificar se não há nenhum erro de compilador:

```
dotnet build RazorUIClassLib
```

A saída do build contém `RazorUIClassLib.dll` e `RazorUIClassLib.Views.dll`. `RazorUIClassLib.Views.dll` contém o conteúdo Razor compilado.

### Use a biblioteca da interface do usuário do Razor de um projeto Páginas Razor

- [Visual Studio](#)
- [CLI do .NET Core](#)

Crie aplicativo Web Páginas Razor:

- No **Gerenciador de Soluções**, clique com o botão direito do mouse na solução > **Adicionar > Novo Projeto**.
- Selecione **Aplicativo Web ASP.NET Core**.
- Nomeie o aplicativo como **WebApp1**.
- Verifique se o **ASP.NET Core 2.1** ou posterior está selecionado.
- Selecione **Aplicativo Web > OK**.
- No **Gerenciador de Soluções**, clique com o botão direito do mouse em **WebApp1** e selecione **Definir como projeto de inicialização**.
- No **Gerenciador de Soluções**, clique com o botão direito do mouse em **WebApp1** e selecione **Dependências de Build > Dependências do Projeto**.
- Marque **RazorUIClassLib** como uma dependência de **WebApp1**.
- No **Gerenciador de Soluções**, clique com o botão direito do mouse em **WebApp1** e selecione **Adicionar > Referência**.
- Na caixa de diálogo **Gerenciador de Referências**, marque **RazorUIClassLib > OK**.

Execute o aplicativo.

### Testar o WebApp1

Verifique se a biblioteca de classes da interface do usuário do Razor está sendo usada.

- Navegue para `/MyFeature/Page1`.

## Substituir exibições, exibições parciais e páginas

Quando uma exibição, uma exibição parcial ou uma Página Razor for encontrada no aplicativo Web e na Biblioteca de Classes Razor, a marcação Razor (arquivo `.cshtml`) no aplicativo Web terá precedência. Por exemplo, adicione `WebApp1/Areas/MyFeature/Pages/Page1.cshtml` ao WebApp1, e a Page1 ao WebApp1 terá precedência sobre a Page1 na biblioteca de classes Razor.

No download de exemplo, renomeie `WebApp1/Areas/MyFeature2` como `WebApp1/Areas/MyFeature` para testar a precedência.

Copie a exibição parcial `RazorUIClassLib/Areas/MyFeature/Pages/Shared/_Message.cshtml` para `WebApp1/Areas/MyFeature/Pages/Shared/_Message.cshtml`. Atualize a marcação para indicar o novo local. Crie e execute o aplicativo para verificar se a versão da parcial do aplicativo está sendo usada.

### Layout de páginas RCL

A RCL como se fosse parte do aplicativo web de conteúdo de referência `páginas` pasta, crie o projeto da RCL com a seguinte estrutura de arquivo:

- `RazorUIClassLib/páginas`
- `RazorUIClassLib/páginas/Shared`

Suponha `RazorUIClassLib/páginas/Shared` contém dois arquivos parciais: `_Header.cshtml` e `_Footer.cshtml`. O `<partial>` marca pode ser adicionadas ao `layout.cshtml` arquivo:

```
<body>
    <partial name="_Header">
        @RenderBody()
    <partial name="_Footer">
</body>
```

# Convenções de rota e aplicativo das Páginas do Razor no ASP.NET Core

30/10/2018 • 33 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

Saiba como usar a página [convenções do provedor de modelo de rota e aplicativo](#) para controlar o roteamento, a descoberta e o processamento de páginas nos aplicativos Páginas do Razor.

Quando precisar configurar rotas de página personalizadas para páginas individuais, configure o roteamento para páginas com a [convenção AddPageRoute](#) descrita mais adiante neste tópico.

Para especificar uma rota de página, adicione segmentos de rota ou adicionar parâmetros a uma rota, use a página `@page` diretiva. Para obter mais informações, consulte [rotas personalizadas](#).

Há palavras reservadas não podem ser usadas como segmentos de rota ou nomes de parâmetro. Para obter mais informações, consulte [roteamento: roteamentos nomes reservados](#).

[Exibir ou baixar código de exemplo \(como baixar\)](#)

CENÁRIO	A AMOSTRA EXPLICA...
<a href="#">Convenções de modelo</a>  Conventions.Add <ul style="list-style-type: none"><li>• <code>IPageRouteModelConvention</code></li><li>• <code>IPageRouteApplicationModelConvention</code></li></ul>	Adicione um cabeçalho e um modelo de rota às páginas de um aplicativo.
<a href="#">Convenções de ação da rota de página</a>  <ul style="list-style-type: none"><li>• <code>AddFolderRouteModelConvention</code></li><li>• <code>AddPageRouteModelConvention</code></li><li>• <code>AddPageRoute</code></li></ul>	Adicione um modelo de rota às páginas em uma pasta e a uma única página.
<a href="#">Convenções de ação do modelo de página</a>  <ul style="list-style-type: none"><li>• <code>AddFolderApplicationModelConvention</code></li><li>• <code>AddPageRouteApplicationModelConvention</code></li><li>• <code>ConfigureFilter</code> (classe de filtro, expressão lambda ou alocador de filtro)</li></ul>	Adicione um cabeçalho às páginas em uma pasta, adicione um cabeçalho a uma única página e configure um <a href="#">alocador de filtro</a> para adicionar um cabeçalho às páginas de um aplicativo.
<a href="#">Provedor de modelo de aplicativo de página padrão</a>	Substitua o provedor de modelo de página padrão para alterar as convenções de nomes de manipulador.
CENÁRIO	A AMOSTRA EXPLICA...
<a href="#">Convenções de modelo</a>  Conventions.Add <ul style="list-style-type: none"><li>• <code>IPageRouteModelConvention</code></li><li>• <code>IPageRouteApplicationModelConvention</code></li><li>• <code>IPageRouteHandlerModelConvention</code></li></ul>	Adicione um cabeçalho e um modelo de rota às páginas de um aplicativo.

CENÁRIO	A AMOSTRA EXPLICA...
<p><a href="#">Convenções de ação da rota de página</a></p> <ul style="list-style-type: none"> <li>• <code>AddFolderRouteModelConvention</code></li> <li>• <code>AddPageRouteModelConvention</code></li> <li>• <code>AddPageRoute</code></li> </ul>	Adicione um modelo de rota às páginas em uma pasta e a uma única página.
<p><a href="#">Convenções de ação do modelo de página</a></p> <ul style="list-style-type: none"> <li>• <code>AddFolderApplicationModelConvention</code></li> <li>• <code>AddPageApplicationModelConvention</code></li> <li>• <code>ConfigureFilter</code> (classe de filtro, expressão lambda ou alocador de filtro)</li> </ul>	Adicione um cabeçalho às páginas em uma pasta, adicione um cabeçalho a uma única página e configure um <a href="#">alocador de filtro</a> para adicionar um cabeçalho às páginas de um aplicativo.
<p><a href="#">Provedor de modelo de aplicativo de página padrão</a></p>	Substitua o provedor de modelo de página padrão para alterar as convenções de nomes de manipulador.

As convenções de Páginas Razor são adicionadas e configuradas usando o método de extensão `AddRazorPagesOptions` para `AddMvc` na coleção de serviços na classe `Startup`. Os exemplos de convenção a seguir estão descritos mais adiante neste tópico:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc()
        .AddRazorPagesOptions(options =>
    {
        options.Conventions.Add( ... );
        options.Conventions.AddFolderRouteModelConvention("/OtherPages", model => { ... });
        options.Conventions.AddPageRouteModelConvention("/About", model => { ... });
        options.Conventions.AddPageRoute("/Contact", "TheContactPage/{text?}");
        options.Conventions.AddFolderApplicationModelConvention("/OtherPages", model => { ... });
        options.Conventions.AddPageApplicationModelConvention("/About", model => { ... });
        options.Conventions.ConfigureFilter(model => { ... });
        options.Conventions.ConfigureFilter( ... );
    });
}
```

## Ordem de rota

As rotas especificam um [Order](#) para processamento (rota correspondente).

PEDIDO	COMPORTAMENTO
-1	A rota é processada antes de outras rotas são processadas.
0	Ordem não for especificada (valor padrão). Não atribui <code>Order</code> ( <code>order = null</code> ) usa como padrão a rota <code>Order</code> como 0 (zero) para processamento.
1, 2, ... n	Especifica a ordem de processamento de rota.

Processamento de rota é estabelecido por convenção:

- As rotas são processadas em ordem sequencial (-1, 0, 1, 2, ... n).
- Quando as rotas têm a mesma `order`, a maioria das rota específica é correspondida primeiro, seguido por rotas menos específicas.

- Quando as rotas com o mesmo `Order` e o mesmo número de parâmetros corresponde a uma URL de solicitação, as rotas são processadas na ordem em que eles forem adicionados à [PageConventionCollection](#).

Se possível, evite dependendo de uma ordem de processamento de rota estabelecido. Em geral, o roteamento seleciona a rota correta com a URL correspondente. Se você deve definir a rota `Order` propriedades para rotear solicitações corretamente, o esquema do aplicativo roteamento é provavelmente confuso para os clientes e frágil para manter. Para simplificar o esquema do aplicativo roteamento de busca. O aplicativo de exemplo requer uma rota explícita de processamento de pedido para demonstrar os vários cenários de roteamento usando um único aplicativo. No entanto, você deve tentar evitar a prática de rota de configuração `Order` em aplicativos de produção.

Roteamento do Razor Pages e do controlador do MVC compartilham uma implementação. Informações sobre a ordem de rota nos tópicos MVC estão disponíveis em [roteamento para ações do controlador: ordenação de rotas de atributo](#).

## Convenções de modelo

Adicione um representante para [IPageConvention](#) para adicionar as [convenções de modelo](#) que se aplicam às Páginas Razor.

### Adicionar uma convenção de modelo de rota para todas as páginas

Use [Convenções](#) para criar e adicionar um [IPageRouteModelConvention](#) à coleção de instâncias de [IPageConvention](#) que são aplicadas durante a construção do modelo de rota de página.

O aplicativo de exemplo adiciona um modelo de rota `{globalTemplate?}` a todas as páginas no aplicativo:

```
public class GlobalTemplatePageRouteModelConvention
    : IPageRouteModelConvention
{
    public void Apply(PageRouteModel model)
    {
        var selectorCount = model.Selectors.Count;
        for (var i = 0; i < selectorCount; i++)
        {
            var selector = model.Selectors[i];
            model.Selectors.Add(new SelectorModel
            {
                AttributeRouteModel = new AttributeRouteModel
                {
                    Order = 1,
                    Template = AttributeRouteModel.CombineTemplates(
                        selector.AttributeRouteModel.Template,
                        "{globalTemplate?}"),
                }
            });
        }
    }
}
```

A propriedade `Order` do [AttributeRouteModel](#) é definida como `1`. Isso garante que a rota correspondência de comportamento no aplicativo de exemplo a seguir:

- Um modelo de rota para `TheContactPage/{text?}` é adicionado posteriormente no tópico. A rota de página Contact tem uma ordem padrão de `null` (`Order = 0`), para que ele corresponda ao antes do `{globalTemplate?}` modelo de rota.
- Um `{aboutTemplate?}` modelo de rota é adicionado posteriormente no tópico. O modelo `{aboutTemplate?}` recebe uma `Order` de `2`. Quando a página About é solicitada no `/About/RouteDataValue`, "RouteDataValue" é carregado no `RouteData.Values["globalTemplate"]` (`Order = 1`) e não `RouteData.Values["aboutTemplate"]` (

`Order = 2`) devido à configuração da propriedade `Order`.

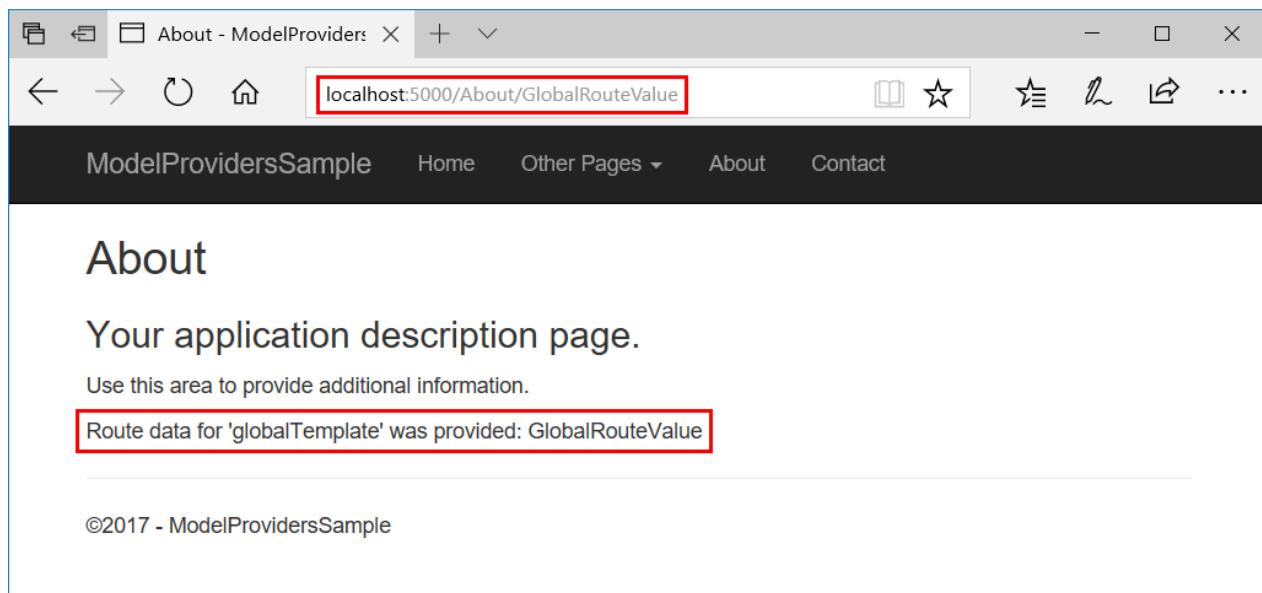
- Um `{otherPagesTemplate?}` modelo de rota é adicionado posteriormente no tópico. O modelo `{otherPagesTemplate?}` recebe uma `Order` de `2`. Quando qualquer página na `páginas/OtherPages` pasta for solicitada com um parâmetro de rota (por exemplo, `/OtherPages/Page1/RouteDataValue`), "RouteDataValue" é carregado no `RouteData.Values["globalTemplate"]` (`order = 1`) e não `RouteData.Values["otherPagesTemplate"]` (`order = 2`) devido à configuração de `Order` propriedade.

Sempre que possível, não defina a `Order`, que resulta em `Order = 0`. Dependem de roteamento para selecionar a rota correta.

As opções de Páginas Razor, como a adição de [Convenções](#), são adicionados quando o MVC é adicionado à coleção de serviços em `Startup.ConfigureServices`. Para obter um exemplo, confira o [aplicativo de exemplo](#).

```
options.Conventions.Add(new GlobalTemplatePageRouteModelConvention());
```

Solicite a página About da amostra em `localhost:5000/About/GlobalRouteValue` e inspecione o resultado:



### Adicionar uma convenção de modelo de aplicativo para todas as páginas

Use [Convenções](#) para criar e adicionar um `IPageApplicationModelConvention` à coleção de instâncias de `IPageConvention` que são aplicadas durante a construção do modelo de aplicativo de página.

Para demonstrar isso e outras convenções mais adiante no tópico, o aplicativo de exemplo inclui uma classe `AddHeaderAttribute`. O construtor de classe aceita uma cadeia de caracteres `name` e uma matriz de cadeia de caracteres `values`. Esses valores são usados em seu método `OnResultExecuting` para definir um cabeçalho de resposta. A classe completa é mostrada na seção [Convenções de ação do modelo de página](#) mais adiante no tópico.

O aplicativo de exemplo usa a classe `AddHeaderAttribute` para adicionar um cabeçalho, `GlobalHeader`, a todas as páginas no aplicativo:

```
public class GlobalHeaderPageApplicationModelConvention
    : IPageApplicationModelConvention
{
    public void Apply(PageApplicationModel model)
    {
        model.Filters.Add(new AddHeaderAttribute(
            "GlobalHeader", new string[] { "Global Header Value" }));
    }
}
```

Startup.cs:

```
options.Conventions.Add(new GlobalHeaderPageApplicationModelConvention());
```

Solicite a página About da amostra em `localhost:5000/About` e inspecione os cabeçalhos para exibir o resultado:

▼ Response Headers [view source](#)

**AboutHeader:** About Header Value  
**Content-Type:** text/html; charset=utf-8  
**Date:** Thu, 19 Oct 2017 21:09:07 GMT  
**FilterFactoryHeader:** Filter Factory Header Value 1  
**FilterFactoryHeader:** Filter Factory Header Value 2  
**GlobalHeader:** Global Header Value    
**Server:** Kestrel  
**Transfer-Encoding:** chunked

### Adicionar uma convenção de modelo de manipulador a todas as páginas

Use [Convenções](#) para criar e adicionar um [IPageHandlerModelConvention](#) à coleção de instâncias de [IPageConvention](#) que são aplicadas durante a construção do modelo de manipulador de página.

```
public class GlobalPageHandlerModelConvention
    : IPageHandlerModelConvention
{
    public void Apply(PageHandlerModel model)
    {
        ...
    }
}
```

Startup.ConfigureServices :

```
services.AddMvc()
    .AddRazorPagesOptions(options =>
    {
        options.Conventions.Add(new GlobalPageHandlerModelConvention());
    });
}
```

## Convenções de ação da rota de página

O provedor de modelo de rota padrão derivado de [IPageRouteModelProvider](#) invoca convenções que foram projetadas para fornecer pontos de extensibilidade para configuração de rotas de página.

### Convenção de modelo de rota de pasta

Use [AddFolderRouteModelConvention](#) para criar e adicionar uma [IPageRouteModelConvention](#) que invoca uma ação no [PageRouteModel](#) para todas as páginas da pasta especificada.

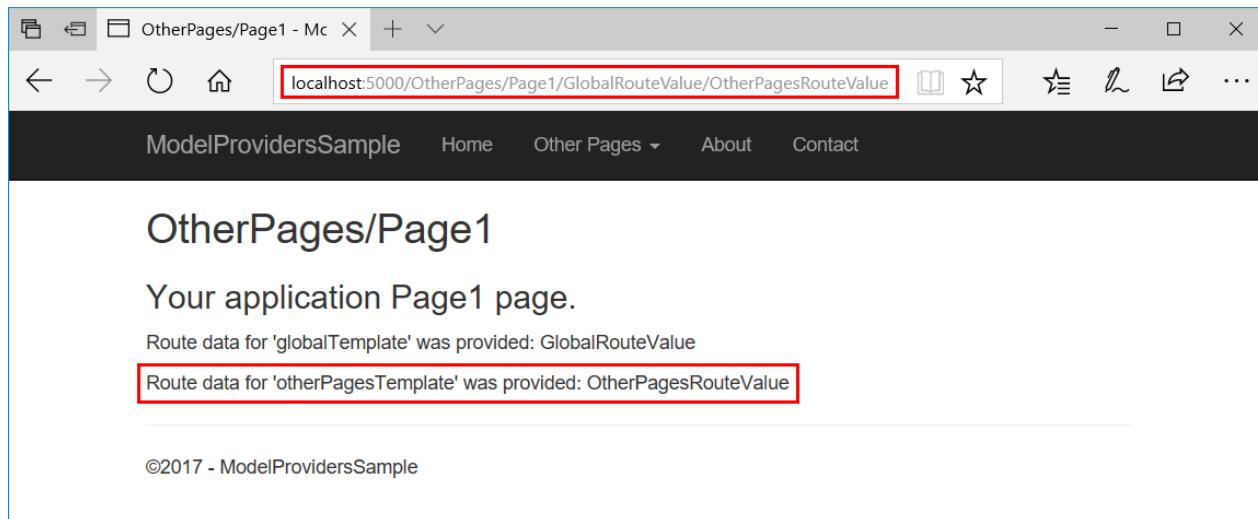
O aplicativo de exemplo usa `AddFolderRouteModelConvention` para adicionar um modelo de rota `{otherPagesTemplate?}` às páginas da pasta `OtherPages`:

```
options.Conventions.AddFolderRouteModelConvention("/OtherPages", model =>
{
    var selectorCount = model.Selectors.Count;
    for (var i = 0; i < selectorCount; i++)
    {
        var selector = model.Selectors[i];
        model.Selectors.Add(new SelectorModel
        {
            AttributeRouteModel = new AttributeRouteModel
            {
                Order = 2,
                Template = AttributeRouteModel.CombineTemplates(
                    selector.AttributeRouteModel.Template,
                    "{otherPagesTemplate?}"),
            }
        });
    }
});
```

A propriedade `Order` do `AttributeRouteModel` é definida como `2`. Isso garante que o modelo para `{globalTemplate?}` (definido anteriormente no tópico para `1`) tem prioridade para os dados de rota a primeira posição de valor quando um único valor de rota é fornecido. Se uma página na `páginas/OtherPages` pasta for solicitada com um valor de parâmetro de rota (por exemplo, `/OtherPages/Page1/RouteHeaderValue`), "RouteHeaderValue" é carregado no `RouteData.Values["globalTemplate"]` (`Order = 1`) e não `RouteData.Values["otherPagesTemplate"]` (`Order = 2`) devido à configuração de `Order` propriedade.

Sempre que possível, não defina a `Order`, que resulta em `Order = 0`. Dependem de roteamento para selecionar a rota correta.

Solicite a página `Page1` da amostra em `localhost:5000/OtherPages/Page1/GlobalRouteValue/OtherPagesRouteValue` e inspecione o resultado:



## Convenção de modelo de rota de página

Use `AddPageRouteModelConvention` para criar e adicionar uma `IPageRouteModelConvention` que invoca uma ação no `PageRouteModel` para a página com o nome especificado.

O aplicativo de exemplo usa `AddPageRouteModelConvention` para adicionar um modelo de rota `{aboutTemplate?}` à página `About`:

```

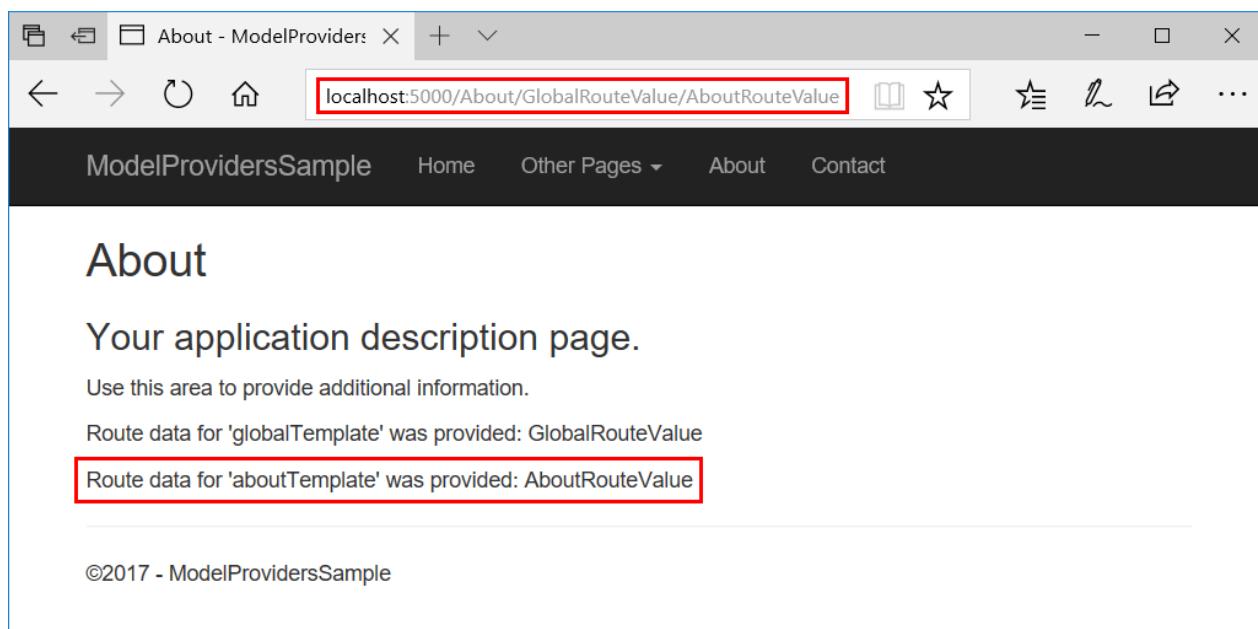
options.Conventions.AddPageRouteModelConvention("/About", model =>
{
    var selectorCount = model.Selectors.Count;
    for (var i = 0; i < selectorCount; i++)
    {
        var selector = model.Selectors[i];
        model.Selectors.Add(new SelectorModel
        {
            AttributeRouteModel = new AttributeRouteModel
            {
                Order = 2,
                Template = AttributeRouteModel.CombineTemplates(
                    selector.AttributeRouteModel.Template,
                    "{aboutTemplate?}"),
            }
        });
    }
});

```

A propriedade `Order` do `AttributeRouteModel` é definida como `2`. Isso garante que o modelo para `{globalTemplate?}` (definido anteriormente no tópico para `1`) tem prioridade para os dados de rota a primeira posição de valor quando um único valor de rota é fornecido. Se a página About é solicitada com um valor de parâmetro de rota no `/About/RouteDataValue`, "RouteDataValue" é carregado no `RouteData.Values["globalTemplate"]` (`Order = 1`) e não `RouteData.Values["aboutTemplate"]` (`Order = 2`) devido à configuração de `Order` propriedade.

Sempre que possível, não defina a `order`, que resulta em `Order = 0`. Dependem de roteamento para selecionar a rota correta.

Solicite a página About da amostra em `localhost:5000/About/GlobalRouteValue/AboutRouteValue` e inspecione o resultado:



## Usar um transformador de parâmetro para personalizar rotas de página

Rotas de página geradas pelo ASP.NET Core podem ser personalizadas usando um transformador de parâmetro. Um transformador de parâmetro implementa `IOutboundParameterTransformer` e transforma o valor dos parâmetros. Por exemplo, um transformador de parâmetro `SlugifyParameterTransformer` personalizado muda o valor de rota `SubscriptionManagement` para `subscription-management`.

O `PageRouteTransformerConvention` convenção de modelo de rota de página aplica-se um transformador de parâmetro para os segmentos de nome de arquivo e pasta de rotas de página gerada automaticamente em um aplicativo. Por exemplo, as páginas do Razor arquivo cada `/Pages/SubscriptionManagement/ViewAll.cshtml` teria sua rota do reescrita `/SubscriptionManagement/ViewAll` para `/subscription-management/view-all`.

`PageRouteTransformerConvention` apenas transforma os segmentos gerados automaticamente de uma rota de página que vêm do nome de arquivo e pasta páginas do Razor. Ele não transformará os segmentos de rota adicionados com o `@page` diretiva. A convenção não transformará rotas adicionadas por [AddPageRoute](#).

O `PageRouteTransformerConvention` está registrado como uma opção na `Startup.ConfigureServices`:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc()
        .AddRazorPagesOptions(options =>
    {
        options.Conventions.Add(
            new PageRouteTransformerConvention(
                new SlugifyParameterTransformer()));
    });
}

public class SlugifyParameterTransformer : IOutboundParameterTransformer
{
    public string TransformOutbound(object value)
    {
        if (value == null) { return null; }

        // Slugify value
        return Regex.Replace(value.ToString(), "([a-z])([A-Z])", "$1-$2").ToLower();
    }
}
```

## Configurar uma rota de página

Use [AddPageRoute](#) para configurar uma rota para uma página no caminho de página especificado. Os links gerados para a página usam a rota especificada. `AddPageRoute` usa `AddPageRouteModelConvention` para estabelecer a rota.

O aplicativo de exemplo cria uma rota para `/TheContactPage` para `Contact.cshtml`:

```
options.Conventions.AddPageRoute("/Contact", "TheContactPage/{text?}");
```

A página Contact também pode ser acessada em `/Contact` por meio de sua rota padrão.

A rota personalizada do aplicativo de exemplo para a página Contact permite um segmento de rota `text` opcional (`{text?}`). A página também inclui esse segmento opcional em sua diretiva `@page`, caso o visitante accesse a página em sua rota `/Contact`:

```

@page "{text?}"
@model ContactModel
 @{
    ViewData["Title"] = "Contact";
}

<h1>@ViewData["Title"]</h1>
<h2>@Model.Message</h2>

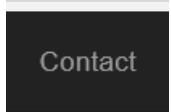
<address>
    One Microsoft Way<br>
    Redmond, WA 98052-6399<br>
    <abbr title="Phone">P:</abbr>
    425.555.0100
</address>

<address>
    <strong>Support:</strong> <a href="mailto:Support@example.com">Support@example.com</a><br>
    <strong>Marketing:</strong> <a href="mailto:Marketing@example.com">Marketing@example.com</a>
</address>

<p>@Model.RouteDataTextTemplateValue</p>

```

Observe que a URL gerada para o link **Contato** na página renderizada reflete a rota atualizada:

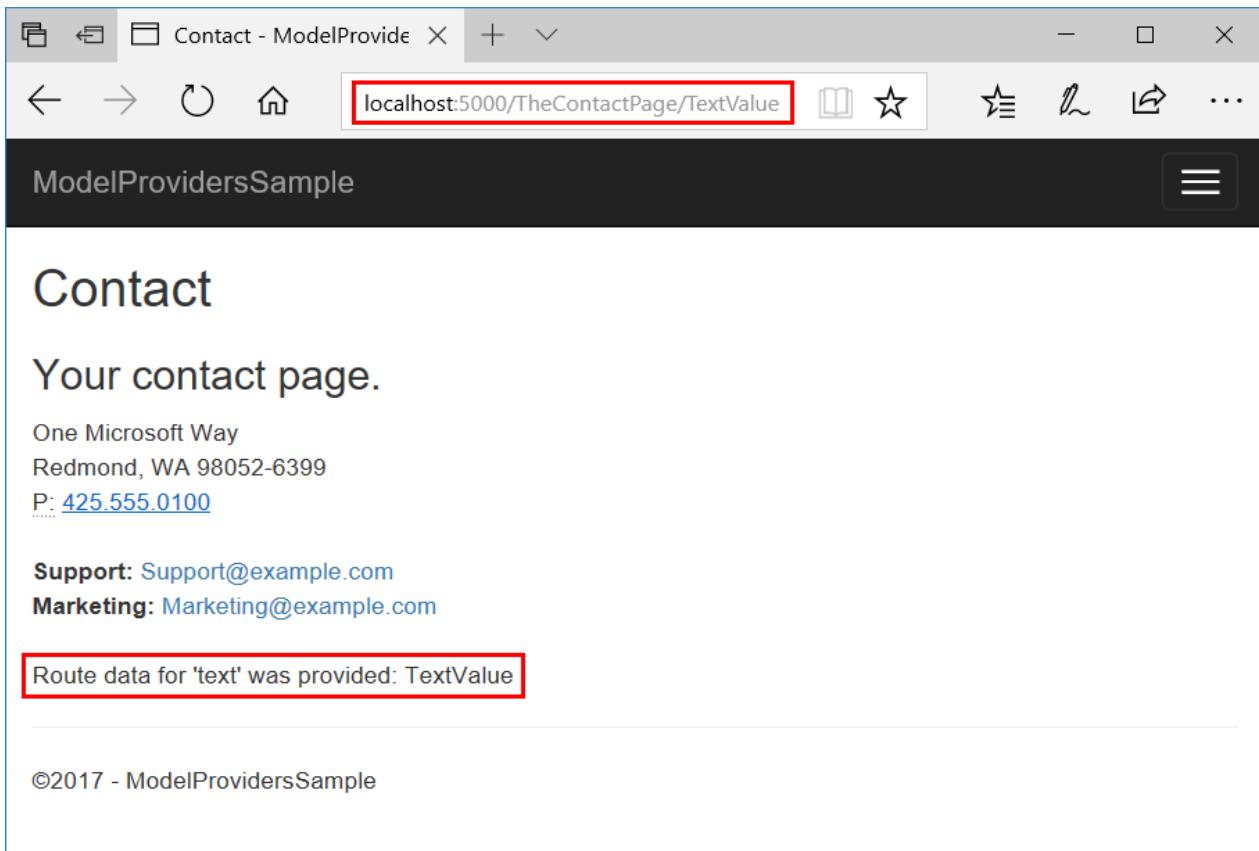


```

▶ <li>...</li>
▼ <li>
    <a href="/TheContactPage">Contact</a>
</li>
::after
</ul>

```

Visite a página Contact em sua rota comum, `/Contact`, ou na rota personalizada, `/TheContactPage`. Se você fornecer um segmento de rota `text` adicional, a página mostrará o segmento codificado em HTML fornecido:



## Convenções de ação do modelo de página

O provedor de modelo de página padrão que implementa [IPageApplicationModelProvider](#) invoca convenções que foram projetadas para fornecer pontos de extensibilidade para configuração de modelos de página. Essas convenções são úteis ao criar e modificar cenários de descoberta e processamento de página.

Para os exemplos desta seção, o aplicativo de exemplo usa uma classe [AddHeaderAttribute](#), que é um [ResultFilterAttribute](#), aplicável a um cabeçalho de resposta:

```
public class AddHeaderAttribute : ResultFilterAttribute
{
    private readonly string _name;
    private readonly string[] _values;

    public AddHeaderAttribute(string name, string[] values)
    {
        _name = name;
        _values = values;
    }

    public override void OnResultExecuting(ResultExecutingContext context)
    {
        context.HttpContext.Response.Headers.Add(_name, _values);
        base.OnResultExecuting(context);
    }
}
```

Usando convenções, a amostra explica como aplicar o atributo a todas as páginas de uma pasta e a uma única página.

### Convenção de modelo de aplicativo de pasta

Use [AddFolderApplicationModelConvention](#) para criar e adicionar uma [IPageApplicationModelConvention](#) que invoca uma ação em instâncias de [PageApplicationModel](#) em todas as páginas na pasta especificada.

A amostra explica o uso de `AddFolderApplicationModelConvention` adicionando um cabeçalho, `OtherPagesHeader`, às páginas dentro da pasta `OtherPages` do aplicativo:

```
options.Conventions.AddFolderApplicationModelConvention("/OtherPages", model =>
{
    model.Filters.Add(new AddHeaderAttribute(
        "OtherPagesHeader", new string[] { "OtherPages Header Value" }));
});
```

Solicite a página `Page1` da amostra em `localhost:5000/OtherPages/Page1` e inspecione os cabeçalhos para exibir o resultado:

▼ Response Headers [view source](#)

**Content-Type:** text/html; charset=utf-8  
**Date:** Sat, 21 Oct 2017 06:13:16 GMT  
**FilterFactoryHeader:** Filter Factory Header Value 1  
**FilterFactoryHeader:** Filter Factory Header Value 2  
**GlobalHeader:** Global Header Value  
**OtherPagesHeader:** OtherPages Header Value OtherPages Header Value  
**Server:** Kestrel  
**Transfer-Encoding:** chunked

## Convenção de modelo de aplicativo de página

Use `AddPageApplicationModelConvention` para criar e adicionar um `IPageApplicationModelConvention` que invoca uma ação no `PageApplicationModel` para a página com o nome especificado.

A amostra explica o uso de `AddPageApplicationModelConvention` adicionando um cabeçalho, `AboutHeader`, à página `About`:

```
options.Conventions.AddPageApplicationModelConvention("/About", model =>
{
    model.Filters.Add(new AddHeaderAttribute(
        "AboutHeader", new string[] { "About Header Value" }));
});
```

Solicite a página `About` da amostra em `localhost:5000/About` e inspecione os cabeçalhos para exibir o resultado:

▼ Response Headers [view source](#)

**AboutHeader:** About Header Value About Header Value  
**Content-Type:** text/html; charset=utf-8  
**Date:** Thu, 19 Oct 2017 21:09:07 GMT  
**FilterFactoryHeader:** Filter Factory Header Value 1  
**FilterFactoryHeader:** Filter Factory Header Value 2  
**GlobalHeader:** Global Header Value  
**Server:** Kestrel  
**Transfer-Encoding:** chunked

## Configurar um filtro

`ConfigureFilter` configura o filtro especificado a ser aplicado. É possível implementar uma classe de filtro, mas o aplicativo de exemplo mostra como implementar um filtro em uma expressão lambda, que é implementado em segundo plano como um alocador que retorna um filtro:

```
options.Conventions.ConfigureFilter(model =>
{
    if (model.RelativePath.Contains("OtherPages/Page2"))
    {
        return new AddHeaderAttribute(
            "OtherPagesPage2Header",
            new string[] { "OtherPages/Page2 Header Value" });
    }
    return new EmptyFilter();
});
```

O modelo de aplicativo de página é usado para verificar o caminho relativo para segmentos que levam à página Page2 na pasta *OtherPages*. Se a condição é aprovada, um cabeçalho é adicionado. Caso contrário, o `EmptyFilter` é aplicado.

`EmptyFilter` é um [filtro de Ação](#). Como os filtros de Ação são ignorados pelas Páginas do Razor, o `EmptyFilter` não funciona conforme esperado se o caminho não contém `OtherPages/Page2`.

Solicite a página Page2 da amostra em `localhost:5000/OtherPages/Page2` e inspecione os cabeçalhos para exibir o resultado:

▼ Response Headers [view source](#)

**Content-Type:** text/html; charset=utf-8  
**Date:** Mon, 23 Oct 2017 06:06:52 GMT  
**FilterFactoryHeader:** Filter Factory Header Value 1  
**FilterFactoryHeader:** Filter Factory Header Value 2  
**GlobalHeader:** Global Header Value  
**OtherPagesHeader:** OtherPages Header Value  
**OtherPagesPage2Header:** OtherPages/Page2 Header Value    
**Server:** Kestrel  
**Transfer-Encoding:** chunked

## Configurar um alocador de filtro

[ConfigureFilter](#) configura o alocador especificado para aplicar [filtros](#) a todas as Páginas do Razor.

O aplicativo de exemplo fornece um exemplo de como usar um [alocador de filtro](#) adicionando um cabeçalho, `FilterFactoryHeader`, com dois valores para as páginas do aplicativo:

```
options.Conventions.ConfigureFilter(new AddHeaderWithFactory());
```

*AddHeaderWithFactory.cs:*

```

public class AddHeaderWithFactory : IFilterFactory
{
    // Implement IFilterFactory
    public IFilterMetadata CreateInstance(IServiceProvider serviceProvider)
    {
        return new AddHeaderFilter();
    }

    private class AddHeaderFilter : IResultFilter
    {
        public void OnResultExecuting(ResultExecutingContext context)
        {
            context.HttpContext.Response.Headers.Add(
                "FilterFactoryHeader",
                new string[]
                {
                    "Filter Factory Header Value 1",
                    "Filter Factory Header Value 2"
                });
        }

        public void OnResultExecuted(ResultExecutedContext context)
        {
        }
    }

    public bool IsReusable
    {
        get
        {
            return false;
        }
    }
}

```

Solicite a página About da amostra em `localhost:5000/About` e inspecione os cabeçalhos para exibir o resultado:

#### ▼ Response Headers [view source](#)

**AboutHeader:** About Header Value  
**Content-Type:** text/html; charset=utf-8  
**Date:** Thu, 19 Oct 2017 21:09:07 GMT  
**FilterFactoryHeader:** Filter Factory Header Value 1  
**FilterFactoryHeader:** Filter Factory Header Value 2  
**GlobalHeader:** Global Header Value  
**Server:** Kestrel  
**Transfer-Encoding:** chunked

## Substituir o provedor de modelo de aplicativo de página padrão

As Páginas do Razor usam a interface `IPageApplicationModelProvider` para criar um `DefaultPageApplicationModelProvider`. Herde do provedor de modelo padrão para fornecer sua própria lógica de implementação para descoberta e processamento do manipulador. A implementação padrão ([fonte de referência](#)) estabelece convenções para a nomenclatura *sem nome* e *nomeado* do manipulador, que é descrita abaixo.

### Métodos de manipulador sem nome padrão

Os métodos de manipulador para verbos HTTP (métodos de manipulador "sem nome") seguem uma convenção: `On<HTTP verb>[Async]` (o acréscimo de `Async` é opcional, mas recomendado para métodos assíncronos).

MÉTODO DE MANIPULADOR SEM NOME	OPERAÇÃO
OnGet / OnGetAsync	Inicializar o estado da página.
OnPost / OnPostAsync	Manipular solicitações POST.
OnDelete / OnDeleteAsync	Manipular solicitações DELETE†.
OnPut / OnPutAsync	Manipular solicitações PUT†.
OnPatch / OnPatchAsync	Manipular solicitações PATCH8224;.

†Usado para fazer chamadas à API para a página.

## Métodos de manipulador nomeado padrão

Os métodos de manipulador fornecidos pelo desenvolvedor (métodos de manipulador "nomeado") seguem uma convenção semelhante. O nome do manipulador é exibido após o verbo HTTP ou entre o verbo HTTP e `[Async]`: `On<HTTP verb><handler name>[Async]` (o acréscimo de `[Async]` é opcional, mas recomendado para métodos assíncronos). Por exemplo, os métodos que processam mensagens podem usar a nomenclatura mostrada na tabela abaixo.

MÉTODO DO MANIPULADOR NOMEADO DE EXEMPLO	OPERAÇÃO DE EXEMPLO
OnGetMessage / OnGetMessageAsync	Obter uma mensagem.
OnPostMessage / OnPostMessageAsync	Executar POST em uma mensagem.
OnDeleteMessage / OnDeleteMessageAsync	Executar DELETE em uma mensagem†.
OnPutMessage / OnPutMessageAsync	Executar PUT em uma mensagem†.
OnPatchMessage / OnPatchMessageAsync	Executar PATCH em uma mensagem†.

†Usado para fazer chamadas à API para a página.

## Personalizar os nomes do método de manipulador

Suponha que você prefira alterar a maneira como os métodos de manipulador nomeado e não nomeado são nomeados. Um esquema de nomenclatura alternativo é evitar que os nomes de método comecem com "On" e usar o primeiro segmento de palavra para determinar o verbo HTTP. Faça outras alterações, como converter os verbos DELETE, PUT e PATCH em POST. Um esquema desse tipo fornece os nomes de método mostrados na tabela a seguir.

MÉTODO DO MANIPULADOR	OPERAÇÃO
Get	Inicializar o estado da página.
Post / PostAsync	Manipular solicitações POST.
Delete / DeleteAsync	Manipular solicitações DELETE†.
Put / PutAsync	Manipular solicitações PUT†.

MÉTODO DO MANIPULADOR	OPERAÇÃO
Patch / PatchAsync	Manipular solicitações PATCH <sup>8224</sup> ;
GetMessage	Obter uma mensagem.
PostMessage / PostMessageAsync	Executar POST em uma mensagem.
DeleteMessage / DeleteMessageAsync	Executar POST em uma mensagem a ser excluída.
PutMessage / PutMessageAsync	Executar POST em uma mensagem a ser colocada.
PatchMessage / PatchMessageAsync	Executar POST em uma mensagem a ser corrigida.

<sup>8</sup>Usado para fazer chamadas à API para a página.

Para estabelecer esse esquema, herde da classe `DefaultPageApplicationModelProvider` e substitua o método `CreateHandlerModel` para fornecer uma lógica personalizada para resolver os nomes de manipulador de `PageModel`. O aplicativo de exemplo mostra como isso é feito em sua classe `CustomPageApplicationModelProvider`:

```
public class CustomPageApplicationModelProvider : 
    DefaultPageApplicationModelProvider
{
    public CustomPageApplicationModelProvider(IModelMetadataProvider modelMetadataProvider,
        IOptions<MvcOptions> options)
        : base (modelMetadataProvider, options)
    {
    }

    protected override PageHandlerModel CreateHandlerModel(MethodInfo method)
    {
        if (method == null)
        {
            throw new ArgumentNullException(nameof(method));
        }

        if (!IsHandler(method))
        {
            return null;
        }

        if (!TryParseHandlerMethod(
            method.Name, out var httpMethod, out var handlerName))
        {
            return null;
        }

        var handlerModel = new PageHandlerModel(
            method,
            method.GetCustomAttributes(inherit: true))
        {
            Name = method.Name,
            HandlerName = handlerName,
            HttpMethod = httpMethod,
        };

        var methodParameters = handlerModel.MethodInfo.GetParameters();

        for (var i = 0; i < methodParameters.Length; i++)
        {
            var parameter = methodParameters[i];
            var parameterModel = CreateParameterModel(parameter);
        }
    }
}
```

```

        parameterModel.Handler = handlerModel;

        handlerModel.Parameters.Add(parameterModel);
    }

    return handlerModel;
}

private static bool TryParseHandlerMethod(
    string methodName, out string httpMethod, out string handler)
{
    httpMethod = null;
    handler = null;

    // Parse the method name according to our conventions to
    // determine the required HTTP verb and optional
    // handler name.
    //
    // Valid names look like:
    // - Get
    // - Post
    // - PostAsync
    // - GetMessage
    // - PostMessage
    // - DeleteMessage
    // - DeleteMessageAsync

    var length = methodName.Length;
    if (methodName.EndsWith("Async", StringComparison.OrdinalIgnoreCase))
    {
        length -= "Async".Length;
    }

    if (length == 0)
    {
        // The method is named "Async". Exit processing.
        return false;
    }

    // The HTTP verb is at the start of the method name. Use
    // casing to determine where it ends.
    var handlerNameStart = 1;
    for (; handlerNameStart < length; handlerNameStart++)
    {
        if (char.ToUpper(methodName[handlerNameStart]))
        {
            break;
        }
    }

    httpMethod = methodName.Substring(0, handlerNameStart);

    // The handler name follows the HTTP verb and is optional.
    // It includes everything up to the end excluding the
    // "Async" suffix, if present.
    handler = handlerNameStart == length ? null : methodName.Substring(0, length);

    if (string.Equals(httpMethod, "GET", StringComparison.OrdinalIgnoreCase) ||
        string.Equals(httpMethod, "POST", StringComparison.OrdinalIgnoreCase))
    {
        // Do nothing. The httpMethod is correct for GET and POST.
        return true;
    }
    if (string.Equals(httpMethod, "DELETE", StringComparison.OrdinalIgnoreCase) ||
        string.Equals(httpMethod, "PUT", StringComparison.OrdinalIgnoreCase) ||
        string.Equals(httpMethod, "PATCH", StringComparison.OrdinalIgnoreCase))
    {
        // Convert HTTP verbs for DELETE, PUT, and PATCH to POST
        // For example: DeleteMessage, PutMessage, PatchMessage -> POST
    }
}

```

```

        httpMethod = "POST";
        return true;
    }
    else
    {
        return false;
    }
}

```

Destaques da classe incluem:

- A classe herda de `DefaultPageApplicationModelProvider`.
- O `TryParseHandlerMethod` processa um manipulador para determinar o verbo HTTP (`httpMethod`) e o nome do manipulador nomeado (`handlerName`) ao criar o `PageHandlerModel`.
  - Um `Async` pós-fixado é ignorado, se presente.
  - O uso de maiúsculas é adotado para analisar o verbo HTTP com base no nome do método.
  - Quando o nome do método (sem `Async`) é igual ao nome do verbo HTTP, não há um manipulador nomeado. O `handlerName` é definido como `null` e o nome do método é `Get`, `Post`, `Delete`, `Put` ou `Patch`.
  - Quando o nome do método (sem `Async`) é maior que o nome do verbo HTTP, há um manipulador nomeado. O `handlerName` é definido como `<method name (less 'Async', if present)>`. Por exemplo, "GetMessage" e "GetMessageAsync" geram um nome de manipulador "GetMessage".
  - Os verbos HTTP DELETE, PUT e PATCH são convertidos em POST.

Registre o `CustomPageApplicationModelProvider` na classe `Startup`:

```

services.AddSingleton<IPageApplicationModelProvider,
    CustomPageApplicationModelProvider>();

```

O modelo de página em `Index.cshtml.cs` mostra como as convenções de nomenclatura comuns de método de manipulador são alteradas para as páginas no aplicativo. A nomenclatura de prefixo comum "On" usada com as Páginas do Razor é removida. O método que inicializa o estado da página é agora nomeado `Get`. Você poderá ver essa convenção usada em todo o aplicativo se abrir qualquer modelo de página de uma das páginas.

Cada um dos outros métodos começam com o verbo HTTP que descreve seu processamento. Os dois métodos que começam com `Delete` normalmente são tratados como verbos HTTP DELETE, mas a lógica em `TryParseHandlerMethod` define de forma explícita o verbo como POST para ambos os manipuladores.

Observe que `Async` é opcional entre `DeleteAllMessages` e `DeleteMessageAsync`. Os dois são métodos assíncronos, mas você pode optar por usar o `Async` pós-fixado ou não; recomendamos que você faça isso. `DeleteAllMessages` é usado aqui para fins de demonstração, mas recomendamos que você nomeie um método desse tipo `DeleteAllMessagesAsync`. Isso não afeta o processamento da implementação da amostra, mas o uso do `Async` pós-fixado indica que se trata de um método assíncrono.

```

public async Task Get()
{
    Messages = await _db.Messages.AsNoTracking().ToListAsync();
}

public async Task<IActionResult> PostMessageAsync()
{
    _db.Messages.Add(Message);
    await _db.SaveChangesAsync();

    Result = $"{nameof(PostMessageAsync)} handler: Message '{Message.Text}' added.";

    return RedirectToPage();
}

public async Task<IActionResult> DeleteAllMessages()
{
    foreach (Message message in _db.Messages)
    {
        _db.Messages.Remove(message);
    }
    await _db.SaveChangesAsync();

    Result = $"{nameof(DeleteAllMessages)} handler: All messages deleted.";

    return RedirectToPage();
}

public async Task<IActionResult> DeleteMessageAsync(int id)
{
    var message = await _db.Messages.FindAsync(id);

    if (message != null)
    {
        _db.Messages.Remove(message);
        await _db.SaveChangesAsync();
    }

    Result = $"{nameof(DeleteMessageAsync)} handler: Message with Id: {id} deleted.";

    return RedirectToPage();
}

```

Observe que os nomes de manipulador fornecidos em *Index.cshtml* correspondem aos métodos de manipulador

`DeleteAllMessages` e `DeleteMessageAsync` :

```

<div class="row">
    <div class="col-md-3">
        <form method="post">
            <h2>Clear all messages</h2>
            <hr>
            <div class="form-group">
                <button type="submit" asp-page-handler="DeleteAllMessages"
                        class="btn btn-danger">Clear All</button>
            </div>
        </form>
    </div>
</div>

<div class="row">
    <div class="col-md-12">
        <form method="post">
            <h2>Messages</h2>
            <hr>
            <ol>
                @foreach (var message in Model.Messages)
                {
                    <li>
                        @message.Text
                        <button type="submit" asp-page-handler="DeleteMessage"
                                class="btn btn-danger"
                                asp-route-id="@message.Id">Delete</button>
                    </li>
                }
            </ol>
        </form>
    </div>

```

`Async` no nome do método de manipulador `DeleteMessageAsync` é excluído do `TryParseHandlerMethod` para a correspondência de manipulador da solicitação POST com o método. É feita a correspondência do nome `asp-page-handler` de `DeleteMessage` ao método de manipulador `DeleteMessageAsync`.

## Filtros MVC e o filtro de Página (IPageFilter)

Os [filtros de Ação](#) MVC são ignorados pelas Páginas do Razor, pois elas usam métodos de manipulador. Outros tipos de filtros MVC estão disponíveis para uso: [Autorização](#), [Exceção](#), [Recurso](#) e [Resultado](#). Para obter mais informações, consulte o tópico [Filtros](#).

O filtro de Página ([IPageFilter](#)) é um filtro que se aplica às Páginas do Razor. Para obter mais informações, confira [Métodos de filtro para Páginas Razor](#).

## Recursos adicionais

- [Convenções de autorização de Páginas Razor](#)

# Carregar arquivos para uma Página Razor no ASP.NET Core

09/12/2018 • 26 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

Este tópico tem como base o [aplicativo de exemplo](#) em [Tutorial: Introdução às Páginas do Razor no ASP.NET Core](#).

Este tópico mostra como usar a associação de modelos simples para carregar arquivos, o que funciona bem para carregar arquivos pequenos. Para obter informações sobre a transmissão de arquivos grandes, consulte [Carregando arquivos grandes com streaming](#).

Nas etapas a seguir, um recurso de upload de arquivo da agenda de filmes será adicionado ao aplicativo de exemplo. Um agendamento de filmes é representado por uma classe `Schedule`. A classe inclui duas versões do agendamento. Uma versão é fornecida aos clientes, `PublicSchedule`. A outra versão é usada para os funcionários da empresa, `PrivateSchedule`. Cada versão é carregada como um arquivo separado. O tutorial demonstra como executar dois carregamentos de arquivos de uma página com um único POST para o servidor.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Considerações sobre segurança

É preciso ter cuidado ao fornecer aos usuários a possibilidade de carregar arquivos em um servidor. Os invasores podem executar uma [negação de serviço](#) e outros ataques em um sistema. Algumas etapas de segurança que reduzem a probabilidade de um ataque bem-sucedido são:

- Fazer o upload de arquivos para uma área de upload de arquivos dedicada no sistema, o que facilita a aplicação de medidas de segurança sobre o conteúdo carregado. Ao permitir os uploads de arquivos, certifique-se de que as permissões de execução estejam desabilitadas no local do upload.
- Use um nome de arquivo seguro determinado pelo aplicativo, não da entrada do usuário ou o nome do arquivo carregado.
- Permitir somente um conjunto específico de extensões de arquivo aprovadas.
- Certifique-se de que as verificações do lado do cliente sejam realizadas no servidor. As verificações do lado do cliente são fáceis de contornar.
- Verifique o tamanho do upload e evite uploads maiores do que o esperado.
- Execute um verificador de vírus/malware no conteúdo carregado.

### WARNING

Carregar códigos mal-intencionados em um sistema é frequentemente a primeira etapa para executar o código que pode:

- Tomar o controle total de um sistema.
- Sobrecarregar um sistema, fazendo com que ele falhe completamente.
- Comprometer dados do sistema ou de usuários.
- Aplicar pichações a uma interface pública.

## Adicionar uma classe FileUpload

Criar uma Página Razor para lidar com um par de carregamentos de arquivos. Adicione uma classe `FileUpload`,

que é vinculada à página para obter os dados do agendamento. Clique com o botão direito do mouse na pasta **Modelos**. Selecione **Adicionar > Classe**. Nomeie a classe **FileUpload** e adicione as seguintes propriedades:

```
using Microsoft.AspNetCore.Http;
using System.ComponentModel.DataAnnotations;

namespace RazorPagesMovie.Models
{
    public class FileUpload
    {
        [Required]
        [Display(Name="Title")]
        [StringLength(60, MinimumLength = 3)]
        public string Title { get; set; }

        [Required]
        [Display(Name="Public Schedule")]
        public IFormFile UploadPublicSchedule { get; set; }

        [Required]
        [Display(Name="Private Schedule")]
        public IFormFile UploadPrivateSchedule { get; set; }
    }
}
```

```
using Microsoft.AspNetCore.Http;
using System.ComponentModel.DataAnnotations;

namespace RazorPagesMovie.Models
{
    public class FileUpload
    {
        [Required]
        [Display(Name="Title")]
        [StringLength(60, MinimumLength = 3)]
        public string Title { get; set; }

        [Required]
        [Display(Name="Public Schedule")]
        public IFormFile UploadPublicSchedule { get; set; }

        [Required]
        [Display(Name="Private Schedule")]
        public IFormFile UploadPrivateSchedule { get; set; }
    }
}
```

A classe tem uma propriedade para o título do agendamento e uma propriedade para cada uma das duas versões do agendamento. Todas as três propriedades são necessárias e o título deve ter 3-60 caracteres.

## Adicionar um método auxiliar para carregar arquivos

Para evitar duplicação de código para processar arquivos do agendamento carregados, primeiro, adicione um método auxiliar estático. Crie uma pasta de *Utilitários* no aplicativo e adicione um arquivo *FileHelpers.cs* com o seguinte conteúdo. O método auxiliar, `ProcessFormFile`, usa um `IFormFile` e `ModelStateDictionary` e retorna uma cadeia de caracteres que contém o tamanho e o conteúdo do arquivo. O comprimento e o tipo de conteúdo são verificados. Se o arquivo não passar em uma verificação de validação, um erro será adicionado ao `ModelState`.

```
using System;
using System.ComponentModel.DataAnnotations;
using System.IO;
```

```
using System.Net;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc.ModelBinding;
using RazorPagesMovie.Models;

namespace RazorPagesMovie.Utilities
{
    public class FileHelpers
    {
        public static async Task<string> ProcessFormFile(IFormFile formFile,
            ModelStateDictionary ModelState)
        {
            var fieldDisplayName = string.Empty;

            // Use reflection to obtain the display name for the model
            // property associated with this IFormFile. If a display
            // name isn't found, error messages simply won't show
            // a display name.
            MemberInfo property =
                typeof(FileUpload).GetProperty(
                    formFile.Name.Substring(formFile.Name.IndexOf(".") + 1));

            if (property != null)
            {
                var displayAttribute =
                    property.GetCustomAttribute(typeof(DisplayAttribute))
                        as DisplayAttribute;

                if (displayAttribute != null)
                {
                    fieldDisplayName = $"{displayAttribute.Name} ";
                }
            }

            // Use Path.GetFileName to obtain the file name, which will
            // strip any path information passed as part of the
            // FileName property. HtmlEncode the result in case it must
            // be returned in an error message.
            var fileName = WebUtility.HtmlEncode(
                Path.GetFileName(formFile.FileName));

            if (formFile.ContentType.ToLower() != "text/plain")
            {
                ModelState.AddModelError(formFile.Name,
                    $"The {fieldDisplayName}file ({fileName}) must be a text file.");
            }

            // Check the file length and don't bother attempting to
            // read it if the file contains no content. This check
            // doesn't catch files that only have a BOM as their
            // content, so a content length check is made later after
            // reading the file's content to catch a file that only
            // contains a BOM.
            if (formFile.Length == 0)
            {
                ModelState.AddModelError(formFile.Name,
                    $"The {fieldDisplayName}file ({fileName}) is empty.");
            }
            else if (formFile.Length > 1048576)
            {
                ModelState.AddModelError(formFile.Name,
                    $"The {fieldDisplayName}file ({fileName}) exceeds 1 MB.");
            }
            else
            {
                try
```

```

        {
            string fileContents;

            // The StreamReader is created to read files that are UTF-8 encoded.
            // If uploads require some other encoding, provide the encoding in the
            // using statement. To change to 32-bit encoding, change
            // new UTF8Encoding(...) to new UTF32Encoding().
            using (
                var reader =
                    new StreamReader(
                        formFile.OpenReadStream(),
                        new UTF8Encoding(encoderShouldEmitUTF8Identifier: false,
                            throwOnInvalidBytes: true),
                        detectEncodingFromByteOrderMarks: true))
            {
                fileContents = await reader.ReadToEndAsync();

                // Check the content length in case the file's only
                // content was a BOM and the content is actually
                // empty after removing the BOM.
                if (fileContents.Length > 0)
                {
                    return fileContents;
                }
                else
                {
                    ModelState.AddModelError(formFile.Name,
                        $"The {fieldDisplayName}file ({fileName}) is empty.");
                }
            }
        }
        catch (Exception ex)
        {
            ModelState.AddModelError(formFile.Name,
                $"The {fieldDisplayName}file ({fileName}) upload failed. " +
                $"Please contact the Help Desk for support. Error: {ex.Message}");
            // Log the exception
        }
    }

    return string.Empty;
}
}
}

```

```

using System;
using System.ComponentModel.DataAnnotations;
using System.IO;
using System.Net;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc.ModelBinding;
using RazorPagesMovie.Models;

namespace RazorPagesMovie.Utilities
{
    public class FileHelpers
    {
        public static async Task<string> ProcessFormFile(
            IFormFile formFile, ModelStateDictionary ModelState)
        {
            var fieldDisplayName = string.Empty;

            // Use reflection to obtain the display name for the model
            // property associated with this IFormFile. If a display

```

```

// name isn't found, error messages simply won't show
// a display name.
MemberInfo property =
    typeof(FileUpload).GetProperty(formFile.Name.Substring(
        formFile.Name.IndexOf(".") + 1));

if (property != null)
{
    var displayAttribute =
        property.GetCustomAttribute(typeof(DisplayAttribute))
            as DisplayAttribute;

    if (displayAttribute != null)
    {
        fieldDisplayName = $"{displayAttribute.Name} ";
    }
}

// Use Path.GetFileName to obtain the file name, which will
// strip any path information passed as part of the
// FileName property. HtmlEncode the result in case it must
// be returned in an error message.
var fileName = WebUtility.HtmlEncode(
    Path.GetFileName(formFile.FileName));

if (formFile.ContentType.ToLower() != "text/plain")
{
    ModelState.AddModelError(formFile.Name,
        $"The {fieldDisplayName}file ({fileName}) must be a text file.");
}

// Check the file length and don't bother attempting to
// read it if the file contains no content. This check
// doesn't catch files that only have a BOM as their
// content, so a content length check is made later after
// reading the file's content to catch a file that only
// contains a BOM.
if (formFile.Length == 0)
{
    ModelState.AddModelError(formFile.Name,
        $"The {fieldDisplayName}file ({fileName}) is empty.");
}
else if (formFile.Length > 1048576)
{
    ModelState.AddModelError(formFile.Name,
        $"The {fieldDisplayName}file ({fileName}) exceeds 1 MB.");
}
else
{
    try
    {
        string fileContents;

        // The StreamReader is created to read files that are UTF-8 encoded.
        // If uploads require some other encoding, provide the encoding in the
        // using statement. To change to 32-bit encoding, change
        // new UTF8Encoding(...) to new UTF32Encoding().
        using (
            var reader =
                new StreamReader(
                    formFile.OpenReadStream(),
                    new UTF8Encoding(encoderShouldEmitUTF8Identifier: false,
                        throwOnInvalidBytes: true),
                    detectEncodingFromByteOrderMarks: true))
        {
            fileContents = await reader.ReadToEndAsync();

            // Check the content length in case the file's only
            // content was a BOM and the content is actually

```

```

        // empty after removing the BOM.
        if (fileContents.Length > 0)
        {
            return fileContents;
        }
        else
        {
            ModelState.AddModelError(formFile.Name,
                $"The {fieldDisplayName}file ({fileName}) is empty.");
        }
    }
    catch (Exception ex)
    {
        ModelState.AddModelError(formFile.Name,
            $"The {fieldDisplayName}file ({fileName}) upload failed. " +
            $"Please contact the Help Desk for support. Error: {ex.Message}");
        // Log the exception
    }
}

return string.Empty;
}
}
}

```

## Salvar o arquivo no disco

O aplicativo de exemplo salva os arquivos carregados em campos de banco de dados. Para salvar um arquivo no disco, use um `FileStream`. O exemplo a seguir copia um arquivo mantido por `FileUpload.UploadPublicSchedule` para um `FileStream` em um método `OnPostAsync`. O `FileStream` grava o arquivo no disco no `<PATH-AND-FILE-NAME>` fornecido:

```

public async Task<IActionResult> OnPostAsync()
{
    // Perform an initial check to catch FileUpload class attribute violations.
    if (!ModelState.IsValid)
    {
        return Page();
    }

    var filePath = "<PATH-AND-FILE-NAME>";

    using (var fileStream = new FileStream(filePath, FileMode.Create))
    {
        await FileUpload.UploadPublicSchedule.CopyToAsync(fileStream);
    }

    return RedirectToPage("./Index");
}

```

O processo de trabalho deve ter permissões de gravação para o local especificado por `filePath`.

### NOTE

O `filePath` precisa incluir o nome do arquivo. Se o nome do arquivo não for fornecido, uma `UnauthorizedAccessException` será gerada no tempo de execução.

## WARNING

Nunca persista os arquivos carregados na mesma árvore de diretório que o aplicativo.

O exemplo de código não oferece proteção do lado do servidor contra carregamentos de arquivos mal-intencionados. Para obter informações de como reduzir a área da superfície de ataque ao aceitar arquivos de usuários, confira os seguintes recursos:

- [Unrestricted File Upload \(Carregamento de arquivo irrestrito\)](#)
- [Segurança do Azure: Verifique se os controles adequados estão em vigor ao aceitar arquivos de usuários](#)

## Salvar o arquivo no Armazenamento de Blobs do Azure

Para carregar o conteúdo do arquivo para o Armazenamento de Blobs do Azure, confira [Introdução ao Armazenamento de Blobs do Azure usando o .NET](#). O tópico demonstra como usar `UploadFromStream` para salvar um `FileStream` para armazenamento de blobs.

## Adicionar a classe de Agendamento

Clique com o botão direito do mouse na pasta *Modelos*. Selecione **Adicionar > Classe**. Nomeie a classe **Agendamento** e adicione as seguintes propriedades:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace RazorPagesMovie.Models
{
    public class Schedule
    {
        public int ID { get; set; }
        public string Title { get; set; }

        public string PublicSchedule { get; set; }

        [Display(Name = "Public Schedule Size (bytes)")]
        [DisplayFormat(DataFormatString = "{0:N1}")]
        public long PublicScheduleSize { get; set; }

        public string PrivateSchedule { get; set; }

        [Display(Name = "Private Schedule Size (bytes)")]
        [DisplayFormat(DataFormatString = "{0:N1}")]
        public long PrivateScheduleSize { get; set; }

        [Display(Name = "Uploaded (UTC)")]
        [DisplayFormat(DataFormatString = "{0:F}")]
        public DateTime UploadDT { get; set; }
    }
}
```

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace RazorPagesMovie.Models
{
    public class Schedule
    {
        public int ID { get; set; }
        public string Title { get; set; }

        public string PublicSchedule { get; set; }

        [Display(Name = "Public Schedule Size (bytes)")]
        [DisplayFormat(DataFormatString = "{0:N1}")]
        public long PublicScheduleSize { get; set; }

        public string PrivateSchedule { get; set; }

        [Display(Name = "Private Schedule Size (bytes)")]
        [DisplayFormat(DataFormatString = "{0:N1}")]
        public long PrivateScheduleSize { get; set; }

        [Display(Name = "Uploaded (UTC)")]
        [DisplayFormat(DataFormatString = "{0:F}")]
        public DateTime UploadDT { get; set; }
    }
}

```

Usa a classe usa os atributos `Display` e `DisplayFormat`, que produzem formatação e títulos fáceis quando os dados de agendamento são renderizados.

## Atualizar o RazorPagesMovieContext

Especifique um `DbSet` no `RazorPagesMovieContext` (`Data/RazorPagesMovieContext.cs`) para os agendamentos:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;

namespace RazorPagesMovie.Models
{
    public class RazorPagesMovieContext : DbContext
    {
        public RazorPagesMovieContext (DbContextOptions<RazorPagesMovieContext> options)
            : base(options)
        {

        }

        public DbSet<RazorPagesMovie.Models.Movie> Movie { get; set; }
        public DbSet<RazorPagesMovie.Models.Schedule> Schedule { get; set; }
    }
}

```

## Atualizar o MovieContext

Especifique um `DbSet` no `MovieContext` (`Models/MovieContext.cs`) para os agendamentos:

```

using Microsoft.EntityFrameworkCore;

namespace RazorPagesMovie.Models
{
    public class MovieContext : DbContext
    {
        public MovieContext(DbContextOptions<MovieContext> options)
            : base(options)
        {

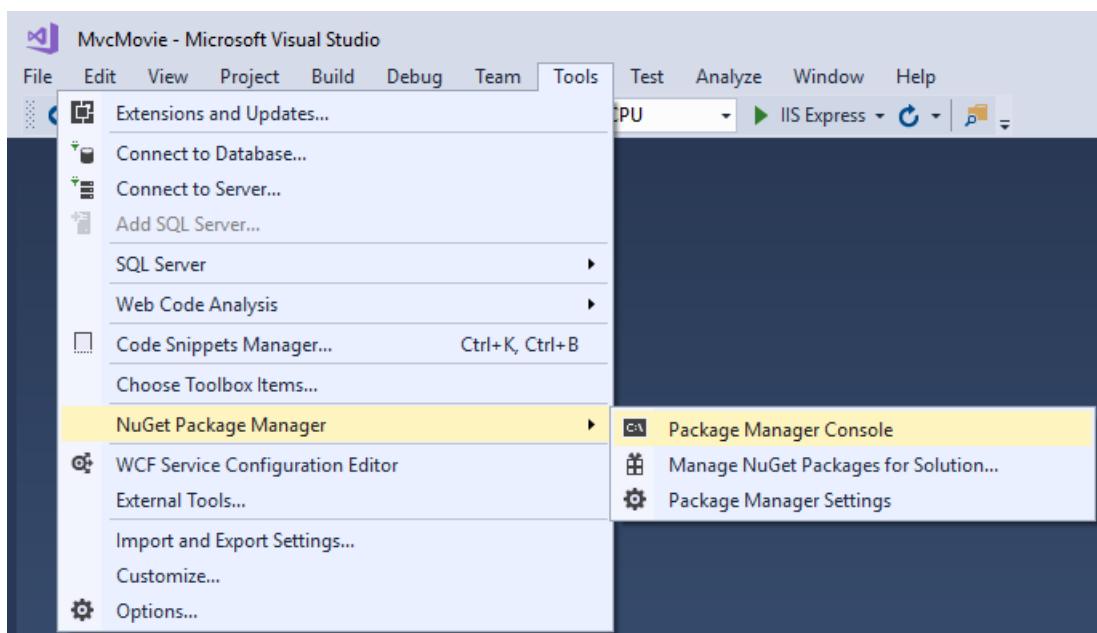
        }

        public DbSet<Movie> Movie { get; set; }
        public DbSet<Schedule> Schedule { get; set; }
    }
}

```

## Adicione a tabela de Agendamento ao banco de dados

Abra o Console Gerenciador de pacote (PMC): **Ferramentas > Gerenciador de pacote NuGet > Console Gerenciador de pacote.**



No PMC, execute os seguintes comandos. Estes comandos adicionam uma tabela `Schedule` ao banco de dados:

```

Add-Migration AddScheduleTable
Update-Database

```

## Adicionar uma página do Razor de upload de arquivo

Na pasta *Páginas*, crie uma pasta *Agendamentos*. Na pasta *Agendamentos*, crie uma página chamada *Index.cshtml* para carregar um agendamento com o seguinte conteúdo:

```

@page
@model RazorPagesMovie.Pages.Schedules.IndexModel

 @{
     ViewData["Title"] = "Schedules";
 }

<h2>Schedules</h2>
<hr />

```

```

<...>

<h3>Upload Schedules</h3>
<div class="row">
    <div class="col-md-4">
        <form method="post" enctype="multipart/form-data">
            <div class="form-group">
                <label asp-for="FileUpload.Title" class="control-label"></label>
                <input asp-for="FileUpload.Title" type="text" class="form-control" />
                <span asp-validation-for="FileUpload.Title" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="FileUpload.UploadPublicSchedule" class="control-label"></label>
                <input asp-for="FileUpload.UploadPublicSchedule" type="file" class="form-control" style="height:auto" />
                <span asp-validation-for="FileUpload.UploadPublicSchedule" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="FileUpload.UploadPrivateSchedule" class="control-label"></label>
                <input asp-for="FileUpload.UploadPrivateSchedule" type="file" class="form-control" style="height:auto" />
                <span asp-validation-for="FileUpload.UploadPrivateSchedule" class="text-danger"></span>
            </div>
            <input type="submit" value="Upload" class="btn btn-default" />
        </form>
    </div>
</div>

<h3>Loaded Schedules</h3>
<table class="table">
    <thead>
        <tr>
            <th></th>
            <th>
                @Html.DisplayNameFor(model => model.Schedule[0].Title)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Schedule[0].UploadDT)
            </th>
            <th class="text-center">
                @Html.DisplayNameFor(model => model.Schedule[0].PublicScheduleSize)
            </th>
            <th class="text-center">
                @Html.DisplayNameFor(model => model.Schedule[0].PrivateScheduleSize)
            </th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model.Schedule) {
            <tr>
                <td>
                    <a asp-page="~/Delete" asp-route-id="@item.ID">Delete</a>
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Title)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.UploadDT)
                </td>
                <td class="text-center">
                    @Html.DisplayFor(modelItem => item.PublicScheduleSize)
                </td>
                <td class="text-center">
                    @Html.DisplayFor(modelItem => item.PrivateScheduleSize)
                </td>
            </tr>
        }
    </tbody>
</table>

```

```

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}

```

```

@page
@model RazorPagesMovie.Pages.Schedules.IndexModel

@{
    ViewData["Title"] = "Schedules";
}

<h2>Schedules</h2>
<hr />

<h3>Upload Schedules</h3>
<div class="row">
    <div class="col-md-4">
        <form method="post" enctype="multipart/form-data">
            <div class="form-group">
                <label asp-for="FileUpload.Title" class="control-label"></label>
                <input asp-for="FileUpload.Title" type="text" class="form-control" />
                <span asp-validation-for="FileUpload.Title" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="FileUpload.UploadPublicSchedule" class="control-label"></label>
                <input asp-for="FileUpload.UploadPublicSchedule" type="file" class="form-control" style="height:auto" />
                <span asp-validation-for="FileUpload.UploadPublicSchedule" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="FileUpload.UploadPrivateSchedule" class="control-label"></label>
                <input asp-for="FileUpload.UploadPrivateSchedule" type="file" class="form-control" style="height:auto" />
                <span asp-validation-for="FileUpload.UploadPrivateSchedule" class="text-danger"></span>
            </div>
            <input type="submit" value="Upload" class="btn btn-default" />
        </form>
    </div>
</div>

<h3>Loaded Schedules</h3>
<table class="table">
    <thead>
        <tr>
            <th></th>
            <th>
                @Html.DisplayNameFor(model => model.Schedule[0].Title)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Schedule[0].UploadDT)
            </th>
            <th class="text-center">
                @Html.DisplayNameFor(model => model.Schedule[0].PublicScheduleSize)
            </th>
            <th class="text-center">
                @Html.DisplayNameFor(model => model.Schedule[0].PrivateScheduleSize)
            </th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model.Schedule) {
    <tr>
        <td>
            <a asp-page=".Delete" asp-route-id="@item.ID">Delete</a>
        </td>
        <td>
            @Html.DisplayNameFor(modelItem => item.Title)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.UploadDT)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.PublicScheduleSize)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.PrivateScheduleSize)
        </td>
    </tr>
}
    </tbody>
</table>

```

```

        @Html.DisplayFor(modelItem => item.Title)
    </td>
    <td>
        @Html.DisplayFor(modelItem => item.UploadDT)
    </td>
    <td class="text-center">
        @Html.DisplayFor(modelItem => item.PublicScheduleSize)
    </td>
    <td class="text-center">
        @Html.DisplayFor(modelItem => item.PrivateScheduleSize)
    </td>
</tr>
}
</tbody>
</table>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}

```

Cada grupo de formulário inclui um **<rótulo>** que exibe o nome de cada propriedade de classe. Os atributos `Display` no modelo `FileUpload` fornecem os valores de exibição para os rótulos. Por exemplo, o nome de exibição da propriedade `uploadPublicSchedule` é definido com `[Display(Name="Public Schedule")]` e, portanto, exibe "Agendamento público" no rótulo quando o formulário é renderizado.

Cada grupo de formulário inclui uma validação **<span>**. Se a entrada do usuário não atender aos atributos de propriedade definidos na classe `FileUpload` ou se qualquer uma das verificações de validação do arquivo de método `ProcessFormFile` falhar, o modelo não será validado. Quando a validação do modelo falha, uma mensagem de validação útil é renderizada para o usuário. Por exemplo, a propriedade `Title` é anotada com `[Required]` e `[StringLength(60, MinimumLength = 3)]`. Se o usuário não fornecer um título, ele receberá uma mensagem indicando que um valor é necessário. Se o usuário inserir um valor com menos de três caracteres ou mais de sessenta, ele receberá uma mensagem indicando que o valor tem um comprimento incorreto. Se um arquivo que não tem nenhum conteúdo for fornecido, uma mensagem aparecerá indicando que o arquivo está vazio.

## Adicionar o modelo de página

Adicione o modelo de página (`Index.cshtml.cs`) à pasta `Schedules`:

```

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using RazorPagesMovie.Models;
using RazorPagesMovie.Utilities;

namespace RazorPagesMovie.Pages.Schedules
{
    public class IndexModel : PageModel
    {
        private readonly RazorPagesMovie.Models.RazorPagesMovieContext _context;

        public IndexModel(RazorPagesMovie.Models.RazorPagesMovieContext context)
        {
            _context = context;
        }

        [BindProperty]
        public FileUpload FileUpload { get; set; }

        public IList<Schedule> Schedule { get; private set; }
    }
}

```

```

public async Task OnGetAsync()
{
    Schedule = await _context.Schedule.AsNoTracking().ToListAsync();
}

public async Task<IActionResult> OnPostAsync()
{
    // Perform an initial check to catch FileUpload class
    // attribute violations.
    if (!ModelState.IsValid)
    {
        Schedule = await _context.Schedule.AsNoTracking().ToListAsync();
        return Page();
    }

    var publicScheduleData =
        await FileHelpers.ProcessFormFile(FileUpload.UploadPublicSchedule, ModelState);

    var privateScheduleData =
        await FileHelpers.ProcessFormFile(FileUpload.UploadPrivateSchedule, ModelState);

    // Perform a second check to catch ProcessFormFile method
    // violations.
    if (!ModelState.IsValid)
    {
        Schedule = await _context.Schedule.AsNoTracking().ToListAsync();
        return Page();
    }

    var schedule = new Schedule()
    {
        PublicSchedule = publicScheduleData,
        PublicScheduleSize = FileUpload.UploadPublicSchedule.Length,
        PrivateSchedule = privateScheduleData,
        PrivateScheduleSize = FileUpload.UploadPrivateSchedule.Length,
        Title = FileUpload.Title,
        UploadDT = DateTime.UtcNow
    };

    _context.Schedule.Add(schedule);
    await _context.SaveChangesAsync();

    return RedirectToPage("./Index");
}
}
}

```

```

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using RazorPagesMovie.Models;
using RazorPagesMovie.Utilities;

namespace RazorPagesMovie.Pages.Schedules
{
    public class IndexModel : PageModel
    {
        private readonly RazorPagesMovie.Models.MovieContext _context;

        public IndexModel(RazorPagesMovie.Models.MovieContext context)
        {
            _context = context;
        }
    }
}

```

```

[BindProperty]
public FileUpload FileUpload { get; set; }

public IList<Schedule> Schedule { get; private set; }

public async Task OnGetAsync()
{
    Schedule = await _context.Schedule.AsNoTracking().ToListAsync();
}

public async Task<IActionResult> OnPostAsync()
{
    // Perform an initial check to catch FileUpload class
    // attribute violations.
    if (!ModelState.IsValid)
    {
        Schedule = await _context.Schedule.AsNoTracking().ToListAsync();
        return Page();
    }

    var publicScheduleData =
        await FileHelpers.ProcessFormFile(FileUpload.UploadPublicSchedule, ModelState);

    var privateScheduleData =
        await FileHelpers.ProcessFormFile(FileUpload.UploadPrivateSchedule, ModelState);

    // Perform a second check to catch ProcessFormFile method
    // violations.
    if (!ModelState.IsValid)
    {
        Schedule = await _context.Schedule.AsNoTracking().ToListAsync();
        return Page();
    }

    var schedule = new Schedule()
    {
        PublicSchedule = publicScheduleData,
        PublicScheduleSize = FileUpload.UploadPublicSchedule.Length,
        PrivateSchedule = privateScheduleData,
        PrivateScheduleSize = FileUpload.UploadPrivateSchedule.Length,
        Title = FileUpload.Title,
        UploadDT = DateTime.UtcNow
    };

    _context.Schedule.Add(schedule);
    await _context.SaveChangesAsync();

    return RedirectToPage("./Index");
}
}
}

```

O modelo de página (`IndexModel` no `Index.cshtml.cs`) associa a classe `FileUpload`:

```

[BindProperty]
public FileUpload FileUpload { get; set; }

```

```

[BindProperty]
public FileUpload FileUpload { get; set; }

```

O modelo também usa uma lista dos agendamentos (`IList<Schedule>`) para exibir os agendamentos armazenados no banco de dados na página:

```
public IList<Schedule> Schedule { get; private set; }
```

```
public IList<Schedule> Schedule { get; private set; }
```

Quando a página for carregada com `OnGetAsync`, `Schedules` é preenchido com o banco de dados e usado para gerar uma tabela HTML de agendamentos carregados:

```
public async Task OnGetAsync()
{
    Schedule = await _context.Schedule.AsNoTracking().ToListAsync();
}
```

```
public async Task OnGetAsync()
{
    Schedule = await _context.Schedule.AsNoTracking().ToListAsync();
}
```

Quando o formulário é enviado para o servidor, o `ModelState` é verificado. Se for inválido, `Schedule` é recriado e a página é renderizada com uma ou mais mensagens de validação informando por que a validação de página falhou. Se for válido, as propriedades `FileUpload` serão usadas em `OnPostAsync` para concluir o upload do arquivo para as duas versões do agendamento e criar um novo objeto `Schedule` para armazenar os dados. O agendamento, em seguida, é salvo no banco de dados:

```
public async Task<IActionResult> OnPostAsync()
{
    // Perform an initial check to catch FileUpload class
    // attribute violations.
    if (!ModelState.IsValid)
    {
        Schedule = await _context.Schedule.AsNoTracking().ToListAsync();
        return Page();
    }

    var publicScheduleData =
        await FileHelpers.ProcessSchedule(FileUpload.UploadPublicSchedule, ModelState);

    var privateScheduleData =
        await FileHelpers.ProcessSchedule(FileUpload.UploadPrivateSchedule, ModelState);

    // Perform a second check to catch ProcessSchedule method
    // violations.
    if (!ModelState.IsValid)
    {
        Schedule = await _context.Schedule.AsNoTracking().ToListAsync();
        return Page();
    }

    var schedule = new Schedule()
    {
        PublicSchedule = publicScheduleData,
        PublicScheduleSize = FileUpload.UploadPublicSchedule.Length,
        PrivateSchedule = privateScheduleData,
        PrivateScheduleSize = FileUpload.UploadPrivateSchedule.Length,
        Title = FileUpload.Title,
        UploadDT = DateTime.UtcNow
    };

    _context.Schedule.Add(schedule);
    await _context.SaveChangesAsync();

    return RedirectToPage("./Index");
}
```

```

public async Task<IActionResult> OnPostAsync()
{
    // Perform an initial check to catch FileUpload class
    // attribute violations.
    if (!ModelState.IsValid)
    {
        Schedule = await _context.Schedule.AsNoTracking().ToListAsync();
        return Page();
    }

    var publicScheduleData =
        await FileHelpers.ProcessSchedule(FileUpload.UploadPublicSchedule, ModelState);

    var privateScheduleData =
        await FileHelpers.ProcessSchedule(FileUpload.UploadPrivateSchedule, ModelState);

    // Perform a second check to catch ProcessSchedule method
    // violations.
    if (!ModelState.IsValid)
    {
        Schedule = await _context.Schedule.AsNoTracking().ToListAsync();
        return Page();
    }

    var schedule = new Schedule()
    {
        PublicSchedule = publicScheduleData,
        PublicScheduleSize = FileUpload.UploadPublicSchedule.Length,
        PrivateSchedule = privateScheduleData,
        PrivateScheduleSize = FileUpload.UploadPrivateSchedule.Length,
        Title = FileUpload.Title,
        UploadDT = DateTime.UtcNow
    };

    _context.Schedule.Add(schedule);
    await _context.SaveChangesAsync();

    return RedirectToPage("./Index");
}

```

## Vincular a página do Razor de upload de arquivo

Abra *Pages/Shared/\_Layout.cshtml* e adicione um link para a barra de navegação para acessar a página de Agendas:

```

<div class="navbar-collapse collapse">
    <ul class="nav navbar-nav">
        <li><a asp-page="/Index">Home</a></li>
        <li><a asp-page="/Schedules/Index">Schedules</a></li>
        <li><a asp-page="/About">About</a></li>
        <li><a asp-page="/Contact">Contact</a></li>
    </ul>
</div>

```

## Adicionar uma página para confirmar a exclusão de agendamento

Quando o usuário clica para excluir um agendamento, é oferecida uma oportunidade de cancelar a operação. Adicione uma página de confirmação de exclusão (*Delete.cshtml*) à pasta *Agendamentos*:

```

@page "{id:int}"
@model RazorPagesMovie.Pages.Schedules.DeleteModel

@{
    ViewData["Title"] = "Delete Schedule";
}

<h2>Delete Schedule</h2>

<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>Schedule</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Schedule.Title)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Schedule.Title)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Schedule.PublicScheduleSize)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Schedule.PublicScheduleSize)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Schedule.PrivateScheduleSize)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Schedule.PrivateScheduleSize)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Schedule.UploadDT)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Schedule.UploadDT)
        </dd>
    </dl>

    <form method="post">
        <input type="hidden" asp-for="Schedule.ID" />
        <input type="submit" value="Delete" class="btn btn-default" /> |
        <a asp-page="./Index">Back to List</a>
    </form>
</div>

```

```

@page "{id:int}"
@model RazorPagesMovie.Pages.Schedules.DeleteModel

@{
    ViewData["Title"] = "Delete Schedule";
}

<h2>Delete Schedule</h2>

<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>Schedule</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Schedule.Title)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Schedule.Title)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Schedule.PublicScheduleSize)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Schedule.PublicScheduleSize)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Schedule.PrivateScheduleSize)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Schedule.PrivateScheduleSize)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Schedule.UploadDT)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Schedule.UploadDT)
        </dd>
    </dl>

    <form method="post">
        <input type="hidden" asp-for="Schedule.ID" />
        <input type="submit" value="Delete" class="btn btn-default" /> |
        <a asp-page="./Index">Back to List</a>
    </form>
</div>

```

O modelo de página (*Delete.cshtml.cs*) carrega um único agendamento identificado por `id` nos dados de rota da solicitação. Adicione o arquivo *Delete.cshtml.cs* à pasta *Agendamentos*:

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using RazorPagesMovie.Models;

namespace RazorPagesMovie.Pages.Schedules
{
    public class DeleteModel : PageModel
    {
        private readonly RazorPagesMovie.Models.RazorPagesMovieContext _context;

        public DeleteModel(RazorPagesMovie.Models.RazorPagesMovieContext context)
        {
            _context = context;
        }

        [BindProperty]
        public Schedule Schedule { get; set; }

        public async Task<IActionResult> OnGetAsync(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            Schedule = await _context.Schedule.SingleOrDefault(m => m.ID == id);

            if (Schedule == null)
            {
                return NotFound();
            }
            return Page();
        }

        public async Task<IActionResult> OnPostAsync(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            Schedule = await _context.Schedule.FindAsync(id);

            if (Schedule != null)
            {
                _context.Schedule.Remove(Schedule);
                await _context.SaveChangesAsync();
            }

            return RedirectToPage("./Index");
        }
    }
}
```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using RazorPagesMovie.Models;

namespace RazorPagesMovie.Pages.Schedules
{
    public class DeleteModel : PageModel
    {
        private readonly RazorPagesMovie.Models.MovieContext _context;

        public DeleteModel(RazorPagesMovie.Models.MovieContext context)
        {
            _context = context;
        }

        [BindProperty]
        public Schedule Schedule { get; set; }

        public async Task<IActionResult> OnGetAsync(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            Schedule = await _context.Schedule.SingleOrDefaultAsync(m => m.ID == id);

            if (Schedule == null)
            {
                return NotFound();
            }
            return Page();
        }

        public async Task<IActionResult> OnPostAsync(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            Schedule = await _context.Schedule.FindAsync(id);

            if (Schedule != null)
            {
                _context.Schedule.Remove(Schedule);
                await _context.SaveChangesAsync();
            }

            return RedirectToPage("./Index");
        }
    }
}

```

O método `OnPostAsync` lida com a exclusão do agendamento pelo seu `id`:

```
public async Task<IActionResult> OnPostAsync(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    Schedule = await _context.Schedule.FindAsync(id);

    if (Schedule != null)
    {
        _context.Schedule.Remove(Schedule);
        await _context.SaveChangesAsync();
    }

    return RedirectToPage("./Index");
}
```

```
public async Task<IActionResult> OnPostAsync(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    Schedule = await _context.Schedule.FindAsync(id);

    if (Schedule != null)
    {
        _context.Schedule.Remove(Schedule);
        await _context.SaveChangesAsync();
    }

    return RedirectToPage("./Index");
}
```

Após a exclusão com êxito do agendamento, o `RedirectToPage` envia o usuário para a página *Index.cshtml* dos agendamentos.

## A página de Razor de agendamentos de trabalho

Quando a página é carregada, os rótulos e entradas para o título do agendamento, o agendamento público e o agendamento privado são renderizados com um botão de envio:

## Upload Schedules

### Title

### Public Schedule

No file chosen

### Private Schedule

No file chosen

Selecionar o botão **Upload** sem preencher nenhum dos campos viola os atributos `[Required]` no modelo. O `ModelState` é inválido. As mensagens de erro de validação são exibidas para o usuário:

## Upload Schedules

### Title

The Title field is required.

### Public Schedule

No file chosen

The Public Schedule field is required.

### Private Schedule

No file chosen

The Private Schedule field is required.

Digite duas letras no campo de **Título**. A mensagem de validação muda para indicar que o título deve ter entre 3 e 60 caracteres:

### Title

The field Title must be a string with a minimum length of 3 and a maximum length of 60.

Quando um ou mais agendamentos são carregados, a seção **Agendamentos carregados** renderiza os agendamentos carregados:

## Loaded Schedules

Title	Uploaded (UTC)	Public Schedule Size (bytes)	Private Schedule Size (bytes)
<a href="#">Delete</a> Schedule #1	Monday, September 11, 2017 9:09:22 PM	3,805.0	2,417.0
<a href="#">Delete</a> Schedule #2	Monday, September 11, 2017 9:09:35 PM	2,023.0	997.0

O usuário pode clicar no link **Excluir** para chegar à exibição de confirmação de exclusão, na qual ele tem a oportunidade de confirmar ou cancelar a operação de exclusão.

## Solução de problemas

Para solucionar problemas de informações com `IFormFile` carregar, consulte [carregamentos de arquivos no ASP.NET Core: solução de problemas](#).

# SDK do Razor do ASP.NET Core

26/01/2019 • 8 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

O [SDK do .NET Core 2.1 ou posteriores](#) inclui o `Microsoft.NET.Sdk.Razor` MSBuild SDK (SDK do Razor). O SDK do Razor:

- Padroniza a experiência de criação, empacotamento e publicação de projetos que contêm arquivos [Razor](#) para projetos baseados no ASP.NET Core MVC.
- Inclui um conjunto de destinos, propriedades e itens predefinidos que permitem personalizar a compilação de arquivos Razor.

## Pré-requisitos

[SDK do .NET Core 2.1 ou posteriores](#)

## Usando o SDK do Razor

A maioria dos aplicativos web não são necessárias para referenciar explicitamente o SDK do Razor.

Para usar o SDK do Razor para criar bibliotecas de classe contendo exibições Razor ou Páginas Razor:

- Use `Microsoft.NET.Sdk.Razor` em vez de `Microsoft.NET.Sdk`:

```
<Project SDK="Microsoft.NET.Sdk.Razor">
  ...
</Project>
```

- Normalmente, uma referência de pacote para `Microsoft.AspNetCore.Mvc` é necessário para receber dependências adicionais que são necessárias para criar e compilar páginas Razor e exibições do Razor. No mínimo, seu projeto deve adicionar referências de pacote para:

- `Microsoft.AspNetCore.Razor.Design`
- `Microsoft.AspNetCore.Mvc.Razor.Extensions`

O `Microsoft.AspNetCore.Razor.Design` pacote fornece os destinos e tarefas de compilação do Razor para o projeto.

Os pacotes anteriores são incluídos em `Microsoft.AspNetCore.Mvc`. A marcação a seguir mostra um arquivo de projeto que usa o SDK do Razor para criar arquivos Razor para um aplicativo páginas Razor do ASP.NET Core:

```

<Project Sdk="Microsoft.NET.Sdk.Razor">

  <PropertyGroup>
    <TargetFramework>netcoreapp2.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Mvc" Version="2.1.3" />
  </ItemGroup>

</Project>

```

### WARNING

O `Microsoft.AspNetCore.Razor.Design` e `Microsoft.AspNetCore.Mvc.Razor.Extensions` pacotes são incluídos na metapacote `Microsoft`. No entanto, o menor de versão `Microsoft.AspNetCore.App` referência de pacote fornece um metapacote para o aplicativo que não inclui a versão mais recente do `Microsoft.AspNetCore.Razor.Design`. Projetos devem fazer referência a uma versão consistente do `Microsoft.AspNetCore.Razor.Design` (ou `Microsoft.AspNetCore.Mvc`) para que as correções mais recentes do tempo de compilação para Razor são incluídas. Para obter mais informações, consulte [esse problema de GitHub](#).

## Propriedades

As seguintes propriedades controlam o comportamento do SDK do Razor como parte de um build de projeto:

- `RazorCompileOnBuild` – Quando `true`, compila e emite o assembly Razor como parte da criação do projeto. Assume o padrão de `true`.
- `RazorCompileOnPublish` – Quando `true`, compila e emite o assembly Razor como parte da publicação do projeto. Assume o padrão de `true`.

As propriedades e os itens na tabela a seguir são usados para configurar entradas e saídas para o SDK do Razor.

ITENS	DESCRIÇÃO
<code>RazorGenerate</code>	Elementos de item (arquivos <code>.cshtml</code> ) que são entradas para os destinos de geração de código.
<code>RazorCompile</code>	Elementos de item ( <code>.CS</code> arquivos) que são entradas para os destinos de compilação do Razor. Use este ItemGroup para especificar arquivos adicionais a serem compilados no assembly Razor.
<code>RazorTargetAssemblyAttribute</code>	Os elementos de item usados para a codificação geram atributos para o assembly Razor. Por exemplo: <code>RazorAssemblyAttribute</code> <code>Include="System.Reflection.AssemblyMetadataAttribute"</code> <code>_Parameter1="BuildSource"</code> <code>_Parameter2="https://docs.microsoft.com/"&gt;</code>
<code>RazorEmbeddedResource</code>	Elementos de item adicionados como recursos incorporados ao assembly Razor gerado.
PROPRIEDADE	DESCRIÇÃO
<code>RazorTargetName</code>	Nome do arquivo (sem extensão) do assembly produzido pelo Razor.

PROPRIEDADE	DESCRIÇÃO
<code>RazorOutputPath</code>	O diretório de saída do Razor.
<code>RazorCompileToolset</code>	Usado para determinar o conjunto de ferramentas usado para criar o assembly do Razor. Os valores válidos são <code>Implicit</code> , <code>RazorSDK</code> e <code>PrecompilationTool</code> .
<code>EnableDefaultContentItems</code>	O padrão é <code>true</code> . Quando <code>true</code> , inclui <code>Web.config</code> , <code>.JSON</code> , e <code>.cshtml</code> arquivos como o conteúdo do projeto. Quando referenciado por meio <code>Microsoft.NET.Sdk.Web</code> , arquivos sob <code>wwwroot</code> e arquivos de configuração também estão incluídos.
<code>EnableDefaultRazorGenerateItems</code>	Quando <code>true</code> , inclui arquivos <code>.cshtml</code> de itens de <code>Content</code> em itens de <code>RazorGenerate</code> .
<code>GenerateRazorTargetAssemblyInfo</code>	Quando <code>true</code> , gera uma <code>.CS</code> arquivo que contém atributos especificados por <code>RazorAssemblyAttribute</code> e inclui o arquivo na saída da compilação.
<code>EnableDefaultRazorTargetAssemblyInfoAttributes</code>	Quando <code>true</code> , adiciona um conjunto padrão de atributos de assembly em <code>RazorAssemblyAttribute</code> .
<code>CopyRazorGenerateFilesToPublishDirectory</code>	Quando <code>true</code> , cópias <code>RazorGenerate</code> itens ( <code>.cshtml</code> ) arquivos para o diretório de publicação. Normalmente, arquivos do Razor não são necessários para um aplicativo publicado se eles participam da compilação em tempo de compilação ou tempo de publicação. Assume o padrão de <code>false</code> .
<code>CopyRefAssembliesToPublishDirectory</code>	Quando <code>true</code> , copia os itens do assembly de referência no diretório de publicação. Normalmente, os assemblies de referência não são necessários para um aplicativo publicado se a compilação do Razor ocorre em tempo de compilação ou tempo de publicação. Definido como <code>true</code> se seu aplicativo publicado requer a compilação de tempo de execução. Por exemplo, defina o valor como <code>true</code> se o aplicativo modifica <code>.cshtml</code> arquivos em tempo de execução ou usa exibições inseridas. Assume o padrão de <code>false</code> .
<code>IncludeRazorContentInPack</code>	Quando <code>true</code> , todos os itens de conteúdo do Razor ( <code>.cshtml</code> arquivos) são marcados para inclusão no pacote do NuGet gerado. Assume o padrão de <code>false</code> .
<code>EmbedRazorGenerateSources</code>	Quando <code>true</code> , adiciona itens de <code>RazorGenerate</code> ( <code>.cshtml</code> ) como arquivos incorporados ao assembly Razor gerado. Assume o padrão de <code>false</code> .
<code>UseRazorBuildServer</code>	Quando <code>true</code> , usa um processo de servidor de build persistente para descarregar o trabalho de geração de código. Seu valor padrão é <code>UseSharedCompilation</code> .

Para saber mais sobre as propriedades, confira [Propriedades do MSBuild](#).

## Destinos

O SDK do Razor define dois destinos primários:

- `RazorGenerate` – Gera código .CS arquivos de `RazorGenerate` elementos de item. Use a propriedade `RazorGenerateDependsOn` para especificar destinos adicionais que podem ser executados antes ou depois desse destino.
- `RazorCompile` – Compila gerada .CS arquivos em um assembly Razor. Use `RazorCompileDependsOn` para especificar destinos adicionais que podem ser executados antes ou depois desse destino.

### Compilação de tempo de execução de modos de exibição do Razor

- Por padrão, o SDK do Razor não publica assemblies de referência que são necessários para realizar compilação no tempo de execução. Isso resulta em falhas de compilação quando o modelo de aplicativo se baseia na compilação em tempo de execução—, por exemplo, o aplicativo usa exibições inseridas ou muda as exibições depois que o aplicativo é publicado. Defina `CopyRefAssembliesToPublishDirectory` como `true` para continuar publicando assemblies de referência.
- Para um aplicativo web, verifique se seu aplicativo for direcionado a `Microsoft.NET.Sdk.Web` SDK.

# Visão geral do ASP.NET Core MVC

30/08/2018 • 19 minutes to read • [Edit Online](#)

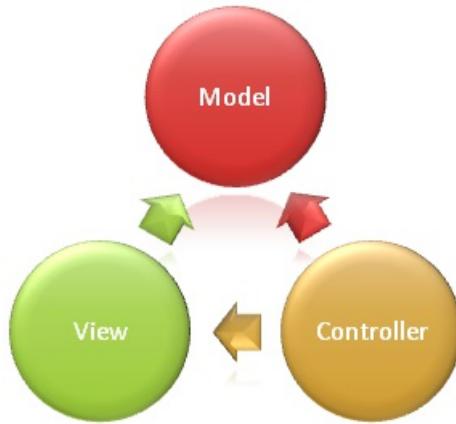
Por Steve Smith

O ASP.NET Core MVC é uma estrutura avançada para a criação de aplicativos Web e APIs usando o padrão de design Model-View-Controller.

## O que é o padrão MVC?

O padrão de arquitetura MVC (Model-View-Controller) separa um aplicativo em três grupos de componentes principais: Modelos, Exibições e Componentes. Esse padrão ajuda a obter a [separação de interesses](#). Usando esse padrão, as solicitações de usuário são encaminhadas para um Controlador, que é responsável por trabalhar com o Modelo para executar as ações do usuário e/ou recuperar os resultados de consultas. O Controlador escolhe a Exibição a ser exibida para o usuário e fornece-a com os dados do Modelo solicitados.

O seguinte diagrama mostra os três componentes principais e quais deles referenciam os outros:



Essa descrição das responsabilidades ajuda você a dimensionar o aplicativo em termos de complexidade, porque é mais fácil de codificar, depurar e testar algo (modelo, exibição ou controlador) que tem um único trabalho (e que segue o [Princípio da Responsabilidade Única](#)). É mais difícil atualizar, testar e depurar um código que tem dependências distribuídas em duas ou mais dessas três áreas. Por exemplo, a lógica da interface do usuário tende a ser alterada com mais frequência do que a lógica de negócios. Se o código de apresentação e a lógica de negócios forem combinados em um único objeto, um objeto que contém a lógica de negócios precisa ser modificado sempre que a interface do usuário é alterada. Isso costuma introduzir erros e exige um novo teste da lógica de negócios após cada alteração mínima da interface do usuário.

### NOTE

A exibição e o controlador dependem do modelo. No entanto, o modelo não depende da exibição nem do controlador. Esse é um dos principais benefícios da separação. Essa separação permite que o modelo seja criado e testado de forma independente da apresentação visual.

### Responsabilidades do Modelo

O Modelo em um aplicativo MVC representa o estado do aplicativo e qualquer lógica de negócios ou operação que deve ser executada por ele. A lógica de negócios deve ser encapsulada no modelo, juntamente com

qualquer lógica de implementação, para persistir o estado do aplicativo. As exibições fortemente tipadas normalmente usam tipos ViewModel criados para conter os dados a serem exibidos nessa exibição. O controlador cria e popula essas instâncias de ViewModel com base no modelo.

#### NOTE

Há várias maneiras de organizar o modelo em um aplicativo que usa o padrão de arquitetura MVC. Saiba mais sobre alguns [tipos diferentes de tipos de modelo](#).

### Responsabilidades da Exibição

As exibições são responsáveis por apresentar o conteúdo por meio da interface do usuário. Elas usam o [mecanismo de exibição do Razor](#) para inserir o código .NET em uma marcação HTML. Deve haver uma lógica mínima nas exibições e qualquer lógica contida nelas deve se relacionar à apresentação do conteúdo. Se você precisar executar uma grande quantidade de lógica em arquivos de exibição para exibir dados de um modelo complexo, considere o uso de um [Componente de Exibição](#), ViewModel ou um modelo de exibição para simplificar a exibição.

### Responsabilidades do Controlador

Os controladores são os componentes que cuidam da interação do usuário, trabalham com o modelo e, em última análise, selecionam uma exibição a ser renderizada. Em um aplicativo MVC, a exibição mostra apenas informações; o controlador manipula e responde à entrada e à interação do usuário. No padrão MVC, o controlador é o ponto de entrada inicial e é responsável por selecionar quais tipos de modelo serão usados para o trabalho e qual exibição será renderizada (daí seu nome – ele controla como o aplicativo responde a determinada solicitação).

#### NOTE

Os controladores não devem ser excessivamente complicados por muitas responsabilidades. Para evitar que a lógica do controlador se torne excessivamente complexa, use o [Princípio da Responsabilidade Única](#) para empurrar a lógica de negócios para fora do controlador e inseri-la no modelo de domínio.

#### TIP

Se você achar que as ações do controlador executam com frequência os mesmos tipos de ações, siga o [Princípio Don't Repeat Yourself](#) movendo essas ações comuns para [filtros](#).

## O que é ASP.NET Core MVC

A estrutura do ASP.NET Core MVC é uma estrutura de apresentação leve, de software livre e altamente testável, otimizada para uso com o ASP.NET Core.

ASP.NET Core MVC fornece uma maneira com base em padrões para criar sites dinâmicos que habilitam uma separação limpa de preocupações. Ele lhe dá controle total sobre a marcação, dá suporte ao desenvolvimento amigável a TDD e usa os padrões da web mais recentes.

## Recursos

ASP.NET Core MVC inclui o seguinte:

- [Roteamento](#)
- [Associação de modelos](#)
- [Validação de modelo](#)

- [Injeção de dependência](#)
- [Filtros](#)
- [Áreas](#)
- [APIs Web](#)
- [Capacidade de teste](#)
- [Mecanismo de exibição do Razor](#)
- [Exibições fortemente tipadas](#)
- [Auxiliares de marcação](#)
- [Componentes da exibição](#)

## Roteamento

O ASP.NET Core MVC baseia-se no [roteamento do ASP.NET Core](#), um componente de mapeamento de URL avançado que permite criar aplicativos que têm URLs comprehensíveis e pesquisáveis. Isso permite que você defina padrões de nomenclatura de URL do aplicativo que funcionam bem para SEO (otimização do mecanismo de pesquisa) e para a geração de links, sem levar em consideração como os arquivos no servidor Web estão organizados. Defina as rotas usando uma sintaxe de modelo de rota conveniente que dá suporte a restrições de valor de rota, padrões e valores opcionais.

O *roteamento baseado em convenção* permite definir globalmente os formatos de URL aceitos pelo aplicativo e como cada um desses formatos é mapeado para um método de ação específico em determinado controlador. Quando uma solicitação de entrada é recebida, o mecanismo de roteamento analisa a URL e corresponde-a a um dos formatos de URL definidos. Em seguida, ele chama o método de ação do controlador associado.

```
routes.MapRoute(name: "Default", template: "{controller=Home}/{action=Index}/{id?}");
```

O *roteamento de atributo* permite que você especifique as informações de roteamento decorando os controladores e as ações com atributos que definem as rotas do aplicativo. Isso significa que as definições de rota são colocadas ao lado do controlador e da ação aos quais estão associadas.

```
[Route("api/[controller]")]
public class ProductsController : Controller
{
    [HttpGet("{id}")]
    public IActionResult GetProduct(int id)
    {
        ...
    }
}
```

## Associação de modelos

ASP.NET Core MVC [associação de modelo](#) converte dados de solicitação de cliente (valores de formulário, os dados de rota, parâmetros de cadeia de caracteres de consulta, os cabeçalhos HTTP) em objetos que o controlador pode manipular. Como resultado, a lógica de controlador não precisa fazer o trabalho de descobrir os dados de solicitação de entrada; ele simplesmente tem os dados como parâmetros para os métodos de ação.

```
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null) { ... }
```

## Validação de modelo

O ASP.NET Core MVC dá suporte à [validação](#) pela decoração do objeto de modelo com atributos de validação de anotação de dados. Os atributos de validação são verificados no lado do cliente antes que os valores sejam postados no servidor, bem como no servidor antes que a ação do controlador seja chamada.

```

using System.ComponentModel.DataAnnotations;
public class LoginViewModel
{
    [Required]
    [EmailAddress]
    public string Email { get; set; }

    [Required]
    [DataType(DataType.Password)]
    public string Password { get; set; }

    [Display(Name = "Remember me?")]
    public bool RememberMe { get; set; }
}

```

Uma ação do controlador:

```

public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null)
{
    if (ModelState.IsValid)
    {
        // work with the model
    }
    // At this point, something failed, redisplay form
    return View(model);
}

```

A estrutura manipula a validação dos dados de solicitação no cliente e no servidor. A lógica de validação especificada em tipos de modelo é adicionada às exibições renderizados como anotações não invasivas e é imposta no navegador com o [jQuery Validation](#).

## Injeção de dependência

O ASP.NET Core tem suporte interno para DI ([injeção de dependência](#)). No ASP.NET Core MVC, os [controladores](#) podem solicitar serviços necessários por meio de seus construtores, possibilitando o acompanhamento do [princípio de dependências explícitas](#).

O aplicativo também pode usar a [injeção de dependência em arquivos no exibição](#), usando a diretiva `@inject`:

```

@inject SomeService ServiceName
<!DOCTYPE html>
<html lang="en">
<head>
    <title>@ServiceName.GetTitle</title>
</head>
<body>
    <h1>@ServiceName.GetTitle</h1>
</body>
</html>

```

## Filtros

Os [filtros](#) ajudam os desenvolvedores a encapsular interesses paralelos, como tratamento de exceção ou autorização. Os filtros permitem a execução de uma lógica pré e pós-processamento personalizada para métodos de ação e podem ser configurados para execução em determinados pontos no pipeline de execução de uma solicitação específica. Os filtros podem ser aplicados a controladores ou ações como atributos (ou podem ser executados globalmente). Vários filtros (como `Authorize`) são incluídos na estrutura. `[Authorize]` é o atributo usado para criar filtros de autorização do MVC.

```
[Authorize]
public class AccountController : Controller
{
```

## Áreas

As [áreas](#) fornecem uma maneira de particionar um aplicativo Web ASP.NET Core MVC grande em agrupamentos funcionais menores. Uma área é uma estrutura MVC dentro de um aplicativo. Em um projeto MVC, componentes lógicos como Modelo, Controlador e Exibição são mantidos em pastas diferentes e o MVC usa convenções de nomenclatura para criar a relação entre esses componentes. Para um aplicativo grande, pode ser vantajoso particionar o aplicativo em áreas de nível alto separadas de funcionalidade. Por exemplo, um aplicativo de comércio eletrônico com várias unidades de negócios, como check-out, cobrança e pesquisa, etc. Cada uma dessas unidades têm suas próprias exibições de componente lógico, controladores e modelos.

## APIs da Web

Além de ser uma ótima plataforma para a criação de sites, o ASP.NET Core MVC tem um excelente suporte para a criação de APIs Web. Crie serviços que alcançam uma ampla gama de clientes, incluindo navegadores e dispositivos móveis.

A estrutura inclui suporte para a negociação de conteúdo HTTP com suporte interno para [formatar dados](#) como JSON ou XML. Escreva [formatadores personalizados](#) para adicionar suporte para seus próprios formatos.

Use a geração de links para habilitar o suporte para hipermídia. Habilite o suporte para o [CORS \(Compartilhamento de Recursos Entre Origens\)](#) com facilidade, de modo que as APIs Web possam ser compartilhadas entre vários aplicativos Web.

## Capacidade de teste

O uso pela estrutura da injecão de dependência e de interfaces a torna adequada para teste de unidade. Além disso, a estrutura inclui recursos (como um provedor TestHost e InMemory para o Entity Framework) que também agiliza e facilita a execução de [testes de integração](#). Saiba mais sobre [como testar a lógica do controlador](#).

## Mecanismo de exibição do Razor

As [exibições do ASP.NET Core MVC](#) usam o [mecanismo de exibição do Razor](#) para renderizar exibições. Razor é uma linguagem de marcação de modelo compacta, expressiva e fluida para definir exibições usando um código C# inserido. O Razor é usado para gerar o conteúdo da Web no servidor de forma dinâmica. Você pode combinar o código do servidor com o código e o conteúdo do lado cliente de maneira limpa.

```
<ul>
    @for (int i = 0; i < 5; i++) {
        <li>List item @i</li>
    }
</ul>
```

Usando o mecanismo de exibição do Razor, você pode definir [layouts](#), [exibições parciais](#) e seções substituíveis.

## Exibições fortemente tipadas

As exibições do Razor no MVC podem ser fortemente tipadas com base no modelo. Os controladores podem passar um modelo fortemente tipado para as exibições, permitindo que elas tenham a verificação de tipo e o suporte do IntelliSense.

Por exemplo, a seguinte exibição renderiza um modelo do tipo `IEnumerable<Product>` :

```
@model IEnumerable<Product>
<ul>
    @foreach (Product p in Model)
    {
        <li>@p.Name</li>
    }
</ul>
```

## Auxiliares de Marca

Os [Auxiliares de Marca](#) permitem que o código do servidor participe da criação e renderização de elementos HTML em arquivos do Razor. Use auxiliares de marca para definir marcas personalizadas (por exemplo, `<environment>`) ou para modificar o comportamento de marcas existentes (por exemplo, `<label>`). Os Auxiliares de Marca associam a elementos específicos com base no nome do elemento e seus atributos. Eles oferecem os benefícios da renderização do lado do servidor, enquanto preservam uma experiência de edição de HTML.

Há muitos Auxiliares de Marca internos para tarefas comuns – como criação de formulários, links, carregamento de ativos e muito mais – e ainda outros disponíveis em repositórios GitHub públicos e como NuGet. Os Auxiliares de Marca são criados no C# e são direcionados a elementos HTML de acordo com o nome do elemento, o nome do atributo ou a marca pai. Por exemplo, o `LinkTagHelper` interno pode ser usado para criar um link para a ação `Login` do `AccountsController`:

```
<p>
    Thank you for confirming your email.
    Please <a asp-controller="Account" asp-action="Login">Click here to Log in</a>.
</p>
```

O `EnvironmentTagHelper` pode ser usado para incluir scripts diferentes nas exibições (por exemplo, bruto ou minimizado) de acordo com o ambiente de tempo de execução, como Desenvolvimento, Preparo ou Produção:

```
<environment names="Development">
    <script src="~/lib/jquery/dist/jquery.js"></script>
</environment>
<environment names="Staging,Production">
    <script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-2.1.4.min.js"
           asp-fallback-src="~/lib/jquery/dist/jquery.min.js"
           asp-fallback-test="window.jQuery">
    </script>
</environment>
```

Os Auxiliares de Marca fornecem uma experiência de desenvolvimento amigável a HTML e um ambiente avançado do IntelliSense para a criação de HTML e marcação do Razor. A maioria dos Auxiliares de Marca internos é direcionada a elementos HTML existentes e fornece atributos do lado do servidor para o elemento.

## Componentes da exibição

Os [Componentes de Exibição](#) permitem que você empacote a lógica de renderização e reutilize-a em todo o aplicativo. São semelhantes às [exibições parciais](#), mas com a lógica associada.

## Versão de compatibilidade

O método `SetCompatibilityVersion` permite que um aplicativo aceite ou recuse as possíveis alterações da falha de comportamento introduzidas no ASP.NET Core MVC 2.1 ou posteriores.

Para obter mais informações, consulte [Versão de compatibilidade do ASP.NET Core MVC](#).

# Criar um aplicativo Web com o ASP.NET Core MVC

23/01/2019 • 2 minutes to read • [Edit Online](#)

Este tutorial ensina a usar o desenvolvimento Web do ASP.NET Core MVC com controladores e exibições. Se você é novo no desenvolvimento da Web ASP.NET Core, considere a versão [Razor Pages](#) deste tutorial, que oferece um ponto inicial mais simples.

A série de tutoriais inclui o seguinte:

1. [Introdução](#)
2. [Adicionar um controlador](#)
3. [Adicionar uma exibição](#)
4. [Adicionar um modelo](#)
5. [Trabalhar com o SQL Server LocalDB](#)
6. [Exibições e métodos do controlador](#)
7. [Adicionar pesquisa](#)
8. [Adicionar um novo campo](#)
9. [Adicionar validação](#)
10. [Examinar os métodos Details e Delete](#)

# Introdução ao ASP.NET Core MVC

10/01/2019 • 10 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

Este tutorial ensina a usar o desenvolvimento Web do ASP.NET Core MVC com controladores e exibições. Se você é novo no desenvolvimento da Web ASP.NET Core, considere a versão [Razor Pages](#) deste tutorial, que oferece um ponto inicial mais simples.

<https://docs.microsoft.com/en-us/visualstudio/ide/visual-studio-ide?view=vs-2017>

Este tutorial ensina as noções básicas de criação de um aplicativo Web ASP.NET Core MVC.

O aplicativo gerencia um banco de dados de títulos de filmes. Você aprenderá como:

- Criar um aplicativo Web.
- Adicionar e gerar o scaffolding de um modelo.
- Trabalhar com um banco de dados.
- Adicionar pesquisa e validação.

No final, você terá um aplicativo que pode gerenciar e exibir dados de filmes.

[Exibir ou baixar um código de exemplo \(como baixar\).](#)

## NOTE

Estamos testando a usabilidade de uma nova estrutura proposta para o sumário do ASP.NET Core. Se você tiver alguns minutos para experimentar um exercício de localização de sete tópicos diferentes no sumário atual ou proposto, [clique aqui para participar do estudo](#).

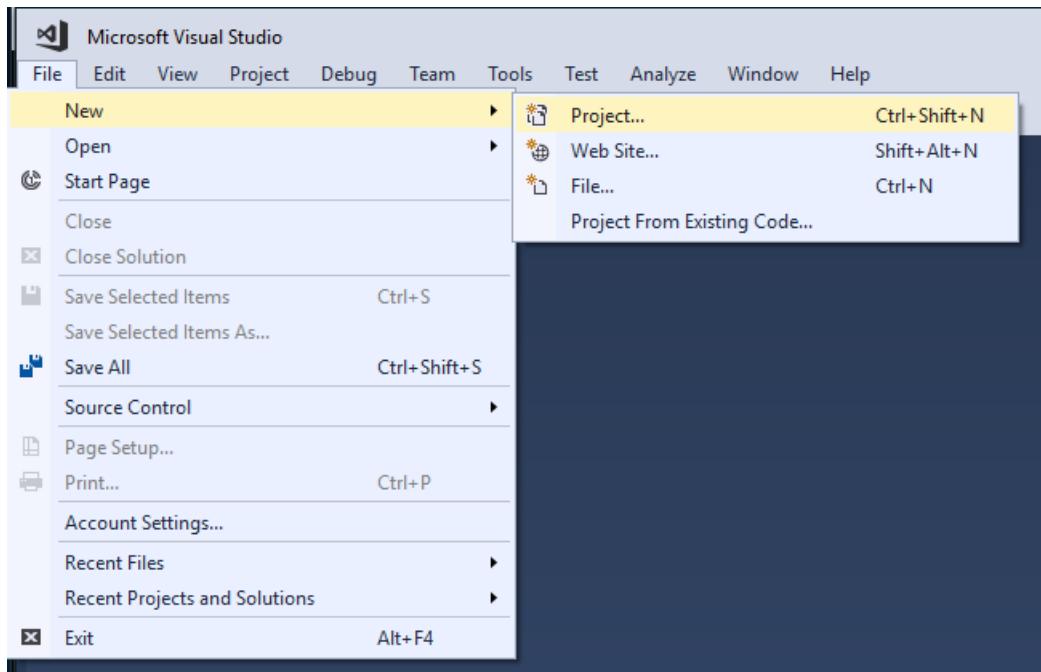
## Pré-requisitos

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- [Visual Studio 2017 versão 15.9 ou posterior](#) com a carga de trabalho **ASP.NET e desenvolvimento para a Web**
- [SDK 2.2 ou posterior do .NET Core](#)

## Como criar um aplicativo Web

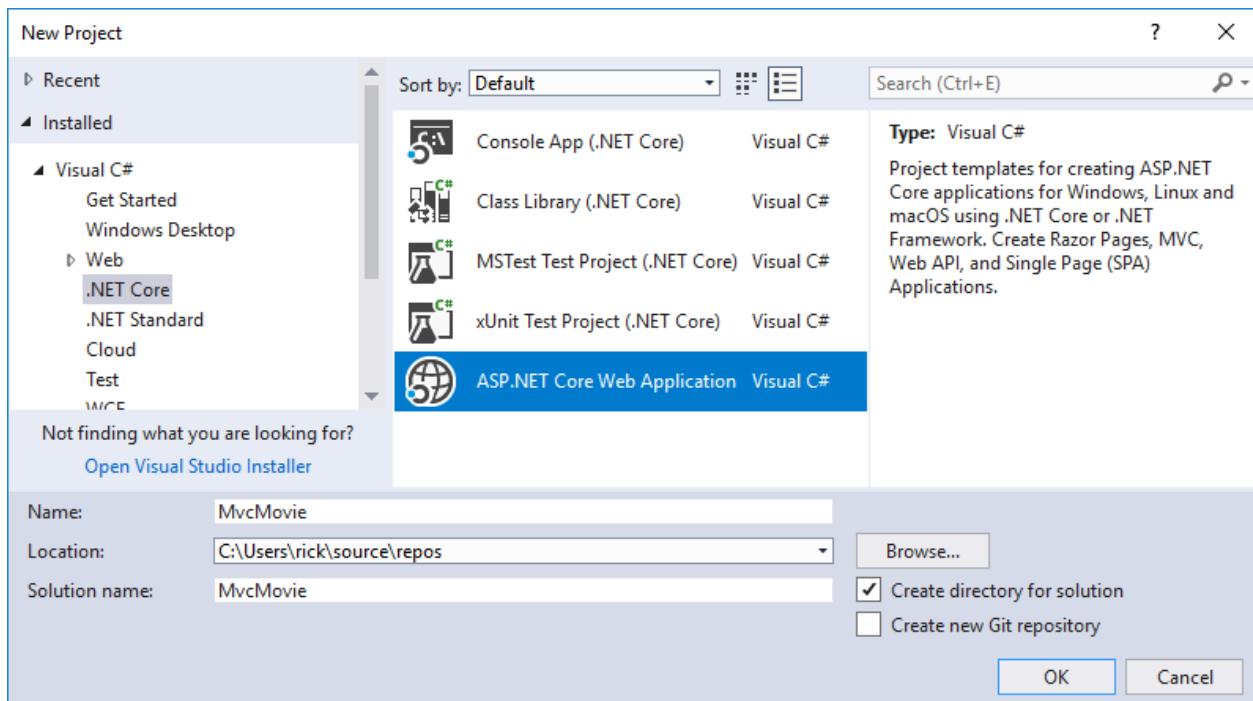
- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)

No Visual Studio, selecione **Arquivo > Novo > Projeto**.



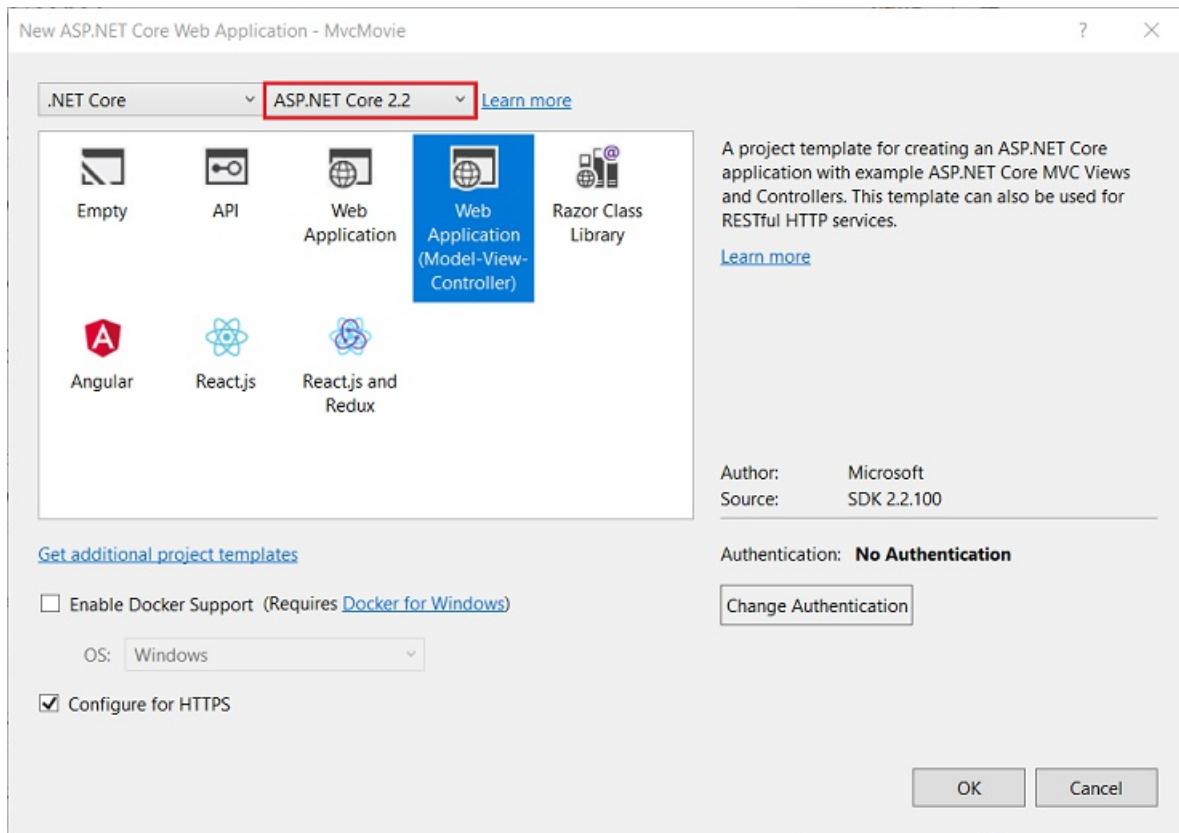
Complete a caixa de diálogo **Novo Projeto**:

- No painel esquerdo, selecione **.NET Core**
- No painel central, selecione **Aplicativo Web ASP.NET Core (.NET Core)**
- Nomeie o projeto "MvcMovie" (é importante nomear o projeto "MvcMovie" para que, quando você copiar o código, o namespace corresponda).
- Selecione **OK**



Faça as configurações necessárias na caixa de diálogo **Novo aplicativo Web ASP.NET Core (.NET Core) – MvcMovie**:

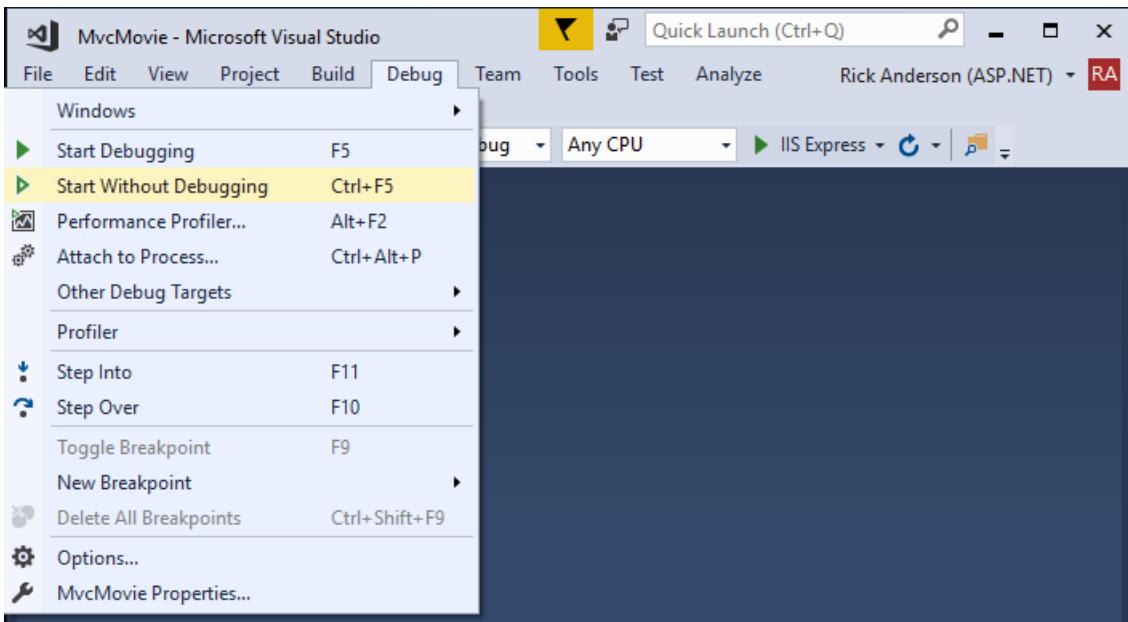
- Na caixa de lista suspensa do seletor de versão, selecione **ASP.NET Core 2.2**
- Selecione **Aplicativo Web (Model-View-Controller)**
- Selecione **OK**.



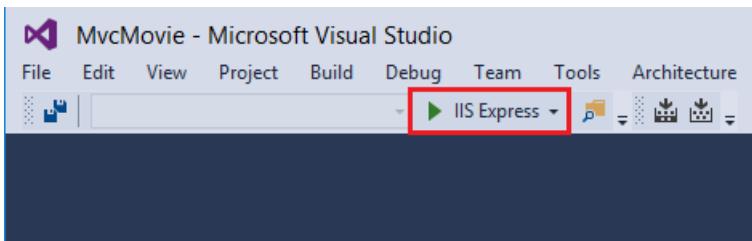
O Visual Studio usou um modelo padrão para o projeto MVC que você acabou de criar. Para que o aplicativo comece a funcionar agora mesmo, digite um nome de projeto e selecione algumas opções. Este é um projeto inicial básico e é um bom ponto de partida.

Pressione **Ctrl+F5** para executar o aplicativo no modo sem depuração.

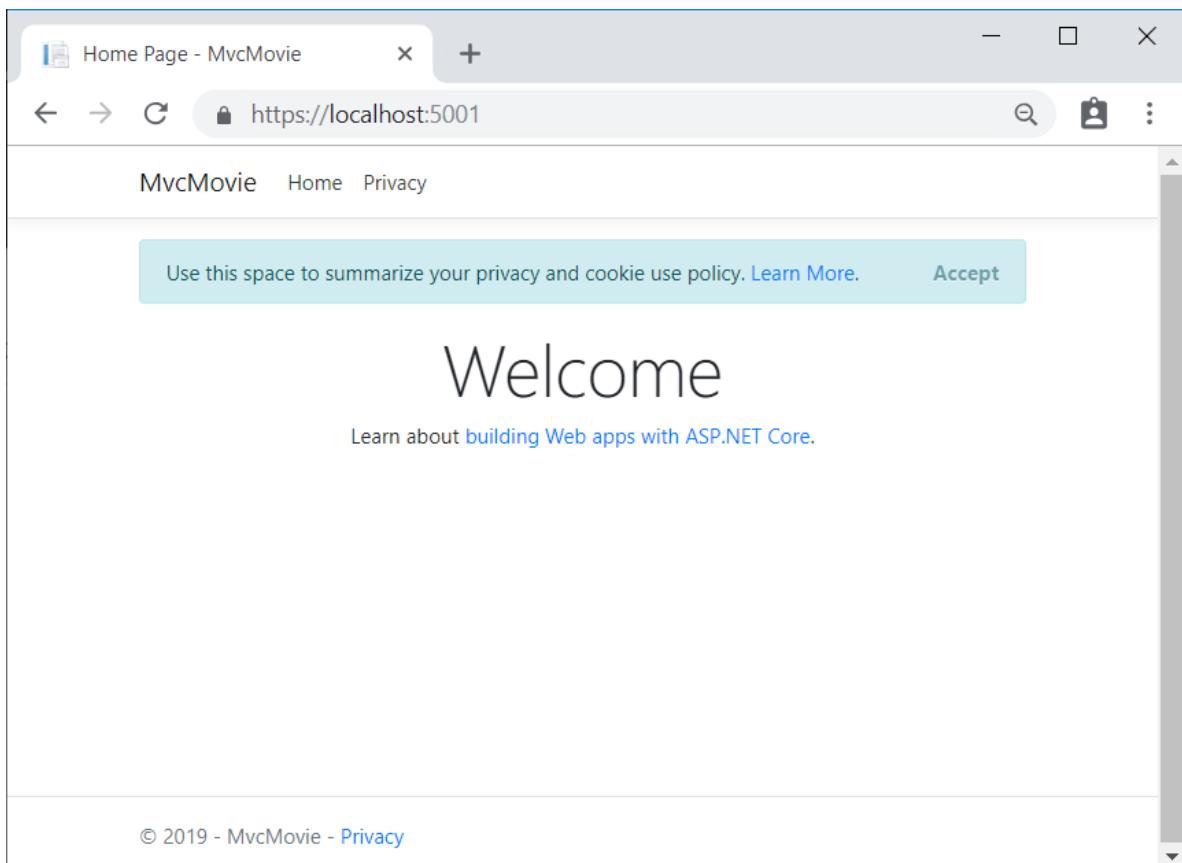
- O Visual Studio inicia o **IIS Express** e executa o aplicativo. Observe que a barra de endereços mostra `localhost:port#` e não algo como `example.com`. Isso ocorre porque `localhost` é o nome do host padrão do computador local. Quando o Visual Studio cria um projeto Web, uma porta aleatória é usada para o servidor Web. Na imagem acima, o número da porta é 5000. A URL no navegador mostra `localhost:5000`. Quando você executar o aplicativo, verá um número de porta diferente.
- Iniciar o aplicativo com **Ctrl+F5** (modo de não depuração) permite que você faça alterações de código, salve o arquivo, atualize o navegador e veja as alterações de código. Muitos desenvolvedores preferem usar modo de não depuração para iniciar o aplicativo e exibir alterações rapidamente.
- Você pode iniciar o aplicativo no modo de não depuração ou de depuração por meio do item de menu **Depurar:**



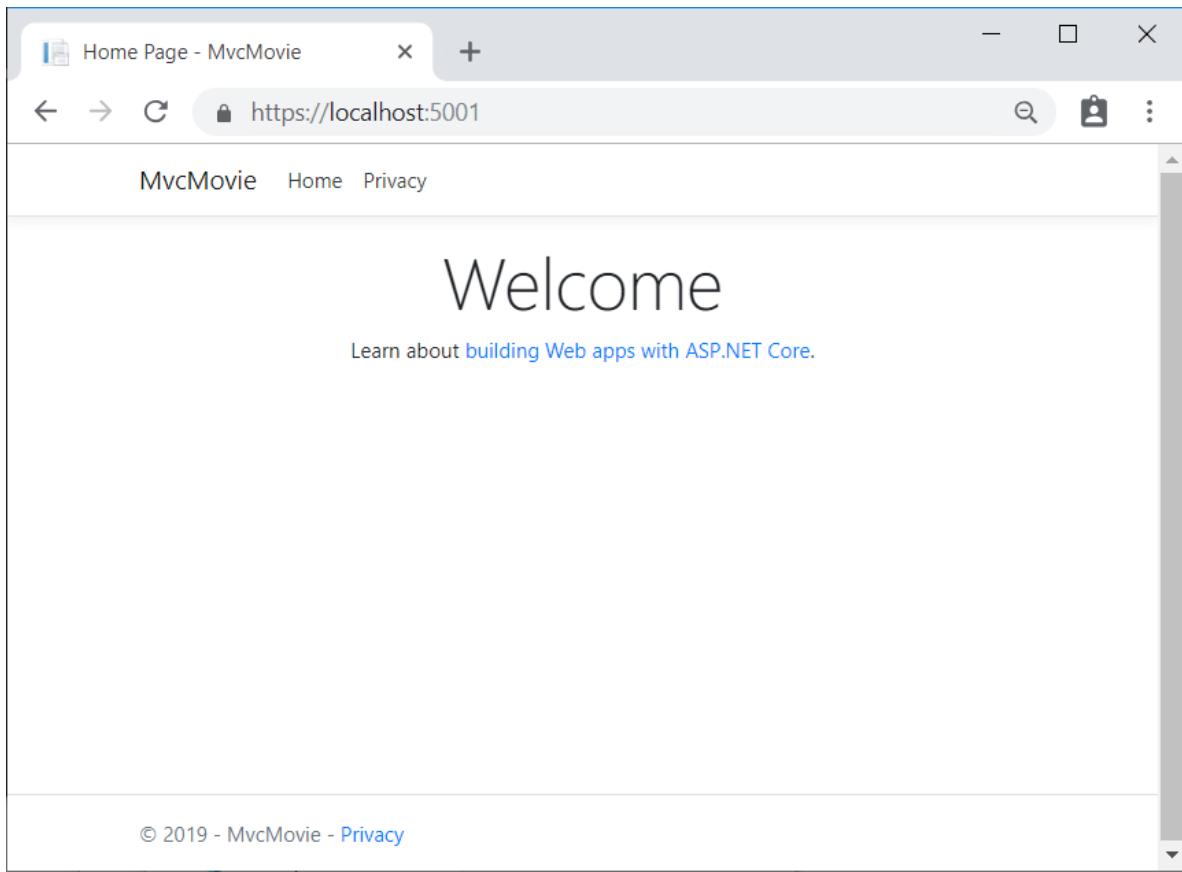
- Você pode depurar o aplicativo selecionando o botão **IIS Express**



- Selecione **Aceitar** para dar consentimento de rastreamento. Este aplicativo não acompanha informações pessoais. O código de modelo gerado inclui ativos para ajudar a cumprir o [RGPD \(Regulamento Geral sobre a Proteção de Dados\)](#).



A imagem a seguir mostra o aplicativo depois de aceitar o rastreamento:



- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)

## Ajuda do Visual Studio

- [Aprenda a depurar o código C# usando o Visual Studio](#)
- [Introdução ao IDE do Visual Studio](#)

Na próxima parte deste tutorial, você saberá mais sobre o MVC e começará a escrever um pouco de código.

[AVANÇAR](#)

# Adicionar um controlador a um aplicativo ASP.NET Core MVC

10/01/2019 • 12 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

O padrão de arquitetura MVC (Model-View-Controller) separa um aplicativo em três componentes principais: **Model**, **View** e **Controller**. O padrão MVC ajuda a criar aplicativos que são mais testáveis e fáceis de atualizar comparado aos aplicativos monolíticos tradicionais. Os aplicativos baseados no MVC contêm:

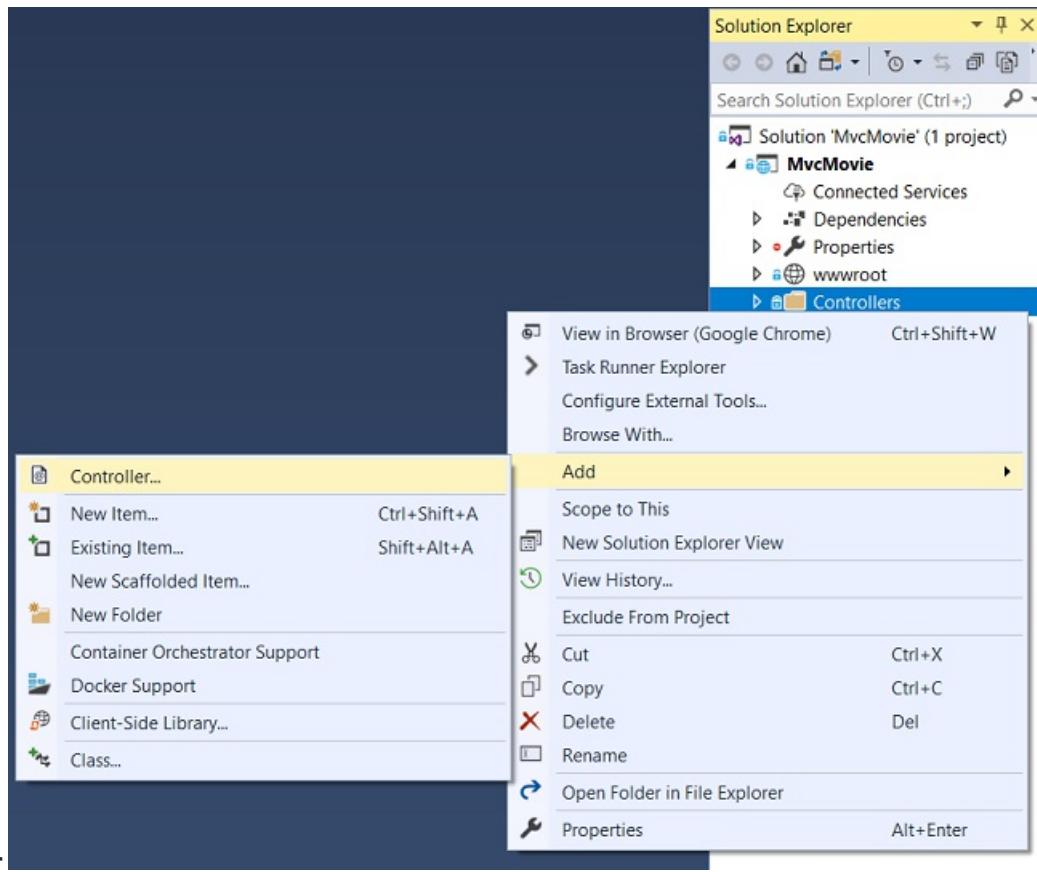
- **Modelos**: classes que representam os dados do aplicativo. As classes de modelo usam a lógica de validação para impor regras de negócio aos dados. Normalmente, os objetos de modelo recuperam e armazenam o estado do modelo em um banco de dados. Neste tutorial, um modelo `Movie` recupera dados de filmes de um banco de dados, fornece-os para a exibição ou atualiza-os. Os dados atualizados são gravados em um banco de dados.
- **Exibições**: são os componentes que exibem a interface do usuário do aplicativo. Em geral, essa interface do usuário exibe os dados de modelo.
- **Controladores**: classes que manipulam as solicitações do navegador. Elas recuperam dados de modelo e chamam modelos de exibição que retornam uma resposta. Em um aplicativo MVC, a exibição mostra apenas informações; o controlador manipula e responde à entrada e à interação do usuário. Por exemplo, o controlador manipula os dados de rota e os valores da cadeia de consulta e passa esses valores para o modelo. O modelo pode usar esses valores para consultar o banco de dados. Por exemplo,  
`https://localhost:1234/Home/About` tem dados de rota de `Home` (o controlador) e `About` (o método de ação a ser chamado no controlador principal). `https://localhost:1234/Movies/Edit/5` é uma solicitação para editar o filme com ID=5 usando o controlador do filme. Os dados de rota são explicados posteriormente no tutorial.

O padrão MVC ajuda a criar aplicativos que separam os diferentes aspectos do aplicativo (lógica de entrada, lógica de negócios e lógica da interface do usuário), ao mesmo tempo que fornece um acoplamento flexível entre esses elementos. O padrão especifica o local em que cada tipo de lógica deve estar localizado no aplicativo. A lógica da interface do usuário pertence à exibição. A lógica de entrada pertence ao controlador. A lógica de negócios pertence ao modelo. Essa separação ajuda a gerenciar a complexidade ao criar um aplicativo, porque permite que você trabalhe em um aspecto da implementação por vez, sem afetar o código de outro. Por exemplo, você pode trabalhar no código de exibição sem depender do código da lógica de negócios.

Abrangemos esses conceitos nesta série de tutoriais e mostraremos como usá-los para criar um aplicativo de filme. O projeto MVC contém pastas para os *Controladores* e as *Exibições*.

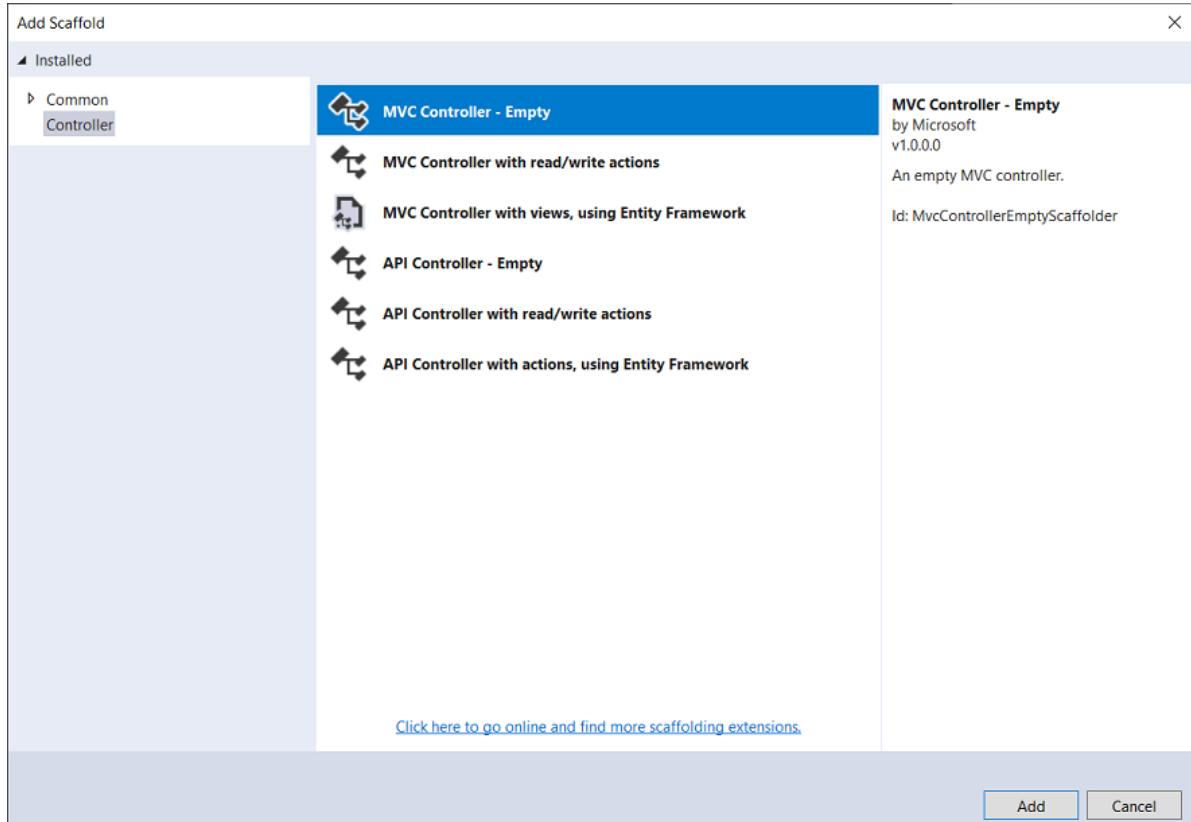
## Adicionar um controlador

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- No **Gerenciador de Soluções**, clique com o botão direito do mouse em **Controladores** > **Adicionar >**



### Controlador

- Na caixa de diálogo **Adicionar Scaffold**, selecione **Controlador MVC – Vazio**



- Na caixa de diálogo **Adicionar Controlador MVC Vazio**, insira **HelloWorldController** e selecione **ADICIONAR**.

Substitua o conteúdo de *Controllers/HelloWorldController.cs* pelo seguinte:

```
using Microsoft.AspNetCore.Mvc;
using System.Text.Encodings.Web;

namespace MvcMovie.Controllers
{
    public class HelloWorldController : Controller
    {
        //
        // GET: /HelloWorld/

        public string Index()
        {
            return "This is my default action...";
        }

        //
        // GET: /HelloWorld/Welcome/

        public string Welcome()
        {
            return "This is the Welcome action method...";
        }
    }
}
```

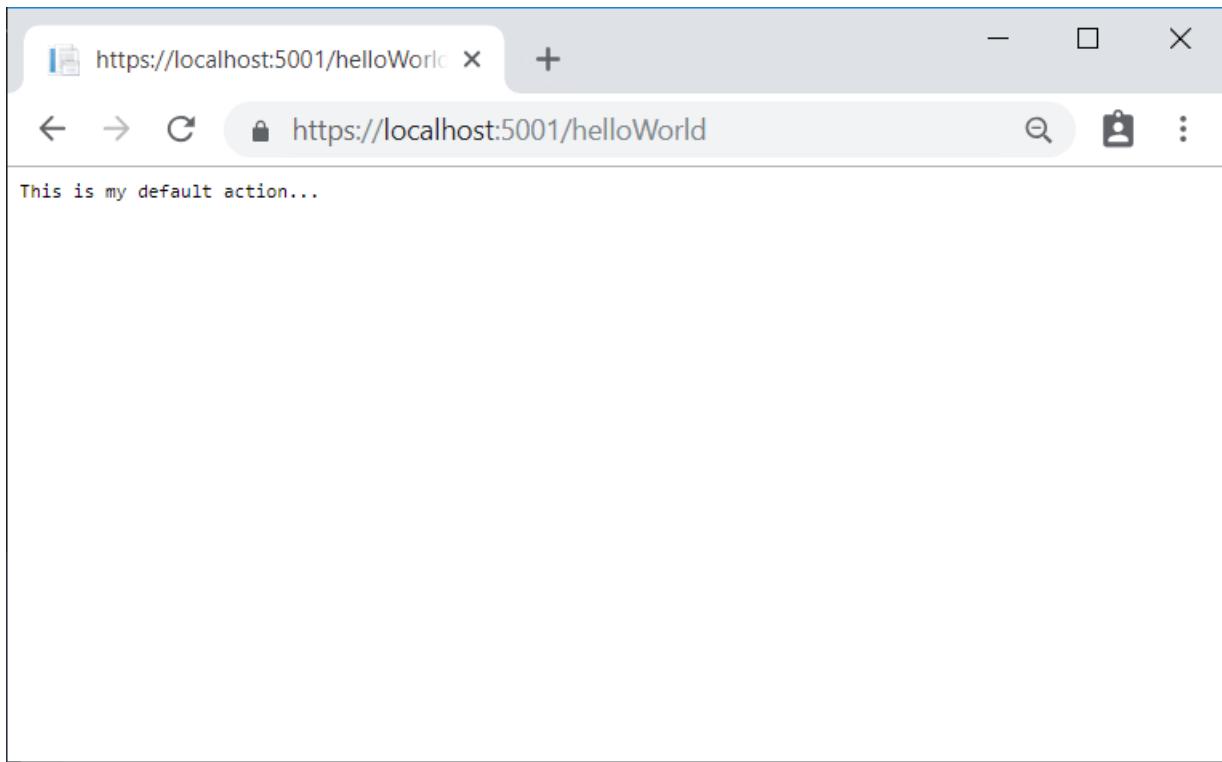
Cada método `public` em um controlador pode ser chamado como um ponto de extremidade HTTP. Na amostra acima, ambos os métodos retornam uma cadeia de caracteres. Observe os comentários que precedem cada método.

Um ponto de extremidade HTTP é uma URL direcionável no aplicativo Web, como

`https://localhost:5001/HelloWorld`, e combina o protocolo usado `HTTPS`, o local de rede do servidor Web (incluindo a porta TCP) `localhost:5001` e o URI de destino `HelloWorld`.

O primeiro comentário indica que este é um método **HTTP GET** invocado por meio do acréscimo de `/HelloWorld/` à URL base. O primeiro comentário especifica um método **HTTP GET** invocado por meio do acréscimo de `/HelloWorld/Welcome/` à URL base. Mais adiante no tutorial, o mecanismo de scaffolding será usado para gerar métodos `HTTP POST` que atualizam dados.

Execute o aplicativo no modo sem depuração e acrescente “HelloWorld” ao caminho na barra de endereços. O método `Index` retorna uma cadeia de caracteres.



O MVC invoca as classes do controlador (e os métodos de ação dentro delas), dependendo da URL de entrada. A [lógica de roteamento de URL](#) padrão usada pelo MVC usa um formato como este para determinar o código a ser invocado:

```
/[Controller]/[ActionName]/[Parameters]
```

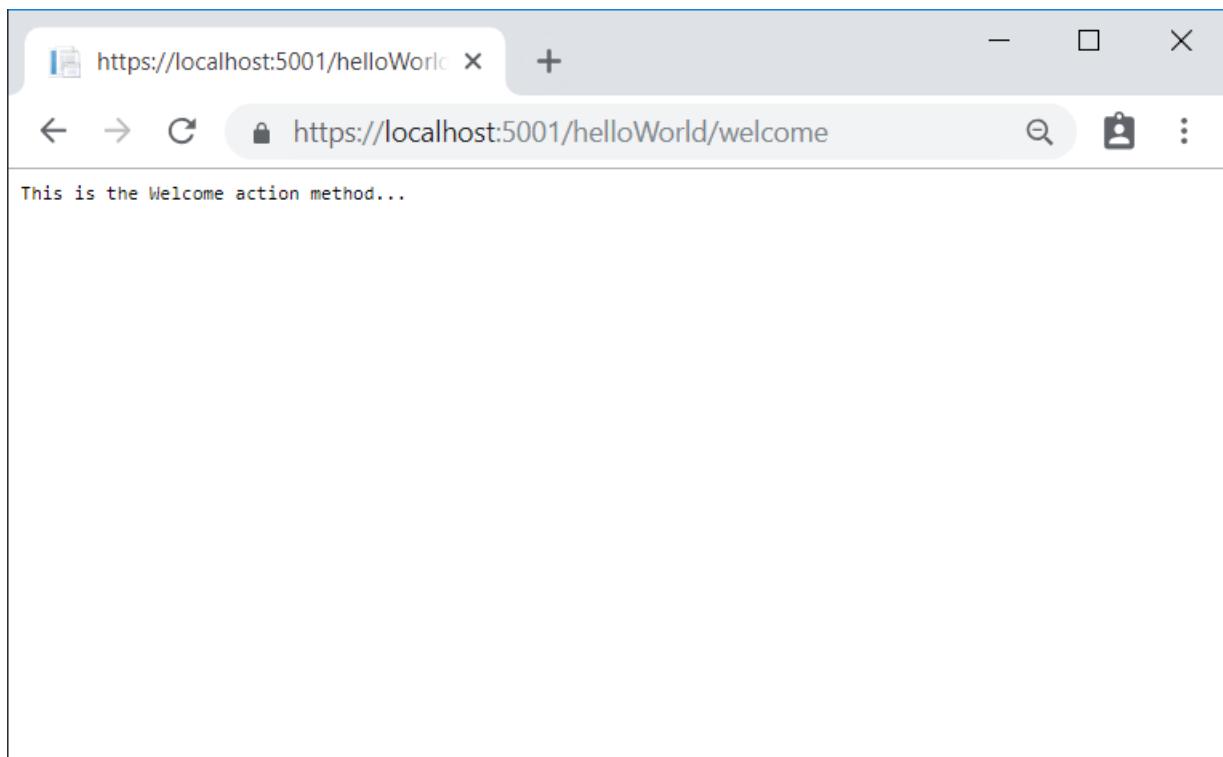
O formato de roteamento é definido no método `Configure` no arquivo `Startup.cs`.

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});
```

Quando você acessa o aplicativo e não fornece nenhum segmento de URL, ele usa como padrão o controlador "Home" e o método "Index" especificado na linha do modelo realçada acima.

O primeiro segmento de URL determina a classe do controlador a ser executada. Portanto, `localhost:xxxx/HelloWorld` é mapeado para a classe `HelloWorldController`. A segunda parte do segmento de URL determina o método de ação na classe. Portanto, `localhost:xxxx/HelloWorld/Index` fará com que o método `Index` da classe `HelloWorldController` seja executado. Observe que você precisou apenas navegar para `localhost:xxxx/HelloWorld` e o método `Index` foi chamado por padrão. Isso ocorre porque `Index` é o método padrão que será chamado em um controlador se um nome de método não for especificado explicitamente. A terceira parte do segmento de URL (`id`) refere-se aos dados de rota. Os dados de rota são explicados posteriormente no tutorial.

Navegue para `https://localhost:xxxx/HelloWorld/Welcome`. O método `Welcome` é executado e retorna a cadeia de caracteres `This is the Welcome action method...`. Para essa URL, o controlador é `HelloWorld` e `Welcome` é o método de ação. Você ainda não usou a parte `[Parameters]` da URL.



Modifique o código para passar algumas informações de parâmetro da URL para o controlador. Por exemplo, `/HelloWorld/Welcome?name=Rick&numtimes=4`. Altere o método `Welcome` para incluir dois parâmetros, conforme mostrado no código a seguir.

```
// GET: /HelloWorld/Welcome/
// Requires using System.Text.Encodings.Web;
public string Welcome(string name, int numTimes = 1)
{
    return HtmlEncoder.Default.Encode($"Hello {name}, NumTimes is: {numTimes}");
}
```

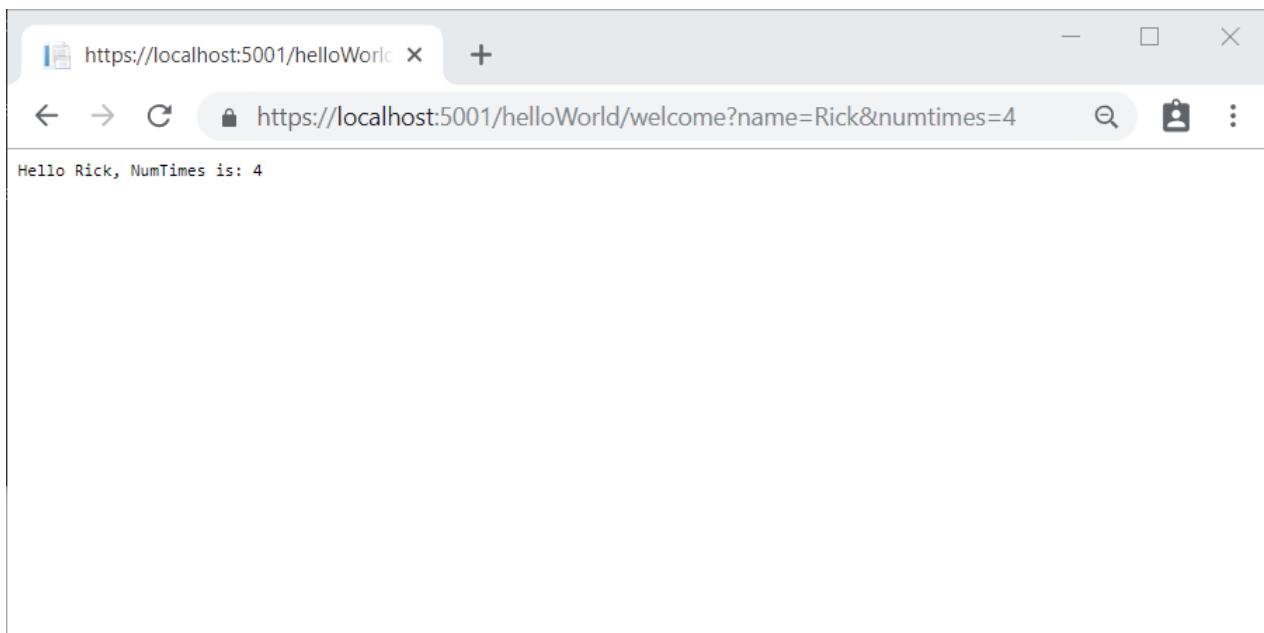
O código anterior:

- Usa o recurso de parâmetro opcional do C# para indicar que o parâmetro `numTimes` usa 1 como padrão se nenhum valor é passado para esse parâmetro.
- Usa `HtmlEncoder.Default.Encode` para proteger o aplicativo contra a entrada mal-intencionada (ou seja, JavaScript).
- Usa [Cadeias de caracteres interpoladas](#) em  `$"Hello {name}, NumTimes is: {numTimes}"`.

Execute o aplicativo e navegue até:

```
https://localhost:xxxx>HelloWorld/Welcome?name=Rick&numtimes=4
```

(Substitua xxxx pelo número da porta.) Você pode tentar valores diferentes para `name` e `numtimes` na URL. O sistema de [model binding](#) do MVC mapeia automaticamente os parâmetros nomeados da cadeia de consulta na barra de endereços para os parâmetros no método. Consulte [Model binding](#) para obter mais informações.



Na imagem acima, o segmento de URL (`Parameters`) não é usado e os parâmetros `name` e `numTimes` são transmitidos como **cadeias de consulta**. O `?` (ponto de interrogação) na URL acima é um separador seguido pelas cadeias de consulta. O caractere `&` separa as cadeias de consulta.

Substitua o método `Welcome` pelo seguinte código:

```
public string Welcome(string name, int ID = 1)
{
    return HtmlEncoder.Default.Encode($"Hello {name}, ID: {ID}");
}
```

Execute o aplicativo e insira a seguinte URL: `https://localhost:xxx/HelloWorld/Welcome/3?name=Rick`

Agora, o terceiro segmento de URL correspondeu ao parâmetro de rota `id`. O método `Welcome` contém um parâmetro `id` que correspondeu ao modelo de URL no método `MapRoute`. O `?` à direita (em `id?`) indica que o parâmetro `id` é opcional.

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});
```

Nestes exemplos, o controlador faz a parte "VC" do MVC – ou seja, o trabalho da exibição e do controlador. O controlador retorna o HTML diretamente. Em geral, você não deseja que os controladores retornem HTML diretamente, pois isso é muito difícil de codificar e manter. Em vez disso, normalmente, você usa um arquivo de modelo de exibição do Razor separado para ajudar a gerar a resposta HTML. Faça isso no próximo tutorial.

[ANTERIOR](#)

[PRÓXIMO](#)

# Adicionar uma exibição a um aplicativo ASP.NET Core MVC

04/02/2019 • 15 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

Nesta seção, você modifica a classe `HelloWorldController` para que ela use os arquivos de exibição do `Razor` para encapsular corretamente o processo de geração de respostas HTML para um cliente.

Crie um arquivo de modelo de exibição usando o `Razor`. Os modelos de exibição baseados no `Razor` têm uma extensão de arquivo `.cshtml`. Eles fornecem uma maneira elegante de criar a saída HTML com o `C#`.

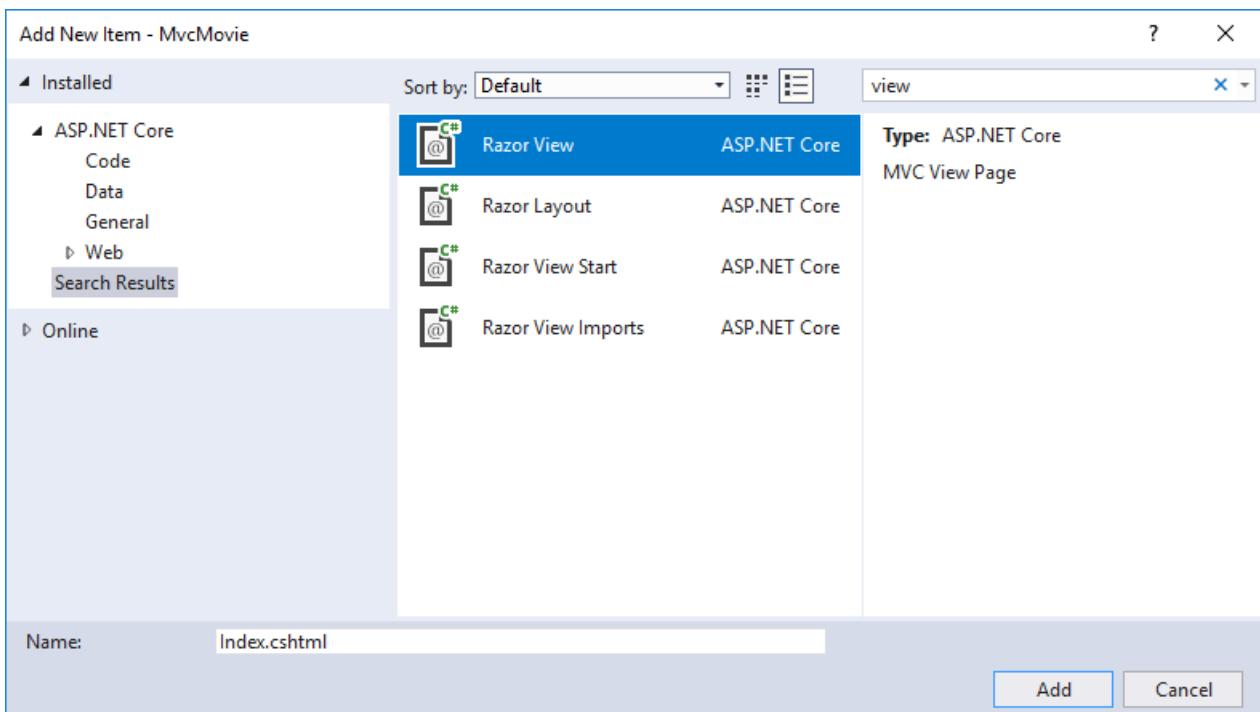
Atualmente, o método `Index` retorna uma cadeia de caracteres com uma mensagem que é embutida em código na classe do controlador. Na classe `HelloWorldController`, substitua o método `Index` pelo seguinte código:

```
public IActionResult Index()
{
    return View();
}
```

O código anterior chama o método `View` do controlador. Ele usa um modelo de exibição para gerar uma resposta HTML. Métodos do controlador (também conhecidos como *métodos de ação*), como o método `Index` acima, geralmente retornam um `IActionResult` (ou uma classe derivada de `ActionResult`), não um tipo como `string`.

## Adicionar uma exibição

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- Clique com o botão direito do mouse na pasta *Exibições* e, em seguida, **Adicionar > Nova Pasta** e nomeie a pasta *HelloWorld*.
- Clique com o botão direito do mouse na pasta *Views/HelloWorld* e, em seguida, clique em **Adicionar > Novo Item**.
- Na caixa de diálogo **Adicionar Novo Item – MvcMovie**
  - Na caixa de pesquisa no canto superior direito, insira *exibição*
  - Selecione **Exibição Razor**
  - Mantenha o valor da caixa **Nome**, *Index.cshtml*.
  - Selecione **Adicionar**



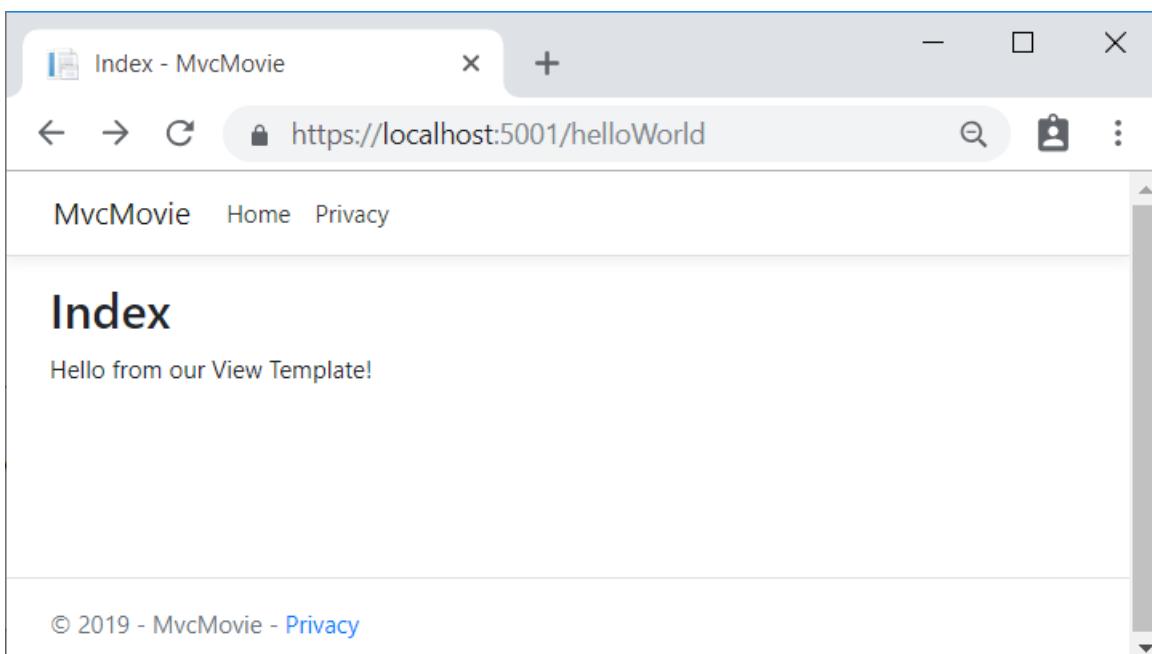
Substitua o conteúdo do arquivo de exibição `Views/HelloWorld/Index.cshtml` do Razor pelo seguinte:

```
@{
    ViewData["Title"] = "Index";
}

<h2>Index</h2>

<p>Hello from our View Template!</p>
```

Navegue para `https://localhost:xxxx>HelloWorld`. O método `Index` no `HelloWorldController` não fez muita coisa: ele executou a instrução `return View();`, que especificou que o método deve usar um arquivo de modelo de exibição para renderizar uma resposta para o navegador. Como você não especificou explicitamente o nome do arquivo do modelo de exibição, o MVC usou como padrão o arquivo de exibição `Index.cshtml` na pasta `/Views/HelloWorld`. A imagem abaixo mostra a cadeia de caracteres “Olá de nosso modelo de exibição!” embutida em código na exibição.



## Alterar exibições e páginas de layout

Selecione os links de menu (**MvcMovie**, **Página Inicial** e **Privacidade**). Cada página mostra o mesmo layout de menu. O layout de menu é implementado no arquivo `Views/Shared/_Layout.cshtml`. Abra o arquivo `Views/Shared/_Layout.cshtml`.

Os modelos de **layout** permitem especificar o layout de contêiner HTML do site em um lugar e, em seguida, aplicá-lo a várias páginas do site. Localize a linha `@RenderBody()`. `RenderBody` é um espaço reservado em que todas as páginas específicas à exibição criadas são mostradas, *encapsuladas* na página de layout. Por exemplo, se você selecionar o link **Privacidade**, a exibição `Views/Home/Privacy.cshtml` será renderizada dentro do método `RenderBody`.

## Alterar o título, o rodapé e o link de menu no arquivo de layout

- Nos elementos de título e de rodapé, altere `MvcMovie` para `Movie App`.
- Altere o elemento de âncora

```
<a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">MvcMovie</a> para  
<a class="navbar-brand" asp-controller="Movies" asp-action="Index">Movie App</a>.
```

A seguinte marcação mostra as alterações realçadas:

```
<!DOCTYPE html>  
<html>  
<head>  
    <meta charset="utf-8" />  
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
    <title>@ViewData["Title"] - Movie App</title>  
  
    <environment include="Development">  
        <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />  
    </environment>  
    <environment exclude="Development">  
        <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/twitter-  
bootstrap/4.1.3/css/bootstrap.min.css"  
              asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"  
              asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-test-  
value="absolute"  
              crossorigin="anonymous"  
              integrity="sha256-eSi1q2PG6J7g7ib17yAaWMcrr5GrtohYChqibrV7PBE="/>  
    </environment>  
    <link rel="stylesheet" href="~/css/site.css" />  
</head>  
<body>  
    <header>  
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-  
shadow mb-3">  
            <div class="container">  
                <a class="navbar-brand" asp-controller="Movies" asp-action="Index">Movie App</a>  
                <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-  
collapse" aria-controls="navbarSupportedContent"  
                      aria-expanded="false" aria-label="Toggle navigation">  
                    <span class="navbar-toggler-icon"></span>  
                </button>  
                <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">  
                    <ul class="navbar-nav flex-grow-1">  
                        <li class="nav-item">  
                            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-  
action="Index">Home</a>  
                        </li>  
                        <li class="nav-item">  
                            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-  
action="Privacy">Privacy</a>  
                        </li>
```

```

        </ul>
    </div>
</div>
</nav>
</header>
<div class="container">
    <partial name="_CookieConsentPartial" />
    <main role="main" class="pb-3">
        @RenderBody()
    </main>
</div>

<footer class="border-top footer text-muted">
    <div class="container">
        &copy; 2019 - Movie App - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
    </div>
</footer>

<environment include="Development">
    <script src="~/lib/jquery/dist/jquery.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.js"></script>
</environment>
<environment exclude="Development">
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"
        asp-fallback-src="~/lib/jquery/dist/jquery.min.js"
        asp-fallback-test="window.jQuery"
        crossorigin="anonymous"
        integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8=">
    </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/twitter-
bootstrap/4.1.3/js/bootstrap.bundle.min.js"
        asp-fallback-src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"
        asp-fallback-test="window.jQuery && window.jQuery.fn && window.jQuery.fn.modal"
        crossorigin="anonymous"
        integrity="sha256-E/V4cWE4qvAe05MOhjtGtqDzPndR01LBk8lJ/PR7CA4=>
    </script>
</environment>
<script src("~/js/site.js" asp-append-version="true"></script>

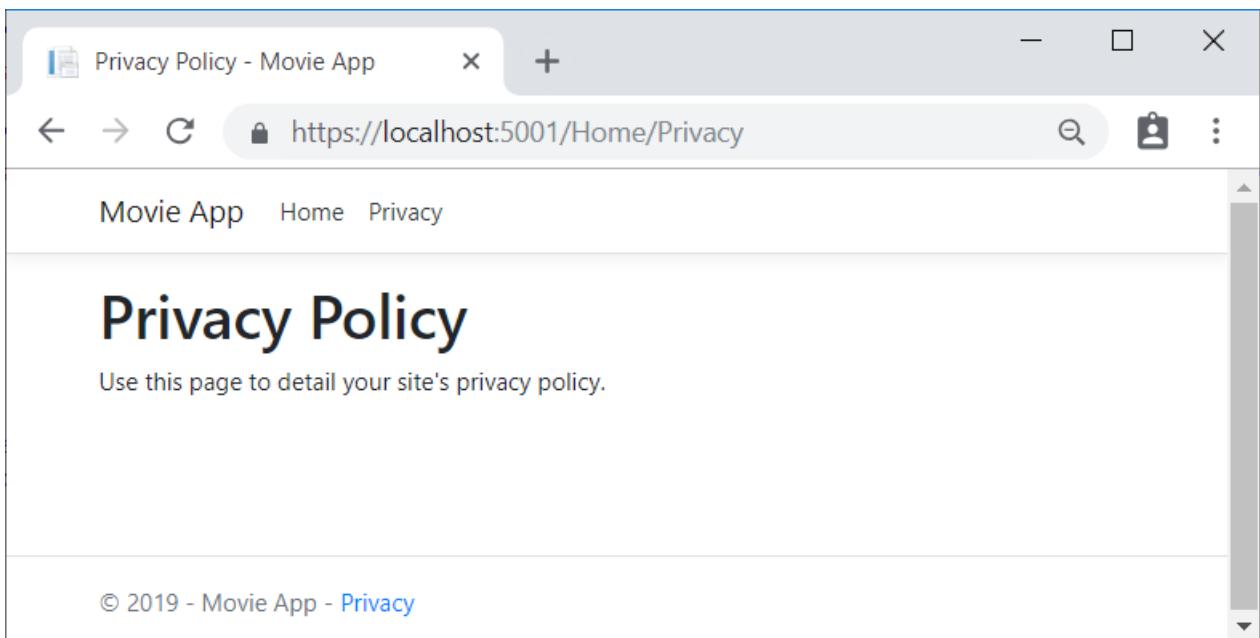
    @RenderSection("Scripts", required: false)
</body>
</html>

```

Na marcação anterior, o `asp-area` atributo do Auxiliar de Marca de Âncora foi omitido porque este aplicativo não está usando Áreas.

**Observação:** O controlador `Movies` não foi implementado. Neste ponto, o link `Movie App` não está funcionando.

Salve suas alterações e selecione o link **Privacidade**. Observe como o título na guia do navegador agora exibe **Política de Privacidade – Aplicativo de filme**, em vez de **Política de Privacidade – Filme MVC**:



Selecione o link **Página Inicial** e observe que o texto do título e de âncora também exibem **Aplicativo de Filme**. Conseguimos fazer a alteração uma vez no modelo de layout e fazer com que todas as páginas no site refletissem o novo texto do link e o novo título.

Examine o arquivo *Views/\_ViewStart.cshtml*:

```
@{  
    Layout = "_Layout";  
}
```

O arquivo *Views/\_ViewStart.cshtml* mostra o arquivo *Views/Shared/\_Layout.cshtml* em cada exibição. A propriedade `Layout` pode ser usada para definir outra exibição de layout ou defina-a como `null` para que nenhum arquivo de layout seja usado.

Altere o título e o elemento `<h2>` do arquivo de exibição *Views/HelloWorld/Index.cshtml*:

```
@{  
    ViewData["Title"] = "Movie List";  
}  
  
<h2>My Movie List</h2>  
  
<p>Hello from our View Template!</p>
```

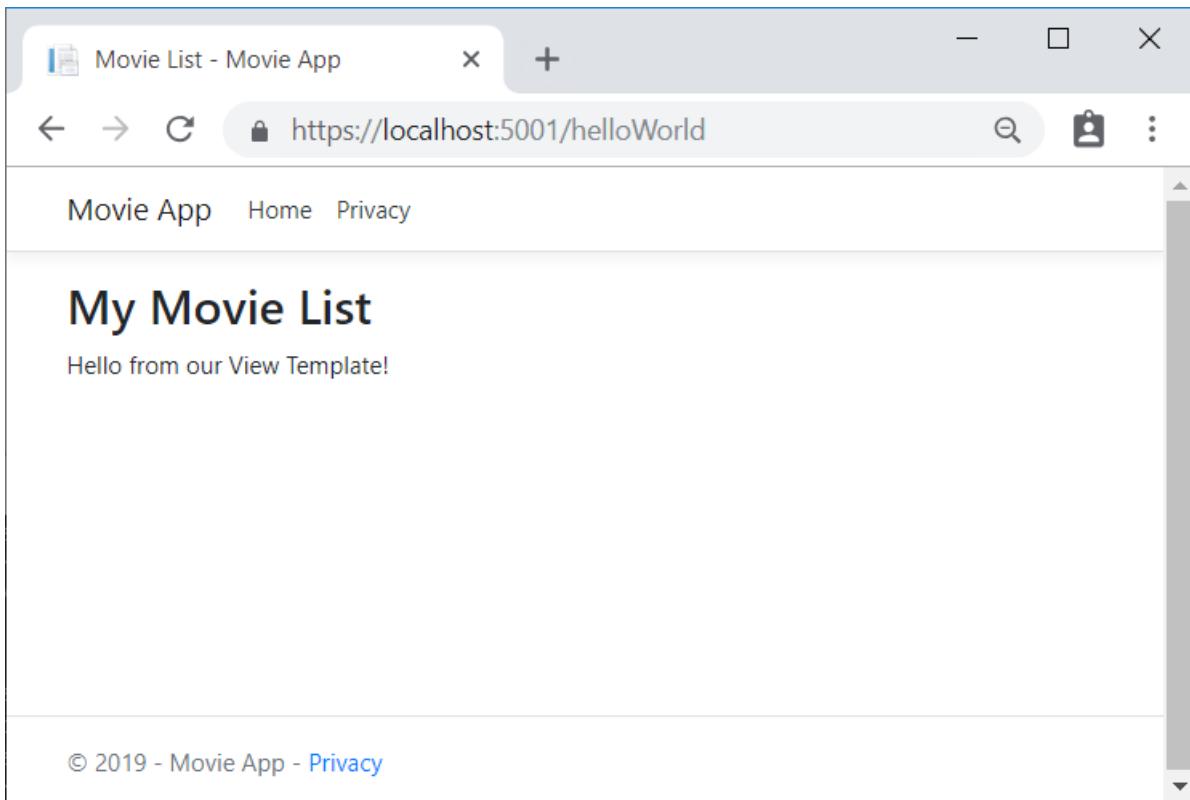
O título e o elemento `<h2>` são ligeiramente diferentes para que possa ver qual parte do código altera a exibição.

`ViewData["Title"] = "Movie List";` no código acima define a propriedade `Title` do dicionário `ViewData` como "Lista de Filmes". A propriedade `Title` é usada no elemento HTML `<title>` na página de layout:

```
<title>@ViewData["Title"] - Movie App</title>
```

Salve as alterações e navegue para `https://localhost:xxxx>HelloWorld`. Observe que o título do navegador, o cabeçalho primário e os títulos secundários foram alterados. (Se as alterações não forem exibidas no navegador, talvez o conteúdo armazenado em cache esteja sendo exibido. Pressione `Ctrl+F5` no navegador para forçar a resposta do servidor a ser carregada.) O título do navegador é criado com `viewData["Title"]` que definimos no modelo de exibição *Index.cshtml* e o "- Aplicativo de Filme" adicional adicionado no arquivo de layout.

Observe também como o conteúdo no modelo de exibição *Index.cshtml* foi mesclado com o modelo de exibição *Views/Shared/\_Layout.cshtml* e uma única resposta HTML foi enviada para o navegador. Os modelos de layout facilitam realmente a realização de alterações que se aplicam a todas as páginas do aplicativo. Para saber mais, consulte [Layout](#).



Apesar disso, nossos poucos “dados” (nesse caso, a mensagem “Olá de nosso modelo de exibição!”) são embutidos em código. O aplicativo MVC tem um “V” (exibição) e você tem um “C” (controlador), mas ainda nenhum “M” (modelo).

## Passando dados do controlador para a exibição

As ações do controlador são invocadas em resposta a uma solicitação de URL de entrada. Uma classe de controlador é o local em que o código é escrito e que manipula as solicitações recebidas do navegador. O controlador recupera dados de uma fonte de dados e decide qual tipo de resposta será enviada novamente para o navegador. Modelos de exibição podem ser usados em um controlador para gerar e formatar uma resposta HTML para o navegador.

Os controladores são responsáveis por fornecer os dados necessários para que um modelo de exibição renderize uma resposta. Uma melhor prática: modelos de exibição **não** devem executar a lógica de negócios nem interagir diretamente com um banco de dados. Em vez disso, um modelo de exibição deve funcionar somente com os dados fornecidos pelo controlador. Manter essa “separação de preocupações” ajuda a manter o código limpo, testável e com capacidade de manutenção.

Atualmente, o método `Welcome` na classe `HelloWorldController` usa um parâmetro `name` e um `ID` e, em seguida, gera os valores diretamente no navegador. Em vez de fazer com que o controlador renderize a resposta como uma cadeia de caracteres, altere o controlador para que ele use um modelo de exibição. O modelo de exibição gera uma resposta dinâmica, o que significa que é necessário passar bits de dados apropriados do controlador para a exibição para gerar a resposta. Faça isso fazendo com que o controlador coloque os dados dinâmicos (parâmetros) que o modelo de exibição precisa em um dicionário `ViewData` que pode ser acessado em seguida pelo modelo de exibição.

Em `HelloWorldController.cs`, altere o método `Welcome` para adicionar um valor `Message` e `NumTimes` ao dicionário `ViewData`. O dicionário `viewData` é um objeto dinâmico, o que significa que qualquer tipo pode ser usado. O

objeto `ViewData` não tem nenhuma propriedade definida até que você insira algo nele. O sistema de [model binding](#) MVC mapeia automaticamente os parâmetros nomeados (`name` e `numTimes`) da cadeia de consulta na barra de endereços para os parâmetros no método. O arquivo `HelloWorldController.cs` completo tem esta aparência:

```
using Microsoft.AspNetCore.Mvc;
using System.Text.Encodings.Web;

namespace MvcMovie.Controllers
{
    public class HelloWorldController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Welcome(string name, int numTimes = 1)
        {
            ViewData["Message"] = "Hello " + name;
            ViewData["NumTimes"] = numTimes;

            return View();
        }
    }
}
```

O objeto de dicionário `ViewData` contém dados que serão passados para a exibição.

Crie um modelo de exibição Boas-vindas chamado `Views/HelloWorld/Welcome.cshtml`.

Você criará um loop no modelo de exibição `Welcome.cshtml` que exibe “Olá” `NumTimes`. Substitua o conteúdo de `Views/HelloWorld/Welcome.cshtml` pelo seguinte:

```
@{
    ViewData["Title"] = "Welcome";
}

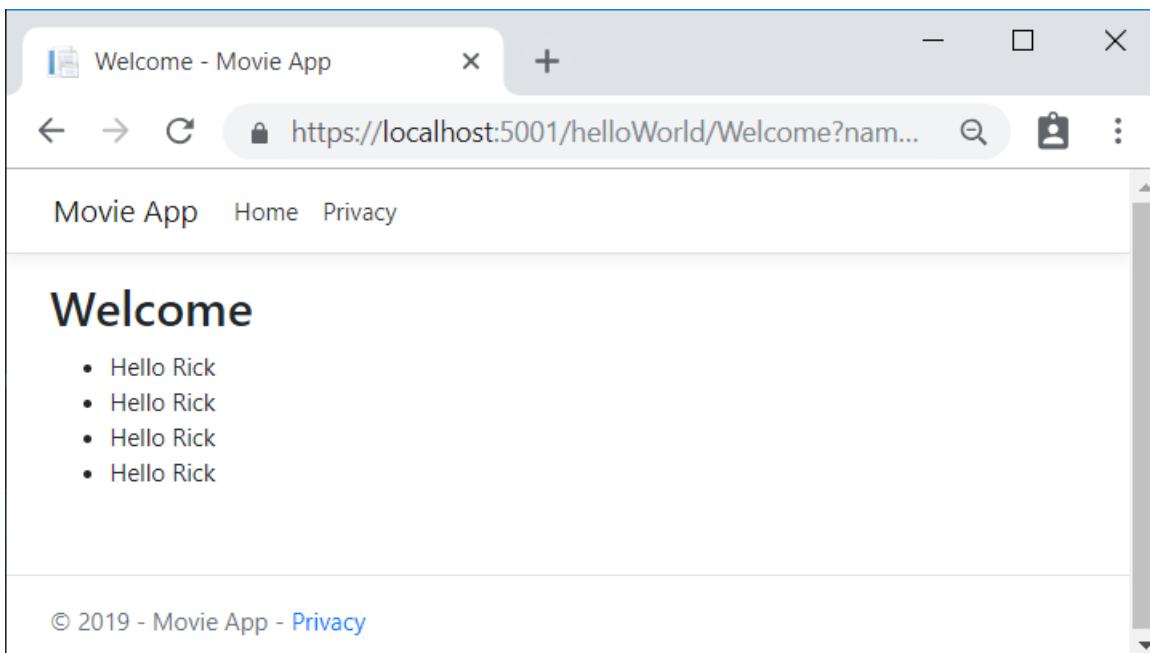
<h2>Welcome</h2>

<ul>
    @for (int i = 0; i < (int)ViewData["NumTimes"]; i++)
    {
        <li>@ViewData["Message"]</li>
    }
</ul>
```

Salve as alterações e navegue para a seguinte URL:

```
https://localhost:xxxx/HelloWorld/Welcome?name=Rick&numtimes=4
```

Os dados são obtidos da URL e passados para o controlador usando o [associador de modelo MVC](#). O controlador empacota os dados em um dicionário `ViewData` e passa esse objeto para a exibição. Em seguida, a exibição renderiza os dados como HTML para o navegador.



No exemplo acima, o dicionário `ViewData` foi usado para passar dados do controlador para uma exibição. Mais adiante no tutorial, um modelo de exibição será usado para passar dados de um controlador para uma exibição. A abordagem de modelo de exibição para passar dados é geralmente a preferida em relação à abordagem do dicionário `ViewData`. Para obter mais informações, confira [Quando usar ViewBag, ViewData ou TempData](#).

No próximo tutorial, será criado um banco de dados de filmes.

[ANTERIOR](#)

[PRÓXIMO](#)

# Adicione um modelo a um aplicativo ASP.NET Core MVC

21/01/2019 • 24 minutes to read • [Edit Online](#)

Por [Rick Anderson](#) e [Tom Dykstra](#)

Nesta seção, você adiciona classes para gerenciamento de filmes em um banco de dados. Essas classes serão a parte “Model” parte do aplicativo **MVC**.

Você usa essas classes com o [EF Core](#) (Entity Framework Core) para trabalhar com um banco de dados. O EF Core é uma estrutura ORM (mapeamento relacional de objetos) que simplifica o código de acesso a dados que você precisa escrever.

As classes de modelo que você cria são conhecidas como classes de dados POCO (de **objetos CLR básicos**) porque elas não têm nenhuma dependência no EF Core. Elas apenas definem as propriedades dos dados que serão armazenados no banco de dados.

Neste tutorial, você escreve as classes de modelo primeiro e o EF Core cria o banco de dados. Uma abordagem alternativa não abordada aqui é gerar classes de modelo de um banco de dados existente. Para obter informações sobre essa abordagem, consulte [ASP.NET Core – Banco de dados existente](#).

## Adicionar uma classe de modelo de dados

- [Visual Studio](#)
- [Visual Studio Code/Visual Studio para Mac](#)

Clique com o botão direito do mouse na pasta *Models* > **Adicionar** > **Classe**. Dê à classe o nome **Movie**.

Add the following properties to the `Movie` class:

```
using System;
using System.ComponentModel.DataAnnotations;

namespace MvcMovie.Models
{
    public class Movie
    {
        public int Id { get; set; }
        public string Title { get; set; }

        [DataType(DataType.Date)]
        public DateTime ReleaseDate { get; set; }
        public string Genre { get; set; }
        public decimal Price { get; set; }
    }
}
```

The `Movie` class contains:

- The `Id` field which is required by the database for the primary key.
- `[DataType(DataType.Date)]`: The `DataType` attribute specifies the type of the data (`Date`). With this attribute:
  - The user is not required to enter time information in the date field.

- Only the date is displayed, not time information.

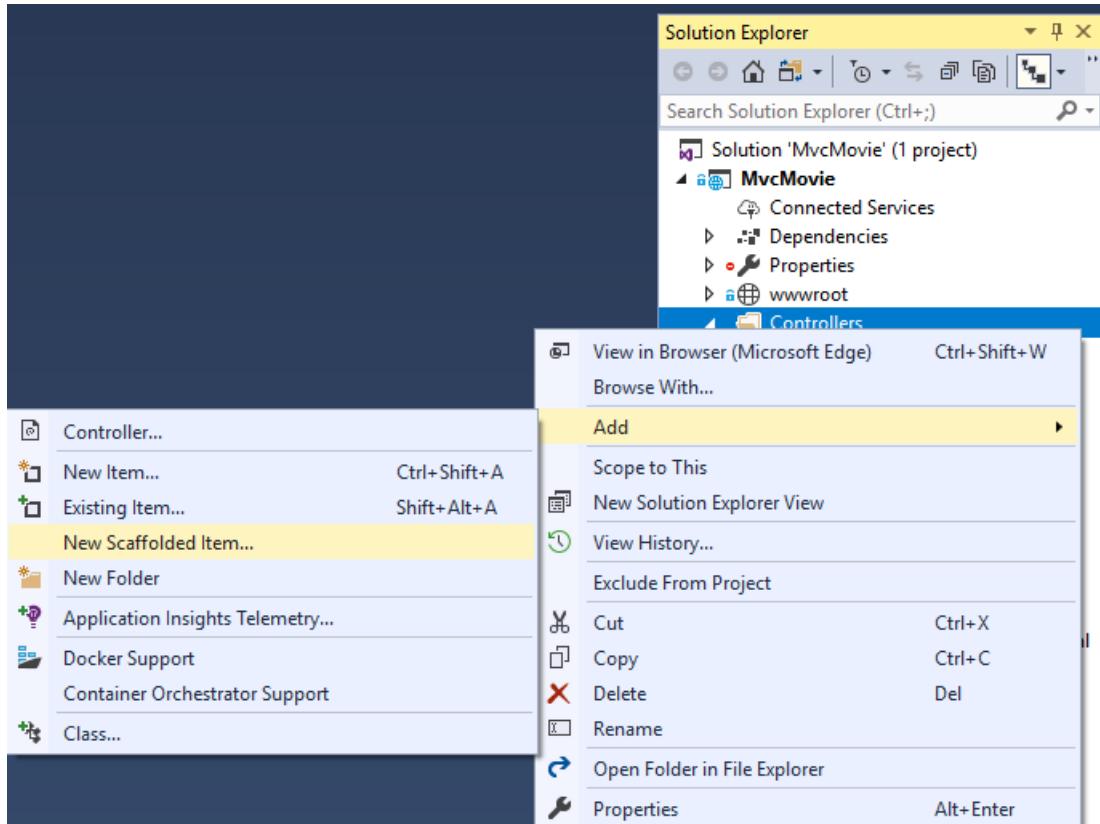
[DataAnnotations](#) are covered in a later tutorial.

## Fazer scaffold do modelo de filme

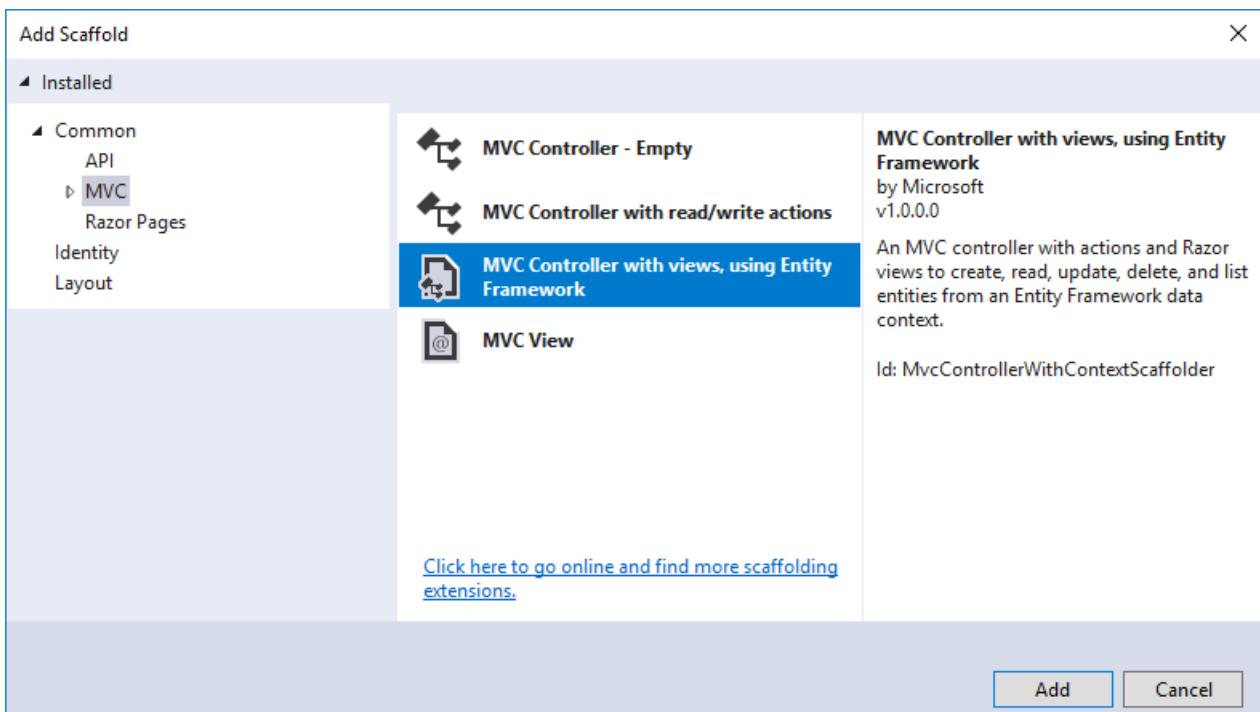
Nesta seção, é feito o scaffold do modelo de filme. Ou seja, a ferramenta de scaffolding gera páginas para operações de CRUD (Criar, Ler, Atualizar e Excluir) para o modelo do filme.

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)

No **Gerenciador de Soluções**, clique com o botão direito do mouse na pasta **Controladores** > **Adicionar** > **Novo Item com Scaffold**.

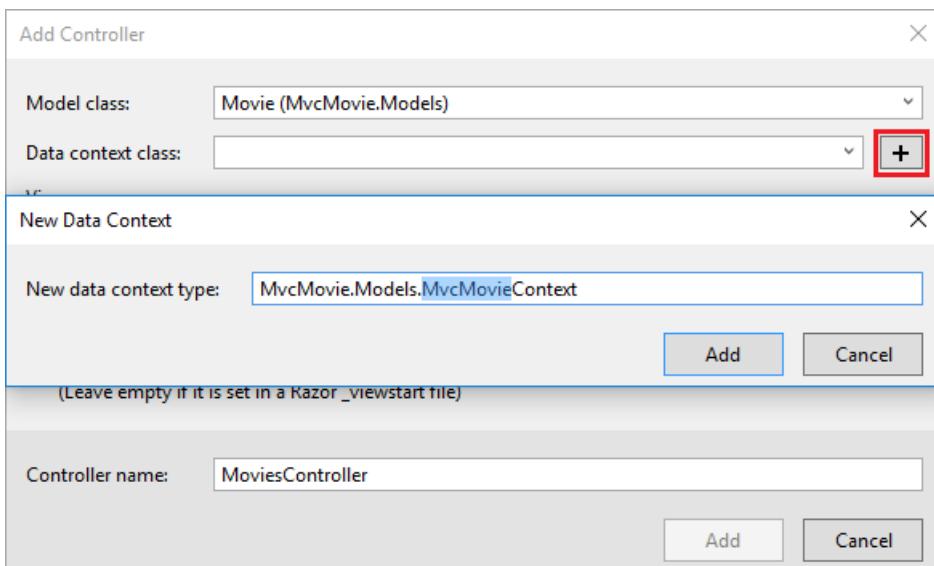


Na caixa de diálogo **Adicionar Scaffold**, selecione **Controlador MVC com exibições, usando o Entity Framework** > **Adicionar**.

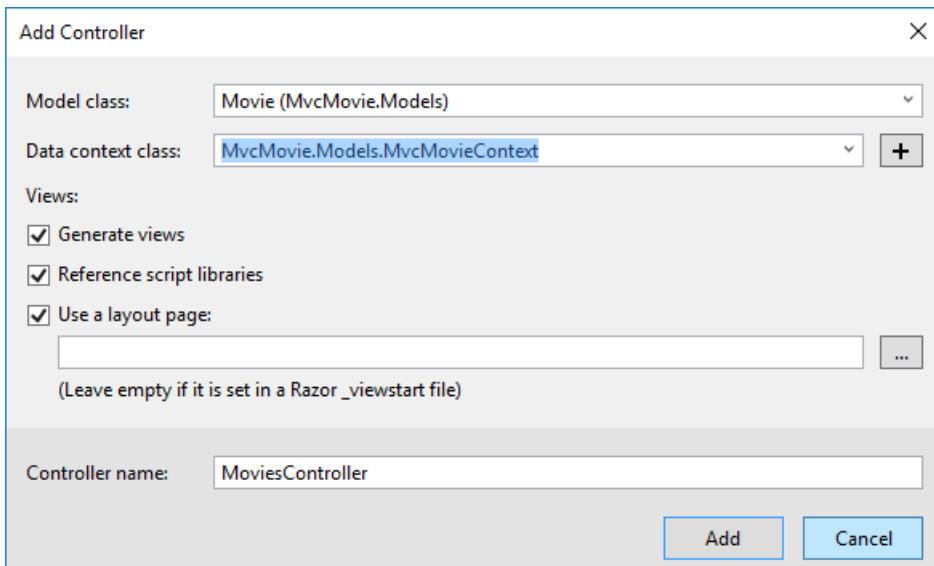


Preencha a caixa de diálogo **Adicionar Controlador**:

- **Classe de modelo:** Movie (*MvcMovie.Models*)
- **Classe de contexto de dados:** selecione o ícone + e adicione o **MvcMovie.Models.MvcMovieContext** padrão



- **Exibições:** mantenha o padrão de cada opção marcado
- **Nome do controlador:** mantenha o *MoviesController* padrão
- Selecione **Adicionar**



O Visual Studio cria:

- Uma [classe de contexto de banco de dados](#) do Entity Framework Core (*Data/MvcMovieContext.cs*)
- Um controlador de filmes (*Controllers/MoviesController.cs*)
- Arquivos de exibição do Razor para as páginas Criar, Excluir, Detalhes, Editar e Índice (*Views/Movies/\*.cshtml*)

A criação automática do contexto de banco de dados e das exibições e métodos de ação **CRUD** (criar, ler, atualizar e excluir) é conhecida como *scaffolding*.

Se você executar o aplicativo e clicar no link **Filme do MVC**, receberá um erro semelhante ao seguinte:

```
An unhandled exception occurred while processing the request.  
  
SqlException: Cannot open database "MvcMovieContext-<GUID removed>" requested by the login. The login failed.  
Login failed for user 'Rick'.  
  
System.Data.SqlClient.SqlInternalConnectionTds..ctor(DbConnectionPoolIdentity identity, SqlConnectionString
```

Você precisa criar o banco de dados e usará o recurso [Migrações](#) do EF Core para fazer isso. As Migrações permitem criar um banco de dados que corresponde ao seu modelo de dados e atualizar o esquema de banco de dados quando o modelo de dados é alterado.

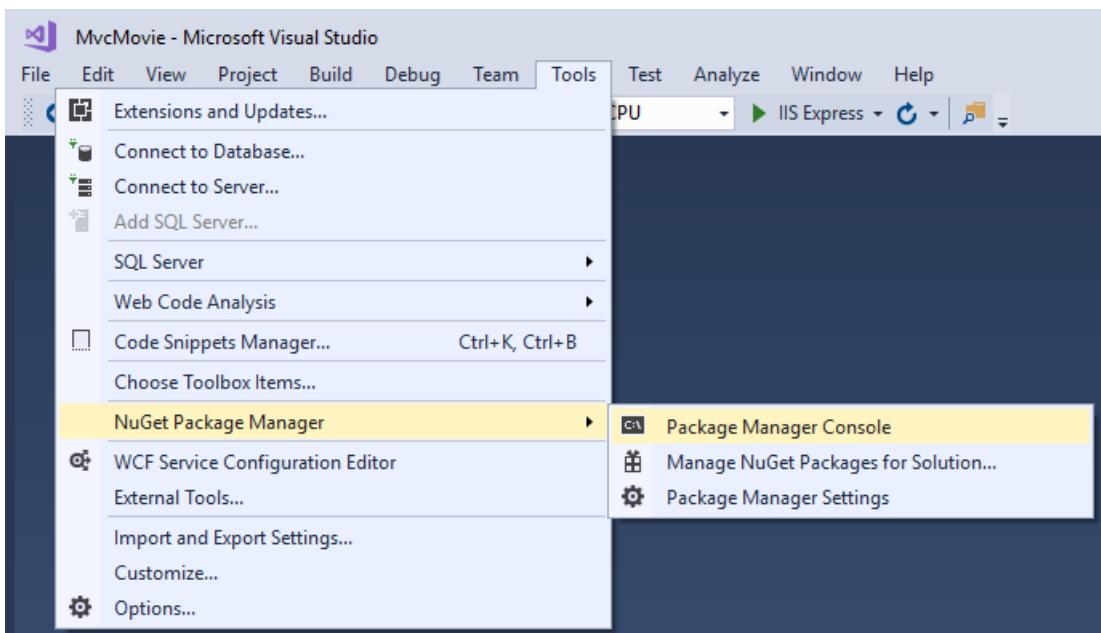
## Migração inicial

- [Visual Studio](#)
- [Visual Studio Code/Visual Studio para Mac](#)

Nesta seção, o PMC (Console de Gerenciador de Pacotes) é usado para:

- Adicione uma migração inicial.
- Atualize o banco de dados com a migração inicial.

No menu **Ferramentas**, selecione **Gerenciador de pacotes NuGet > Console do Gerenciador de pacotes**.



No PMC, insira os seguintes comandos:

```
Add-Migration Initial  
Update-Database
```

O comando `Add-Migration` gera código para criar o esquema de banco de dados inicial.

Os comandos anteriores geram o seguinte aviso: "Nenhum tipo foi especificado para a coluna decimal "Preço" no tipo de entidade "Filme". Isso fará com que valores sejam truncados silenciosamente se não couberem na precisão e na escala padrão. Especifique explicitamente o tipo de coluna do SQL Server que pode acomodar todos os valores usando 'HasColumnType()'."

Você pode ignorar esse aviso, ele será corrigido em um tutorial posterior.

O esquema é baseado no modelo especificado no `DbContext` (no arquivo *Models/MvcMovieContext.cs*). O argumento `InitialCreate` é usado para nomear as migrações. Qualquer nome pode ser usado, mas, por convenção, um nome que descreve a migração é selecionado.

O comando `ef database update` executa o método `Up` no arquivo *Migrations/<time-stamp>\_InitialCreate.cs*. O método `Up` cria o banco de dados.

- [Visual Studio](#)
- [Visual Studio Code/Visual Studio para Mac](#)

## Examinar o contexto registrado com a injeção de dependência

O ASP.NET Core é construído com a [injeção de dependência](#). Serviços (como o contexto de BD do EF Core) são registrados com injeção de dependência durante a inicialização do aplicativo. Os componentes que exigem esses serviços (como as Páginas do Razor) recebem esses serviços por meio de parâmetros do construtor. O código de construtor que obtém uma instância de contexto do BD será mostrado mais adiante no tutorial.

A ferramenta de scaffolding criou automaticamente um contexto de BD e o registrou no contêiner da injeção de dependência.

Examine o método `Startup.ConfigureServices`. A linha destacada foi adicionada pelo scaffolder:

```

public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        // This lambda determines whether user consent for non-essential cookies
        // is needed for a given request.
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);

    services.AddDbContext<MvcMovieContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("MvcMovieContext")));
}

```

O `MvcMovieContext` coordena a funcionalidade do EF Core (Criar, Ler, Atualizar, Excluir etc.) para o modelo `Movie`. O contexto de dados (`MvcMovieContext`) deriva de `Microsoft.EntityFrameworkCore.DbContext`. O contexto de dados especifica quais entidades são incluídas no modelo de dados:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;

namespace MvcMovie.Models
{
    public class MvcMovieContext : DbContext
    {
        public MvcMovieContext (DbContextOptions<MvcMovieContext> options)
            : base(options)
        {
        }

        public DbSet<MvcMovie.Models.Movie> Movie { get; set; }
    }
}

```

O código anterior cria uma propriedade `DbSet<Movie>` para o conjunto de entidades. Na terminologia do Entity Framework, um conjunto de entidades normalmente corresponde a uma tabela de banco de dados. Uma entidade corresponde a uma linha da tabela.

O nome da cadeia de conexão é passado para o contexto com a chamada de um método em um objeto `DbContextOptions`. Para o desenvolvimento local, o [sistema de configuração do ASP.NET Core](#) lê a cadeia de conexão do arquivo `appsettings.json`.

O esquema é baseado no modelo especificado no `MvcMovieContext` (no arquivo `Data/MvcMovieContext.cs`). O argumento `Initial` é usado para nomear as migrações. Qualquer nome pode ser usado, mas, por convenção, um nome que descreve a migração é usado. Consulte [Introdução às migrações](#) para obter mais informações.

O comando `Update-Database` executa o método `Up` no arquivo `Migrations/{time-stamp}_InitialCreate.cs`, que cria o banco de dados.

## Testar o aplicativo

- Executar o aplicativo e acrescentar `/Movies` à URL no navegador (`http://localhost:port/movies`).

Se você receber uma exceção de banco de dados semelhante ao seguinte:

```
SqlException: Cannot open database "MvcMovieContext-GUID" requested by the login. The login failed.  
Login failed for user 'User-name'.
```

Você perdeu a etapa de migrações.

- Teste o link **Criar**.

#### NOTE

Talvez você não consiga inserir casas decimais ou vírgulas no campo `Price`. Para dar suporte à [validação do jQuery](#) para localidades com idiomas diferentes do inglês que usam uma vírgula (",") para um ponto decimal e formatos de data diferentes do inglês dos EUA, o aplicativo precisa ser globalizado. Para obter instruções sobre a globalização, consulte [esse problema no GitHub](#).

- Teste os links **Editar**, **Detalhes** e **Excluir**.

Examine a classe `Startup`:

```
public void ConfigureServices(IServiceCollection services)  
{  
    services.Configure<CookiePolicyOptions>(options =>  
    {  
        // This lambda determines whether user consent for non-essential cookies  
        // is needed for a given request.  
        options.CheckConsentNeeded = context => true;  
        options.MinimumSameSitePolicy = SameSiteMode.None;  
    });  
  
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);  
  
    services.AddDbContext<MvcMovieContext>(options =>  
        options.UseSqlServer(Configuration.GetConnectionString("MvcMovieContext")));  
}
```

o código realçado acima mostra o contexto de banco de dados do filme que está sendo adicionado ao contêiner [Injeção de Dependência](#):

- `services.AddDbContext<MvcMovieContext>(options =>` especifica o banco de dados a ser usado e a cadeia de conexão.
- `=>` é um [operador lambda](#)

Abra o arquivo `Controllers/MoviesController.cs` e examine o construtor:

```
public class MoviesController : Controller  
{  
    private readonly MvcMovieContext _context;  
  
    public MoviesController(MvcMovieContext context)  
    {  
        _context = context;  
    }
```

O construtor usa a [Injeção de Dependência](#) para injetar o contexto de banco de dados (`MvcMovieContext`) no controlador. O contexto de banco de dados é usado em cada um dos métodos [CRUD](#) no controlador.

## Modelos fortemente tipados e a palavra-chave `@model`

Anteriormente neste tutorial, você viu como um controlador pode passar dados ou objetos para uma exibição usando o dicionário `ViewData`. O dicionário `ViewData` é um objeto dinâmico que fornece uma maneira conveniente de associação tardia para passar informações para uma exibição.

O MVC também fornece a capacidade de passar objetos de modelo fortemente tipados para uma exibição. Essa abordagem fortemente tipada habilita uma melhor verificação do tempo de compilação do código. O mecanismo de scaffolding usou essa abordagem (ou seja, passando um modelo fortemente tipado) com a classe `MoviesController` e as exibições quando ele criou os métodos e as exibições.

Examine o método `Details` gerado no arquivo `Controllers/MoviesController.cs`:

```
// GET: Movies/Details/5
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var movie = await _context.Movie
        .FirstOrDefaultAsync(m => m.Id == id);
    if (movie == null)
    {
        return NotFound();
    }

    return View(movie);
}
```

O parâmetro `id` geralmente é passado como dados de rota. Por exemplo,

`https://localhost:5001/movies/details/1` define:

- O controlador para o controlador `movies` (o primeiro segmento de URL).
- A ação para `details` (o segundo segmento de URL).
- A ID como 1 (o último segmento de URL).

Você também pode passar a `id` com uma cadeia de consulta da seguinte maneira:

`https://localhost:5001/movies/details?id=1`

O parâmetro `id` é definido como um [tipo que permite valor nulo](#) (`int?`), caso um valor de ID não seja fornecido.

Um [expressão lambda](#) é passada para `FirstOrDefaultAsync` para selecionar as entidades de filmes que correspondem ao valor da cadeia de consulta ou de dados da rota.

```
var movie = await _context.Movie
    .FirstOrDefaultAsync(m => m.Id == id);
```

Se for encontrado um filme, uma instância do modelo `Movie` será passada para a exibição `Details`:

```
return View(movie);
```

Examine o conteúdo do arquivo `Views/Movies/Details.cshtml`:

```

@model MvcMovie.Models.Movie

@{
    ViewData["Title"] = "Details";
}

<h1>Details</h1>

<div>
    <h4>Movie</h4>
    <hr />
    <dl class="row">
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Title)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Title)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.ReleaseDate)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.ReleaseDate)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Genre)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Genre)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Price)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Price)
        </dd>
    </dl>
</div>
<div>
    <a href="#">Edit</a> | 
    <a href="#">Back to List</a>
</div>

```

Incluindo uma instrução `@model` na parte superior do arquivo de exibição, você pode especificar o tipo de objeto esperado pela exibição. Quando você criou o controlador de filmes, a seguinte instrução `@model` foi automaticamente incluída na parte superior do arquivo *Details.cshtml*:

```

@model MvcMovie.Models.Movie

```

Esta diretiva `@model` permite acessar o filme que o controlador passou para a exibição usando um objeto `Model` fortemente tipado. Por exemplo, na exibição *Details.cshtml*, o código passa cada campo de filme para os Auxiliares de HTML `DisplayNameFor` e `DisplayFor` com o objeto `Model` fortemente tipado. Os métodos `Create` e `Edit` e as exibições também passam um objeto de modelo `Movie`.

Examine a exibição *Index.cshtml* e o método `Index` no controlador `Movies`. Observe como o código cria um objeto `List` quando ele chama o método `View`. O código passa esta lista `Movies` do método de ação `Index` para a exibição:

```
// GET: Movies
public async Task<IActionResult> Index()
{
    return View(await _context.Movie.ToListAsync());
}
```

Quando você criou o controlador de filmes, o scaffolding incluiu automaticamente a seguinte instrução `@model` na parte superior do arquivo *Index.cshtml*:

```
@model IEnumerable<MvcMovie.Models.Movie>
```

A diretiva `@model` permite acessar a lista de filmes que o controlador passou para a exibição usando um objeto `Model` fortemente tipado. Por exemplo, na exibição *Index.cshtml*, o código executa um loop pelos filmes com uma instrução `foreach` no objeto `Model` fortemente tipado:

```

@model IEnumerable<MvcMovie.Models.Movie>

 @{
     ViewData["Title"] = "Index";
 }

<h1>Index</h1>

<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Title)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.ReleaseDate)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Genre)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Price)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Title)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.ReleaseDate)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Genre)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Price)
        </td>
        <td>
            <a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |
            <a asp-action="Details" asp-route-id="@item.Id">Details</a> |
            <a asp-action="Delete" asp-route-id="@item.Id">Delete</a>
        </td>
    </tr>
}
    </tbody>
</table>

```

Como o objeto `Model` é fortemente tipado (como um objeto `IEnumerable<Movie>`), cada item no loop é tipado como `Movie`. Entre outros benefícios, isso significa que você obtém a verificação do tempo de compilação do código:

## Recursos adicionais

- [Auxiliares de marcação](#)
- [Globalização e localização](#)

ANTERIOR – ADICIONANDO UMA  
EXIBIÇÃO

PRÓXIMO – TRABALHANDO COM O  
SQL

# Trabalhar com o SQL no ASP.NET Core

10/01/2019 • 6 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

O objeto `MvcMovieContext` cuida da tarefa de se conectar ao banco de dados e mapear objetos `Movie` para registros do banco de dados. O contexto de banco de dados é registrado com o contêiner [Injeção de Dependência](#) no método `ConfigureServices` no arquivo `Startup.cs`:

- [Visual Studio](#)
- [Visual Studio Code/Visual Studio para Mac](#)

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        // This lambda determines whether user consent for non-essential cookies
        // is needed for a given request.
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);

    services.AddDbContext<MvcMovieContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("MvcMovieContext")));
}
```

O sistema de [Configuração](#) do ASP.NET Core lê a `ConnectionString`. Para o desenvolvimento local, ele obtém a cadeia de conexão do arquivo `appsettings.json`:

```
"ConnectionStrings": {
    "MvcMovieContext": "Server=(localdb)\\mssqllocaldb;Database=MvcMovieContext-
2;Trusted_Connection=True;MultipleActiveResultSets=true"
}
```

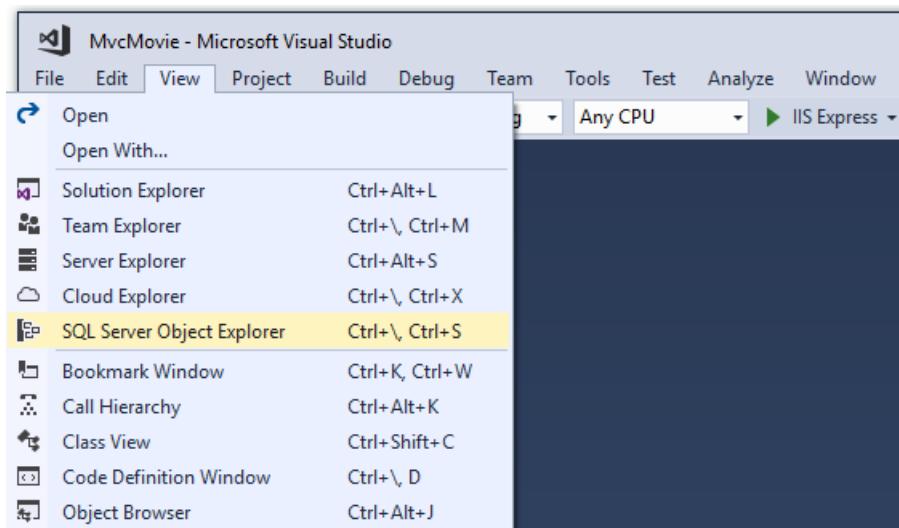
Quando você implanta o aplicativo em um servidor de teste ou de produção, você pode usar uma variável de ambiente ou outra abordagem para definir a cadeia de conexão como um SQL Server real. Consulte [Configuração](#) para obter mais informações.

- [Visual Studio](#)
- [Visual Studio Code/Visual Studio para Mac](#)

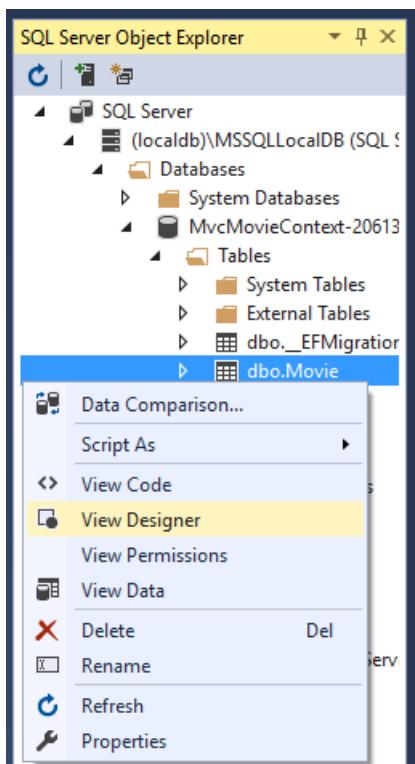
## SQL Server Express LocalDB

O LocalDB é uma versão leve do Mecanismo de Banco de Dados do SQL Server Express, que é direcionado para o desenvolvimento de programas. O LocalDB é iniciado sob demanda e executado no modo de usuário e, portanto, não há nenhuma configuração complexa. Por padrão, o banco de dados LocalDB cria arquivos `.mdf` no diretório `C:/Users/{user}`.

- No menu **Exibir**, abra **SSOX** (Pesquisador de Objetos do SQL Server).



- Clique com o botão direito do mouse na tabela `Movie` > **Designer de Exibição**



The screenshot shows the SSMS interface with the 'dbo.Movie [Design]' window open. The table structure is displayed in a grid with columns for Name, Data Type, Allow Nulls, and other properties. The 'Keys' section on the right shows a primary key constraint named 'PK\_Movie'. Below it are sections for Check Constraints, Indexes, Foreign Keys, and Triggers. The 'T-SQL' tab in the bottom navigation bar is selected, displaying the generated CREATE TABLE SQL code. The status bar at the bottom shows connection details.

Observe o ícone de chave ao lado de `ID`. Por padrão, o EF tornará uma propriedade chamada `Id` a chave primária.

- Clique com o botão direito do mouse na tabela `Movie` > **Dados de Exibição**

The screenshot shows the SQL Server Object Explorer with the 'Movie' table selected under the 'Tables' node of the 'MvcMovieContext-20613' database. A context menu is open over the table, with the 'View Data' option highlighted. Other options like 'Data Comparison...', 'Script As', 'View Code', 'View Designer', 'View Permissions', 'Delete', 'Rename', 'Refresh', and 'Properties' are also visible.

The screenshot shows the Microsoft Visual Studio interface. The title bar reads "MvcMovie - Microsoft V...". The menu bar includes File, Edit, View, Project, Build, Debug, Team, SQL, Tools, Architecture, Test, Analyze, Window, and Help. A user profile "rick" is shown in the top right. The main area displays the "SQL Server Object Explorer" and a "Solution Explorer" window. In the center, there is a grid view titled "dbo.Movie [Data]" showing the following data:

	ID	Genre	Price	ReleaseDate	Title
▶	1	Comedy	1.99	11/18/2015 12:00:00 AM	When Harry Met...
	2	Comedy	2.99	1/11/2016 12:00:00 AM	Ghost Busters IV
	3	Comedy	3.99	12/11/2015 12:00:00 AM	Ghost Busters 7
*	NULL	NULL	NULL	NULL	NULL

Below the grid, connection details are shown: Connection Re... ((localdb)\MSSQLLocalDB) REDMOND\riande aspnet5-MvcMovie-8f261... . The status bar at the bottom indicates 3 Rows | Cell is Read Only | Ln 1 | Col 1.

## Propagar o banco de dados

Crie uma nova classe chamada `SeedData` na pasta `Models`. Substitua o código gerado pelo seguinte:

```

using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using System;
using System.Linq;

namespace MvcMovie.Models
{
    public static class SeedData
    {
        public static void Initialize(IServiceProvider serviceProvider)
        {
            using (var context = new MvcMovieContext(
                serviceProvider.GetRequiredService<
                    DbContextOptions<MvcMovieContext>>()))
            {
                // Look for any movies.
                if (context.Movie.Any())
                {
                    return; // DB has been seeded
                }

                context.Movie.AddRange(
                    new Movie
                    {
                        Title = "When Harry Met Sally",
                        ReleaseDate = DateTime.Parse("1989-2-12"),
                        Genre = "Romantic Comedy",
                        Price = 7.99M
                    },
                    new Movie
                    {
                        Title = "Ghostbusters ",
                        ReleaseDate = DateTime.Parse("1984-3-13"),
                        Genre = "Comedy",
                        Price = 8.99M
                    },
                    new Movie
                    {
                        Title = "Ghostbusters 2",
                        ReleaseDate = DateTime.Parse("1986-2-23"),
                        Genre = "Comedy",
                        Price = 9.99M
                    },
                    new Movie
                    {
                        Title = "Rio Bravo",
                        ReleaseDate = DateTime.Parse("1959-4-15"),
                        Genre = "Western",
                        Price = 3.99M
                    }
                );
                context.SaveChanges();
            }
        }
    }
}

```

Se houver um filme no BD, o inicializador de semeadura será retornado e nenhum filme será adicionado.

```
if (context.Movie.Any())
{
    return; // DB has been seeded.
}
```

## Adicionar o inicializador de semeadura

Substitua o conteúdo de *Program.cs* pelo código a seguir:

```
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using System;
using Microsoft.EntityFrameworkCore;
using MvcMovie.Models;
using MvcMovie;

namespace MvcMovie
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var host = CreateWebHostBuilder(args).Build();

            using (var scope = host.Services.CreateScope())
            {
                var services = scope.ServiceProvider;

                try
                {
                    var context = services.GetRequiredService<MvcMovieContext>();
                    context.Database.Migrate();
                    SeedData.Initialize(services);
                }
                catch (Exception ex)
                {
                    var logger = services.GetRequiredService<ILogger<Program>>();
                    logger.LogError(ex, "An error occurred seeding the DB.");
                }
            }

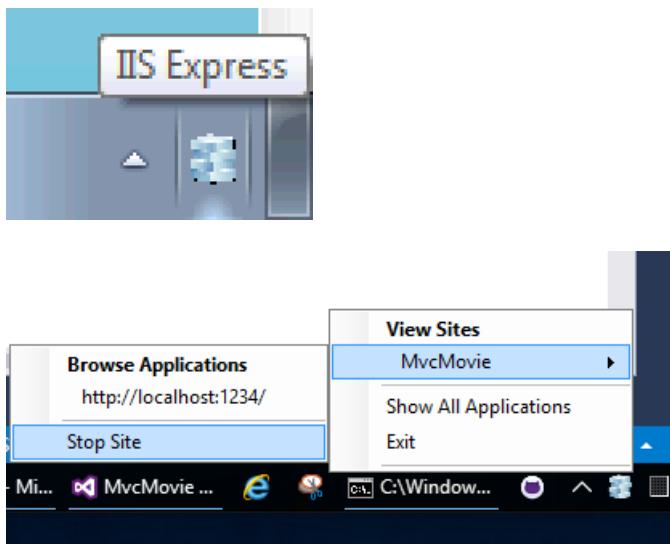
            host.Run();
        }

        public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>();
    }
}
```

## Testar o aplicativo

- [Visual Studio](#)
- [Visual Studio Code/Visual Studio para Mac](#)
- Exclua todos os registros no BD. Faça isso com os links Excluir no navegador ou no SSOX.
- Force o aplicativo a ser inicializado (chame os métodos na classe `Startup`) para que o método de semeadura seja executado. Para forçar a inicialização, o IIS Express deve ser interrompido e reiniciado. Faça isso com uma das seguintes abordagens:

- Clique com botão direito do mouse no ícone de bandeja do sistema do IIS Express na área de notificação e toque em **Sair** ou **Parar Site**



- Se você estiver executando o VS no modo sem depuração, pressione F5 para executar no modo de depuração
- Se você estiver executando o VS no modo de depuração, pare o depurador e pressione F5

O aplicativo mostra os dados propagados.

Title	ReleaseDate	Genre	Price	
When Harry Met Sally	2/12/1989	Romantic Comedy	7.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Ghostbusters	3/13/1984	Comedy	8.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Ghostbusters 2	2/23/1986	Comedy	9.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Rio Bravo	4/15/1959	Western	3.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

© 2019 - Movie App - [Privacy](#)

[ANTERIOR](#)

[PRÓXIMO](#)

# Os métodos e as exibições do controlador no ASP.NET Core

16/01/2019 • 15 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

Temos um bom começo para o aplicativo de filme, mas a apresentação não é ideal. Por exemplo, **ReleaseDate** deveria ser separado em duas palavras.

The screenshot shows a browser window with the following details:

- Title Bar:** Index - Movie App
- Address Bar:** https://localhost:5001/Movies
- Page Content:**
  - Header: Movie App, Home, Privacy
  - Section: **Index**
  - Link: Create New
  - Table:

Title	ReleaseDate	Genre	Price	
When Harry Met Sally	2/12/1989	Romantic Comedy	7.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Ghostbusters	3/13/1984	Comedy	8.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Ghostbusters 2	2/23/1986	Comedy	9.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Rio Bravo	4/15/1959	Western	3.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
  - Footer: © 2019 - Movie App - [Privacy](#)

Abra o arquivo *Models/Movie.cs* e adicione as linhas realçadas mostradas abaixo:

```

using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace MvcMovie.Models
{
    public class Movie
    {
        public int Id { get; set; }
        public string Title { get; set; }

        [Display(Name = "Release Date")]
        [DataType(DataType.Date)]
        public DateTime ReleaseDate { get; set; }
        public string Genre { get; set; }

        [Column(TypeName = "decimal(18, 2)")]
        public decimal Price { get; set; }
    }
}

```

Abordaremos [DataAnnotations](#) no próximo tutorial. O atributo [Display](#) especifica o que deve ser exibido no nome de um campo (neste caso, “Release Date” em vez de “ReleaseDate”). O atributo [DataType](#) especifica o tipo de dados (Data) e, portanto, as informações de hora armazenadas no campo não são exibidas.

A anotação de dados `[Column(TypeName = "decimal(18, 2)")]` é necessária para que o Entity Framework Core possa mapear corretamente o `Price` para a moeda no banco de dados. Para obter mais informações, veja [Tipos de Dados](#).

Procure o controlador `Movies` e mantenha o ponteiro do mouse pressionado sobre um link **Editar** para ver a URL de destino.

Title	Release Date	Genre	Price	
When Harry Met Sally	2/12/1989	Romantic Comedy	7.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Ghostbusters	3/13/1984	Comedy	8.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Ghostbusters 2	2/23/1986	Comedy	9.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Rio Bravo	4/15/1959	Western	3.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

https://localhost:5001/Movies/Edit/4

Os links **Editar**, **Detalhes** e **Excluir** são gerados pelo Auxiliar de Marcação de Âncora do MVC Core no arquivo `Views/Movies/Index.cshtml`.

```
<a asp-action="Edit" asp-route-id="@item.ID">Edit</a> |
<a asp-action="Details" asp-route-id="@item.ID">Details</a> |
<a asp-action="Delete" asp-route-id="@item.ID">Delete</a>
</td>
</tr>
```

Os [Auxiliares de Marcação](#) permitem que o código do servidor participe da criação e renderização de elementos HTML em arquivos do Razor. No código acima, o `AnchorTagHelper` gera dinamicamente o valor do atributo `href` HTML com base na ID de rota e no método de ação do controlador. Use a opção **Exibir Código-fonte** em seu navegador favorito ou as ferramentas do desenvolvedor para examinar a marcação gerada. Uma parte do HTML gerado é mostrada abaixo:

```
<td>
<a href="/Movies/Edit/4"> Edit </a> |
<a href="/Movies/Details/4"> Details </a> |
<a href="/Movies/Delete/4"> Delete </a>
</td>
```

Lembre-se do formato do [roteamento](#) definido no arquivo *Startup.cs*:

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});
```

O ASP.NET Core converte `https://localhost:5001/Movies/Edit/4` de uma solicitação no método de ação `Edit` do controlador `Movies` com o parâmetro `Id` igual a 4. (Os métodos do controlador também são conhecidos como métodos de ação.)

Os [Auxiliares de Marcação](#) são um dos novos recursos mais populares do ASP.NET Core. Para obter mais informações, consulte [Recursos adicionais](#).

Abra o controlador `Movies` e examine os dois métodos de ação `Edit`. O código a seguir mostra o método `HTTP GET Edit`, que busca o filme e popula o formato de edição gerado pelo arquivo *Edit.cshtml* do Razor.

```
// GET: Movies/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var movie = await _context.Movie.FindAsync(id);
    if (movie == null)
    {
        return NotFound();
    }
    return View(movie);
}
```

O código a seguir mostra o método `HTTP POST Edit`, que processa os valores de filmes postados:

```

// POST: Movies/Edit/5
// To protect from overposting attacks, please enable the specific properties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("ID,Title,ReleaseDate,Genre,Price")] Movie movie)
{
    if (id != movie.ID)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(movie);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!MovieExists(movie.ID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction("Index");
    }
    return View(movie);
}

```

```

// GET: Movies/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var movie = await _context.Movie.SingleOrDefaultAsync(m => m.ID == id);
    if (movie == null)
    {
        return NotFound();
    }
    return View(movie);
}

```

O código a seguir mostra o método `HTTP POST Edit`, que processa os valores de filmes postados:

```

// POST: Movies/Edit/5
// To protect from overposting attacks, please enable the specific properties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("ID,Title,ReleaseDate,Genre,Price")] Movie movie)
{
    if (id != movie.ID)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(movie);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!MovieExists(movie.ID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction("Index");
    }
    return View(movie);
}

```

O atributo `[Bind]` é uma maneira de proteger contra o [excesso de postagem](#). Você somente deve incluir as propriedades do atributo `[Bind]` que deseja alterar. Para obter mais informações, consulte [Proteger o controlador contra o excesso de postagem](#). [ViewModels](#) fornece uma abordagem alternativa para prevenir o excesso de postagem.

Observe se o segundo método de ação `Edit` é precedido pelo atributo `[HttpPost]`.

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("ID,Title,ReleaseDate,Genre,Price")] Movie movie)
{
    if (id != movie.ID)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(movie);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!MovieExists(movie.ID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(movie);
}
```

```

// POST: Movies/Edit/5
// To protect from overposting attacks, please enable the specific properties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("ID,Title,ReleaseDate,Genre,Price")] Movie movie)
{
    if (id != movie.ID)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(movie);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!MovieExists(movie.ID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction("Index");
    }
    return View(movie);
}

```

O atributo `HttpPost` especifica que esse método `Edit` pode ser invocado *somente* para solicitações `POST`. Você pode aplicar o atributo `[HttpGet]` ao primeiro método de edição, mas isso não é necessário porque `[HttpGet]` é o padrão.

O atributo `ValidateAntiForgeryToken` é usado para [prevenir a falsificação de uma solicitação](#) e é associado a um token antifalsificação gerado no arquivo de exibição de edição (`Views/Movies/Edit.cshtml`). O arquivo de exibição de edição gera o token antifalsificação com o [Auxiliar de Marcação de Formulário](#).

```
<form asp-action="Edit">
```

O [Auxiliar de Marcação de Formulário](#) gera um token antifalsificação oculto que deve corresponder ao token antifalsificação gerado `[ValidateAntiForgeryToken]` no método `Edit` do controlador `Movies`. Para obter mais informações, consulte [Falsificação de antissolicitação](#).

O método `HttpGet Edit` usa o parâmetro `ID` de filme, pesquisa o filme usando o método `SingleOrDefaultAsync` do Entity Framework e retorna o filme selecionado para a exibição de Edição. Se um filme não for encontrado, `NotFound` (HTTP 404) será retornado.

```
// GET: Movies/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var movie = await _context.Movie.FindAsync(id);
    if (movie == null)
    {
        return NotFound();
    }
    return View(movie);
}
```

Quando o sistema de scaffolding criou a exibição de Edição, ele examinou a classe `Movie` e o código criado para renderizar os elementos `<label>` e `<input>` de cada propriedade da classe. O seguinte exemplo mostra a exibição de Edição que foi gerada pelo sistema de scaffolding do Visual Studio:

```

@model MvcMovie.Models.Movie

@{
    ViewData["Title"] = "Edit";
}

<h1>Edit</h1>

<h4>Movie</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Edit">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <input type="hidden" asp-for="Id" />
            <div class="form-group">
                <label asp-for="Title" class="control-label"></label>
                <input asp-for="Title" class="form-control" />
                <span asp-validation-for="Title" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="ReleaseDate" class="control-label"></label>
                <input asp-for="ReleaseDate" class="form-control" />
                <span asp-validation-for="ReleaseDate" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Genre" class="control-label"></label>
                <input asp-for="Genre" class="form-control" />
                <span asp-validation-for="Genre" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Price" class="control-label"></label>
                <input asp-for="Price" class="form-control" />
                <span asp-validation-for="Price" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Save" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>

<div>
    <a asp-action="Index">Back to List</a>
</div>

@section Scripts {
    @await Html.RenderPartialAsync("_ValidationScriptsPartial");
}

```

Observe como o modelo de exibição tem uma instrução `@model MvcMovie.Models.Movie` na parte superior do arquivo. `@model MvcMovie.Models.Movie` especifica que a exibição espera que o modelo de exibição seja do tipo `Movie`.

O código com scaffolding usa vários métodos de Auxiliares de Marcação para simplificar a marcação HTML. O – [Auxiliar de Marcação de Rótulo](#) exibe o nome do campo (“Title”, “ReleaseDate”, “Genre” ou “Price”). O [Auxiliar de Marcação de Entrada](#) renderiza um elemento `<input>` HTML. O [Auxiliar de Marcação de Validação](#) exibe todas as mensagens de validação associadas a essa propriedade.

Execute o aplicativo e navegue para a URL `/Movies`. Clique em um link **Editar**. No navegador, exiba a origem da página. O HTML gerado para o elemento `<form>` é mostrado abaixo.

```

<form action="/Movies/Edit/7" method="post">
    <div class="form-horizontal">
        <h4>Movie</h4>
        <hr />
        <div class="text-danger" />
        <input type="hidden" data-val="true" data-val-required="The ID field is required." id="ID" name="ID" value="7" />
        <div class="form-group">
            <label class="control-label col-md-2" for="Genre" />
            <div class="col-md-10">
                <input class="form-control" type="text" id="Genre" name="Genre" value="Western" />
                <span class="text-danger field-validation-valid" data-valmsg-for="Genre" data-valmsg-replace="true"></span>
            </div>
        </div>
        <div class="form-group">
            <label class="control-label col-md-2" for="Price" />
            <div class="col-md-10">
                <input class="form-control" type="text" data-val="true" data-val-number="The field Price must be a number." data-val-required="The Price field is required." id="Price" name="Price" value="3.99" />
                <span class="text-danger field-validation-valid" data-valmsg-for="Price" data-valmsg-replace="true"></span>
            </div>
        </div>
        <!-- Markup removed for brevity -->
        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Save" class="btn btn-default" />
            </div>
        </div>
    </div>
    <input name="__RequestVerificationToken" type="hidden" value="CfDJ8Inyxgp63fRFqUePGvuI5jGZsloJu1L7X9le1gy7NC1lSduCRx9jDQC1rV9pOTTmqUyXnJBXhmrjcUVDJyDUMm7-MF_9rK8aAZdRdlOri7FmKVkRe_2v5LIHGKFcTjPrWPYnc9AdSbomkiOSaTEg7RU" />
</form>

```

Os elementos `<input>` estão em um elemento HTML `<form>` cujo atributo `action` está definido para ser postado para a URL `/Movies/Edit/id`. Os dados de formulário serão postados com o servidor quando o botão `Save` receber um clique. A última linha antes do elemento `</form>` de fechamento mostra o token **XSRF** oculto gerado pelo [Auxiliar de Marcação de Formulário](#).

## Processando a solicitação POST

A lista a seguir mostra a versão `[HttpPost]` do método de ação `Edit`.

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("ID,Title,ReleaseDate,Genre,Price")] Movie movie)
{
    if (id != movie.ID)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(movie);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!MovieExists(movie.ID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(movie);
}
```

```

// POST: Movies/Edit/5
// To protect from overposting attacks, please enable the specific properties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("ID,Title,ReleaseDate,Genre,Price")] Movie movie)
{
    if (id != movie.ID)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(movie);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!MovieExists(movie.ID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction("Index");
    }
    return View(movie);
}

```

O atributo `[ValidateAntiForgeryToken]` valida o token **XSRF** oculto gerado pelo gerador de tokens antifalsificação no [Auxiliar de Marcação de Formulário](#)

O sistema de **model binding** usa os valores de formulário postados e cria um objeto `Movie` que é passado como o parâmetro `movie`. O método `ModelState.IsValid` verifica se os dados enviados no formulário podem ser usados para modificar (editar ou atualizar) um objeto `Movie`. Se os dados forem válidos, eles serão salvos. Os dados de filmes atualizados (editados) são salvos no banco de dados chamando o método `SaveChangesAsync` do contexto de banco de dados. Depois de salvar os dados, o código redireciona o usuário para o método de ação `Index` da classe `MoviesController`, que exibe a coleção de filmes, incluindo as alterações feitas recentemente.

Antes que o formulário seja postado no servidor, a validação do lado do cliente verifica as regras de validação nos campos. Se houver erros de validação, será exibida uma mensagem de erro e o formulário não será postado. Se o JavaScript estiver desabilitado, você não terá a validação do lado do cliente, mas o servidor detectará os valores postados que não forem válidos e os valores de formulário serão exibidos novamente com mensagens de erro. Mais adiante no tutorial, examinaremos a [Validação de Modelos](#) mais detalhadamente. O [Auxiliar de Marcação de Validação](#) no modelo de exibição `Views/Movies/Edit.cshtml` é responsável por exibir as mensagens de erro apropriadas.

Edit - Movie App

Movie App Home Privacy

# Edit

## Movie

Title

Release Date

 x ▲ ▼

The Release Date field is required.

Genre

Price

The field Price must be a number.

[Save](#)

[Back to List](#)

© 2019 - Movie App - Privacy

Todos os métodos `HttpGet` no controlador de filme seguem um padrão semelhante. Eles obtêm um objeto de filme (ou uma lista de objetos, no caso de `Index`) e passam o objeto (modelo) para a exibição. O método `Create` passa um objeto de filme vazio para a exibição `Create`. Todos os métodos que criam, editam, excluem ou, de outro modo, modificam dados fazem isso na sobrecarga `[HttpPost]` do método. A modificação de dados em um método `HTTP GET` é um risco de segurança. A modificação de dados em um método `HTTP GET` também viola as melhores práticas de HTTP e o padrão REST de arquitetura, que especifica que as solicitações GET não devem alterar o estado do aplicativo. Em outras palavras, a execução de uma operação GET deve ser uma operação segura que não tem efeitos colaterais e não modifica os dados persistentes.

## Recursos adicionais

- [Globalização e localização](#)
- [Introdução aos auxiliares de marcação](#)
- [Auxiliares de marca de autor](#)
- [Falsificação anti-solicitação](#)

- Proteger o controlador contra o [excesso de postagem](#)
- [ViewModels](#)
- [Auxiliar de marcação de formulário](#)
- [Auxiliar de marcação de entrada](#)
- [Auxiliar de marcação de rótulo](#)
- [Selecionar o auxiliar de marcação](#)
- [Auxiliar de marcação de validação](#)

[ANTERIOR](#)

[PRÓXIMO](#)

# Adicionar a pesquisa a um aplicativo ASP.NET Core MVC

10/01/2019 • 12 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

Nesta seção, você adiciona a funcionalidade de pesquisa ao método de ação `Index` que permite pesquisar filmes por *gênero* ou *nome*.

Atualize o método `Index` pelo seguinte código:

```
public async Task<IActionResult> Index(string searchString)
{
    var movies = from m in _context.Movie
                 select m;

    if (!String.IsNullOrEmpty(searchString))
    {
        movies = movies.Where(s => s.Title.Contains(searchString));
    }

    return View(await movies.ToListAsync());
}
```

A primeira linha do método de ação `Index` cria uma consulta [LINQ](#) para selecionar os filmes:

```
var movies = from m in _context.Movie
             select m;
```

A consulta é *somente* definida neste ponto; ela **não** foi executada no banco de dados.

Se o parâmetro `searchString` contiver uma cadeia de caracteres, a consulta de filmes será modificada para filtrar o valor da cadeia de caracteres de pesquisa:

```
if (!String.IsNullOrEmpty(searchString))
{
    movies = movies.Where(s => s.Title.Contains(searchString));
}
```

O código `s => s.Title.Contains()` acima é uma [Expressão Lambda](#). Lambdas são usados em consultas [LINQ](#) baseadas em método como argumentos para métodos de operadores de consulta padrão, como o método `Where` ou `Contains` (usado no código acima). As consultas LINQ não são executadas quando são definidas ou no momento em que são modificadas com uma chamada a um método, como `Where`, `Contains` ou `OrderBy`. Em vez disso, a execução da consulta é adiada. Isso significa que a avaliação de uma expressão é atrasada até que seu valor realizado seja, de fato, iterado ou o método `ToListAsync` seja chamado. Para obter mais informações sobre a execução de consulta adiada, consulte [Execução da consulta](#).

Observação: o método `Contains` é executado no banco de dados, não no código C# mostrado acima. A diferenciação de maiúsculas e minúsculas na consulta depende do banco de dados e da ordenação. No SQL Server, `Contains` é mapeado para [SQL LIKE](#), que não diferencia maiúsculas de minúsculas. No SQLite, com a ordenação padrão, ele diferencia maiúsculas de minúsculas.

Navegue para `/Movies/Index`. Acrescente uma cadeia de consulta, como `?searchString=Ghost`, à URL. Os filmes filtrados são exibidos.

Index

Create New

Title	Release Date	Genre	Price	
Ghostbusters	3/13/1984	Comedy	8.99	Edit   Details   Delete
Ghostbusters 2	2/23/1986	Comedy	9.99	Edit   Details   Delete

© 2019 - Movie App - Privacy

Se você alterar a assinatura do método `Index` para que ele tenha um parâmetro chamado `id`, o parâmetro `id` corresponderá ao espaço reservado `{id}` opcional com as rotas padrão definidas em `Startup.cs`.

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});
```

Altere o parâmetro para `id` e todas as ocorrências de `searchString` altere para `id`.

O método `Index` anterior:

```
public async Task<IActionResult> Index(string searchString)
{
    var movies = from m in _context.Movie
                 select m;

    if (!String.IsNullOrEmpty(searchString))
    {
        movies = movies.Where(s => s.Title.Contains(searchString));
    }

    return View(await movies.ToListAsync());
}
```

O método `Index` atualizado com o parâmetro `id`:

```

public async Task<IActionResult> Index(string id)
{
    var movies = from m in _context.Movie
                 select m;

    if (!String.IsNullOrEmpty(id))
    {
        movies = movies.Where(s => s.Title.Contains(id));
    }

    return View(await movies.ToListAsync());
}

```

Agora você pode passar o título de pesquisa como dados de rota (um segmento de URL), em vez de como um valor de cadeia de consulta.

Title	Release Date	Genre	Price	
Ghostbusters	3/13/1984	Comedy	8.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Ghostbusters 2	2/23/1986	Comedy	9.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

© 2019 - Movie App - [Privacy](#)

No entanto, você não pode esperar que os usuários modifiquem a URL sempre que desejarem pesquisar um filme. Agora você adicionará os elementos da interface do usuário para ajudá-los a filtrar filmes. Se você tiver alterado a assinatura do método `Index` para testar como passar o parâmetro `ID` associado à rota, altere-o novamente para que ele use um parâmetro chamado `searchString`:

```

public async Task<IActionResult> Index(string searchString)
{
    var movies = from m in _context.Movie
                 select m;

    if (!String.IsNullOrEmpty(searchString))
    {
        movies = movies.Where(s => s.Title.Contains(searchString));
    }

    return View(await movies.ToListAsync());
}

```

Abra o arquivo `Views/Movies/Index.cshtml` e, em seguida, adicione a marcação `<form>` realçada abaixo:

```
    ViewData["Title"] = "Index";
}

<h2>Index</h2>

<p>
    <a asp-action="Create">Create New</a>
</p>

<form asp-controller="Movies" asp-action="Index">
    <p>
        Title: <input type="text" name="SearchString">
        <input type="submit" value="Filter" />
    </p>
</form>

<table class="table">
    <thead>
```

A marcação `<form>` HTML usa o [Auxiliar de Marcação de Formulário](#). Portanto, quando você enviar o formulário, a cadeia de caracteres de filtro será enviada para a ação `Index` do controlador de filmes. Salve as alterações e, em seguida, teste o filtro.

Title	Release Date	Genre	Price	
Ghostbusters	3/13/1984	Comedy	8.99	Edit   Details   Delete
Ghostbusters 2	2/23/1986	Comedy	9.99	Edit   Details   Delete

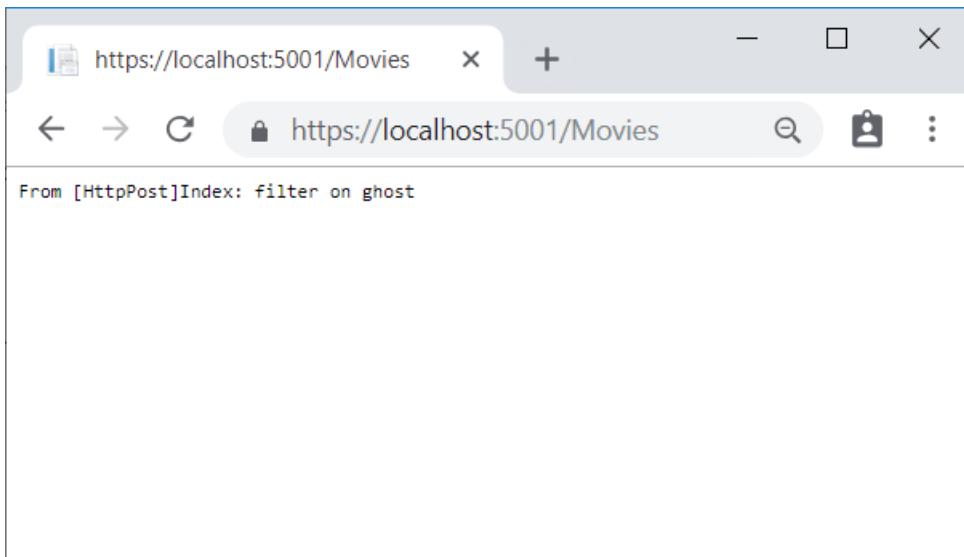
Não há nenhuma sobrecarga `[HttpPost]` do método `Index` que poderia ser esperada. Isso não é necessário, porque o método não está alterando o estado do aplicativo, apenas filtrando os dados.

Você poderá adicionar o método `[HttpPost] Index` a seguir.

```
[HttpPost]
public string Index(string searchString, bool notUsed)
{
    return "From [HttpPost]Index: filter on " + searchString;
}
```

O parâmetro `notUsed` é usado para criar uma sobrecarga para o método `Index`. Falaremos sobre isso mais adiante no tutorial.

Se você adicionar esse método, o chamador de ação fará uma correspondência com o método `[HttpPost] Index` e o método `[HttpPost] Index` será executado conforme mostrado na imagem abaixo.



No entanto, mesmo se você adicionar esta versão `[HttpPost]` do método `Index`, haverá uma limitação na forma de como isso tudo foi implementado. Imagine que você deseja adicionar uma pesquisa específica como Favoritos ou enviar um link para seus amigos para que eles possam clicar para ver a mesma lista filtrada de filmes. Observe que a URL da solicitação HTTP POST é a mesma que a URL da solicitação GET (`localhost:xxxxx/Movies/Index`) – não há nenhuma informação de pesquisa na URL. As informações de cadeia de caracteres de pesquisa são enviadas ao servidor como um [valor de campo de formulário](#). Verifique isso com as ferramentas do Desenvolvedor do navegador ou a excelente [ferramenta Fiddler](#). A imagem abaixo mostra as ferramentas do Desenvolvedor do navegador Chrome:

The screenshot shows the Network tab in the Chrome DevTools developer tools. A red box highlights the 'Network' tab in the top navigation bar. Another red box highlights the 'Movies' entry in the list of network requests. A third red box highlights the 'General' section of the request details, which includes the Request URL, Request Method, Status Code, and other metadata. A fourth red box highlights the 'Form Data' section, which contains the 'SearchString' and 'RequestVerificationToken' fields.

Veja o parâmetro de pesquisa e o token **XSRF** no corpo da solicitação. Observe, conforme mencionado no tutorial anterior, que o [Auxiliar de Marcação de Formulário](#) gera um token antifalsificação **XSRF**. Não modificaremos os dados e, portanto, não precisamos validar o token no método do controlador.

Como o parâmetro de pesquisa está no corpo da solicitação e não na URL, não é possível capturar essas informações de pesquisa para adicionar como Favoritos ou compartilhar com outras pessoas. Corrigimos isso especificando que a solicitação deve ser **HTTP GET**:

```

@model IEnumerable<MvcMovie.Models.Movie>

 @{
     ViewData["Title"] = "Index";
 }

<h1>Index</h1>

<p>
    <a asp-action="Create">Create New</a>
</p>
<form asp-controller="Movies" asp-action="Index" method="get">
    <p>
        Title: <input type="text" name="SearchString">
        <input type="submit" value="Filter" />
    </p>
</form>

<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Title)

```

Agora, quando você enviar uma pesquisa, a URL conterá a cadeia de consulta da pesquisa. A pesquisa também irá para o método de ação `HttpGet Index`, mesmo se você tiver um método `HttpPost Index`.

Title	Release Date	Genre	Price	
Ghostbusters	3/13/1984	Comedy	8.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Ghostbusters 2	2/23/1986	Comedy	9.99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

A seguinte marcação mostra a alteração para a marcação `form`:

```
<form asp-controller="Movies" asp-action="Index" method="get">
```

## Adicionar pesquisa por gênero

Adicione a seguinte classe `MovieGenreViewModel` à pasta `Models`:

```
using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace MvcMovie.Models
{
    public class MovieGenreViewModel
    {
        public List<Movie> Movies;
        public SelectList Genres;
        public string MovieGenre { get; set; }
        public string SearchString { get; set; }
    }
}
```

O modelo de exibição do gênero de filme conterá:

- Uma lista de filmes.
  - Uma `SelectList` que contém a lista de gêneros. Isso permite que o usuário selecione um gênero na lista.
  - `MovieGenre`, que contém o gênero selecionado.
  - `SearchString`, que contém o texto que os usuários inserem na caixa de texto de pesquisa.

Substitua o método `Index` em `MoviesController.cs` pelo seguinte código:

```
// GET: Movies
public async Task<IActionResult> Index(string movieGenre, string searchString)
{
    // Use LINQ to get list of genres.
    IQueryable<string> genreQuery = from m in _context.Movie
                                         orderby m.Genre
                                         select m.Genre;

    var movies = from m in _context.Movie
                 select m;

    if (!string.IsNullOrEmpty(searchString))
    {
        movies = movies.Where(s => s.Title.Contains(searchString));
    }

    if (!string.IsNullOrEmpty(movieGenre))
    {
        movies = movies.Where(x => x.Genre == movieGenre);
    }

    var movieGenreVM = new MovieGenreViewModel
    {
        Genres = new SelectList(await genreQuery.Distinct().ToListAsync(),
        Movies = await movies.ToListAsync()
    };
}

return View(movieGenreVM);
}
```

O código a seguir é uma consulta LINQ que recupera todos os gêneros do banco de dados.

```
// Use LINQ to get list of genres.  
IQueryable<string> genreQuery = from m in _context.Movie  
                                orderby m.Genre  
                                select m.Genre;
```

A `SelectList` de gêneros é criada com a projeção dos gêneros distintos (não desejamos que nossa lista de seleção tenha gêneros duplicados).

Quando o usuário pesquisa o item, o valor de pesquisa é mantido na caixa de pesquisa.

## Adicionar pesquisa por gênero à exibição Índice

Atualize `Index.cshtml` da seguinte maneira:

```

@model MvcMovie.Models.MovieGenreViewModel

 @{
     ViewData["Title"] = "Index";
 }

<h1>Index</h1>

<p>
    <a asp-action="Create">Create New</a>
</p>
<form asp-controller="Movies" asp-action="Index" method="get">
    <p>

        <select asp-for="MovieGenre" asp-items="Model.Genres">
            <option value="">All</option>
        </select>

        Title: <input type="text" asp-for="SearchString" />
        <input type="submit" value="Filter" />
    </p>
</form>

<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Movies[0].Title)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Movies[0].ReleaseDate)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Movies[0].Genre)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Movies[0].Price)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model.Movies)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.Title)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.ReleaseDate)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Genre)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Price)
                </td>
                <td>
                    <a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |
                    <a asp-action="Details" asp-route-id="@item.Id">Details</a> |
                    <a asp-action="Delete" asp-route-id="@item.Id">Delete</a>
                </td>
            </tr>
        }
    </tbody>
</table>

```

Examine a expressão lambda usada no seguinte Auxiliar de HTML:

```
@Html.DisplayNameFor(model => model.Movies[0].Title)
```

No código anterior, o Auxiliar de HTML `DisplayNameFor` inspeciona a propriedade `Title` referenciada na expressão lambda para determinar o nome de exibição. Como a expressão lambda é inspecionada em vez de avaliada, você não recebe uma violação de acesso quando `model`, `model.Movies` ou `model.Movies[0]` é `null` ou vazio. Quando a expressão lambda é avaliada (por exemplo, `@Html.DisplayFor(modelItem => item.Title)` ), os valores da propriedade do modelo são avaliados.

Teste o aplicativo pesquisando por gênero, título do filme e por ambos:

The screenshot shows a browser window titled "Index - Movie App". The address bar contains the URL <https://localhost:5001/Movies?MovieGenre=Comedy&SearchString=2>. The page content is the "Index" view of the Movie App. It features a navigation bar with links for "Movie App", "Home", and "Privacy". Below the navigation is a search/filter section with a dropdown menu set to "Comedy", a text input field containing "2", and a "Filter" button. A table lists movie details: Title, Release Date, Genre, and Price. The first row shown is "Ghostbusters 2", "2/23/1986", "Comedy", and "9.99". To the right of this row are "Edit", "Details", and "Delete" links. At the bottom of the page, there is a copyright notice "© 2019 - Movie App" and a link to "Privacy".

[ANTERIOR](#) [PRÓXIMO](#)

# Adicionar um novo campo a um aplicativo ASP.NET Core MVC

14/01/2019 • 7 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

Nesta seção, as Migrações do [Entity Framework](#) Code First são usadas para:

- Adicionar um novo campo ao modelo.
- Migrar o novo campo para o banco de dados.

Ao usar o Code First do EF para criar automaticamente um banco de dados, o Code First:

- Adiciona uma tabela no banco de dados para rastrear o esquema do banco de dados.
- Verifica se o banco de dados está em sincronia com as classes de modelo das quais foi gerado. Se ele não estiver sincronizado, o EF gerará uma exceção. Isso facilita a descoberta de problemas de código/banco de dados inconsistente.

## Adicionar uma Propriedade de Classificação ao Modelo de Filme

Adicionar uma propriedade `Rating` a *Models/Movie.cs*:

```
public class Movie
{
    public int Id { get; set; }
    public string Title { get; set; }

    [Display(Name = "Release Date")]
    [DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }
    public string Genre { get; set; }

    [Column(TypeName = "decimal(18, 2)")]
    public decimal Price { get; set; }
    public string Rating { get; set; }
}
```

Compile o aplicativo (Ctrl+Shift+B).

Como adicionou um novo campo à classe `Movie`, você também precisará atualizar a lista de permissões de associação para que essa nova propriedade seja incluída. Em *MoviesController.cs*, atualize o atributo `[Bind]` dos métodos de ação `Create` e `Edit` para incluir a propriedade `Rating`:

```
[Bind("ID,Title,ReleaseDate,Genre,Price,Rating")]
```

Atualize os modelos de exibição para exibir, criar e editar a nova propriedade `Rating` na exibição do navegador.

Edite o arquivo */Views/Movies/Index.cshtml* e adicione um campo `Rating`:

```

<thead>
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Movies[0].Title)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Movies[0].ReleaseDate)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Movies[0].Genre)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Movies[0].Price)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Movies[0].Rating)
        </th>
        <th></th>
    </tr>
</thead>
<tbody>
    @foreach (var item in Model.Movies)
    {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Title)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.ReleaseDate)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Genre)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Price)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Rating)
            </td>
            <td>
                <a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |

```

Atualize `/Views/Movies/Create.cshtml` com um campo `Rating`.

- [Visual Studio/Visual Studio para Mac](#)
- [Visual Studio Code](#)

Copie/cole o “grupo de formulário” anterior e permita que o IntelliSense ajude você a atualizar os campos. O IntelliSense funciona com os [Auxiliares de Marcação](#).

```

        </div>
        <div class="form-group">
            <label asp-for="Title" class="col-md-2 control-label"></label>
            <div class="col-md-10">
                <input asp-for="Title" class="form-control" />
                <span asp-validation-for="Title" class="text-danger" />
            </div>
        </div>
        <div class="form-group">
            <label asp-for="R" class="col-md-2 control-label"></label>
            <div class="col->
                <input asp-f<span asp-va<input type=</div>
                <div class="form-gro<div class="col->
                    <input type=</div>
                </div>
            </div>
        </div>
    </div>
</form>

```

Atualize a classe `SeedData` para que ela forneça um valor para a nova coluna. Uma alteração de amostra é mostrada abaixo, mas é recomendável fazer essa alteração em cada `new Movie`.

```

new Movie
{
    Title = "When Harry Met Sally",
    ReleaseDate = DateTime.Parse("1989-1-11"),
    Genre = "Romantic Comedy",
    Rating = "R",
    Price = 7.99M
},

```

O aplicativo não funcionará até que o BD seja atualizado para incluir o novo campo. Se ele tiver sido executado agora, o seguinte `SqlException` será gerado:

```
SqlException: Invalid column name 'Rating'.
```

Este erro ocorre porque a classe de modelo `Movie` atualizada é diferente do esquema da tabela `Movie` do banco de dados existente. (Não há nenhuma coluna `Rating` na tabela de banco de dados.)

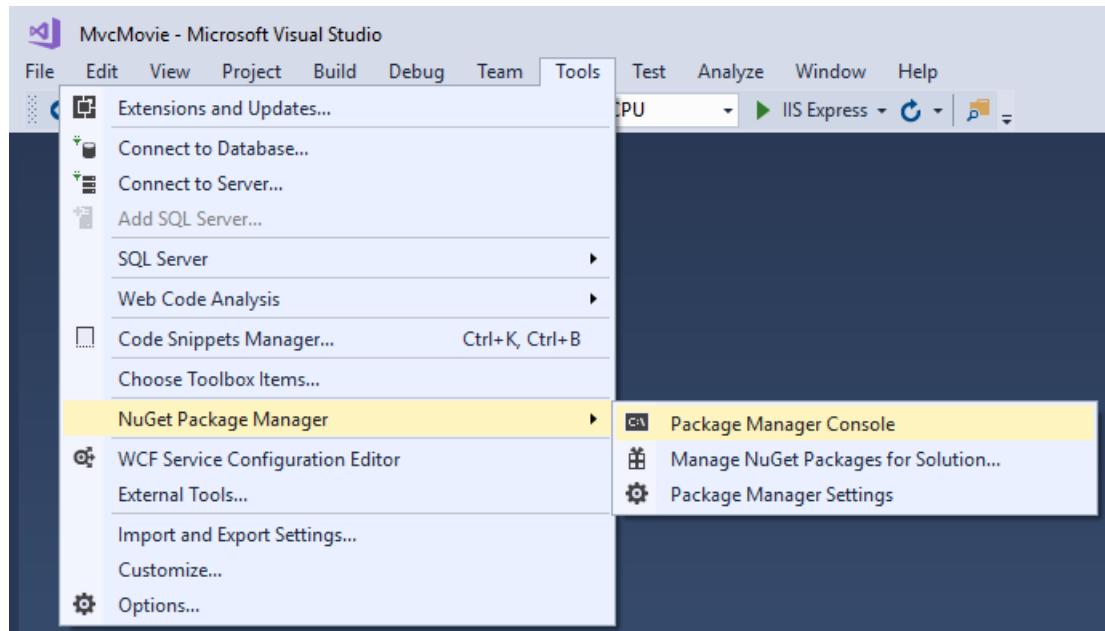
Existem algumas abordagens para resolver o erro:

1. Faça com que o Entity Framework remova automaticamente e recrie o banco de dados com base no novo esquema de classe de modelo. Essa abordagem é muito conveniente no início do ciclo de desenvolvimento, quando você está fazendo o desenvolvimento ativo em um banco de dados de teste. Ela permite que você desenvolva rapidamente o modelo e o esquema de banco de dados juntos. No entanto, a desvantagem é que você perde os dados existentes no banco de dados – portanto, você não deseja usar essa abordagem em um banco de dados de produção! Muitas vezes, o uso de um inicializador para propagar um banco de dados com os dados de teste automaticamente é uma maneira produtiva de desenvolver um aplicativo. Isso é uma boa abordagem para o desenvolvimento inicial e ao usar o SQLite.
2. Modifique explicitamente o esquema do banco de dados existente para que ele corresponda às classes de modelo. A vantagem dessa abordagem é que você mantém os dados. Faça essa alteração manualmente ou criando um script de alteração de banco de dados.
3. Use as Migrações do Code First para atualizar o esquema de banco de dados.

Para este tutorial, as Migrações do Code First são usadas.

- [Visual Studio](#)
- [Visual Studio Code/Visual Studio para Mac](#)

No menu **Ferramentas**, selecione **Gerenciador de Pacotes NuGet > Console do Gerenciador de Pacotes**.



No PMC, insira os seguintes comandos:

```
Add-Migration Rating  
Update-Database
```

O comando `Add-Migration` informa a estrutura de migração para examinar o atual modelo `Movie` com o atual esquema de BD `Movie` e criar o código necessário para migrar o BD para o novo modelo.

O nome "Classificação" é arbitrário e é usado para nomear o arquivo de migração. É útil usar um nome significativo para o arquivo de migração.

Se você excluir todos os registros do BD, o método de inicialização propagará o BD e incluirá o campo `Rating`.

Execute o aplicativo e verifique se você pode criar/editar/exibir filmes com um campo `Rating`. Você deve adicionar o campo `Rating` aos modelos de exibição `Edit`, `Details` e `Delete`.

[ANTERIOR](#)

[PRÓXIMO](#)

# Adicionar a validação a um aplicativo ASP.NET Core MVC

10/01/2019 • 18 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

Nesta seção:

- A lógica de validação é adicionada ao modelo `Movie`.
- Você garante que as regras de validação sejam impostas sempre que um usuário criar ou editar um filme.

## Mantendo o processo DRY

Um dos princípios de design do MVC é o [DRY](#) (“Don't Repeat Yourself”). O ASP.NET Core MVC incentiva você a especificar a funcionalidade ou o comportamento somente uma vez e, em seguida, refleti-lo em qualquer lugar de um aplicativo. Isso reduz a quantidade de código que você precisa escrever e faz com que o código escrito seja menos propenso a erros e mais fácil de testar e manter.

O suporte de validação fornecido pelo MVC e pelo Entity Framework Core Code First é um bom exemplo do princípio DRY em ação. Especifique as regras de validação de forma declarativa em um único lugar (na classe de modelo) e as regras são impostas em qualquer lugar no aplicativo.

## Adicionando regras de validação ao modelo de filme

Abra o arquivo `Movie.cs`. DataAnnotations fornece um conjunto interno de atributos de validação que são aplicados de forma declarativa a qualquer classe ou propriedade. (Também contém atributos de formatação como `DataType`, que ajudam com a formatação e não fornecem nenhuma validação.)

Atualize a classe `Movie` para aproveitar os atributos de validação `Required`, `StringLength`, `RegularExpression` e `Range` internos.

```

public class Movie
{
    public int Id { get; set; }

    [StringLength(60, MinimumLength = 3)]
    [Required]
    public string Title { get; set; }

    [Display(Name = "Release Date")]
    [DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }

    [Range(1, 100)]
    [DataType(DataType.Currency)]
    [Column(TypeName = "decimal(18, 2)")]
    public decimal Price { get; set; }

    [RegularExpression(@"^([A-Z][a-zA-Z\s-]*$")]
    [Required]
    [StringLength(30)]
    public string Genre { get; set; }

    [RegularExpression(@"^([A-Z][a-zA-Z0-9\s-]*$")]
    [StringLength(5)]
    [Required]
    public string Rating { get; set; }
}

```

Os atributos de validação especificam o comportamento que você deseja impor nas propriedades de modelo às quais eles são aplicados:

- Os atributos `Required` e `MinimumLength` indicam que uma propriedade deve ter um valor; porém, nada impede que um usuário insira um espaço em branco para atender a essa validação.
- O atributo `RegularExpression` é usado para limitar quais caracteres podem ser inseridos. No código acima, `Genre` e `Rating` devem usar apenas letras (primeira letra maiúscula, espaço em branco, números e caracteres especiais não são permitidos).
- O atributo `Range` restringe um valor a um intervalo especificado.
- O atributo `StringLength` permite definir o tamanho máximo de uma propriedade de cadeia de caracteres e, opcionalmente, seu tamanho mínimo.
- Os tipos de valor (como `decimal`, `int`, `float`, `DateTime`) são inherentemente necessários e não precisam do atributo `[Required]`.

Ter as regras de validação automaticamente impostas pelo ASP.NET Core ajuda a tornar seu aplicativo mais robusto. Também garante que você não se esqueça de validar algo e inadvertidamente permita dados incorretos no banco de dados.

## Interface do usuário do erro de validação no MVC

Execute o aplicativo e navegue para o controlador `Movies`.

Toque no link **Criar Novo** para adicionar um novo filme. Preencha o formulário com alguns valores inválidos. Assim que a validação do lado do cliente do jQuery detecta o erro, ela exibe uma mensagem de erro.

Create - Movie App

https://localhost:5001/Movies/Create

Movie App Home Privacy

# Create

## Movie

Title

The field Title must be a string with a minimum length of 3 and a maximum length of 60.

Release Date

 x ▲ ▼

The Release Date field is required.

Genre

The field Genre must match the regular expression '^[A-Z]+[a-zA-Z"\s-]\*\$'.

Price

The field Price must be a number.

Rating

The field Rating must match the regular expression '^[A-Z]+[a-zA-Z0-9"\s-]\*\$'.

[Create](#)

[Back to List](#)

© 2019 - Movie App - [Privacy](#)

#### NOTE

Talvez você não consiga inserir vírgulas decimais em campos decimais. Para dar suporte à [validação do jQuery](#) para localidades de idiomas diferentes do inglês que usam uma vírgula (",") para um ponto decimal e formatos de data diferentes do inglês dos EUA, você deve tomar medidas para globalizar o aplicativo. Veja [Problema 4076 do GitHub](#) para obter instruções sobre como adicionar casas decimais.

Observe como o formulário renderizou automaticamente uma mensagem de erro de validação apropriada em cada campo que contém um valor inválido. Os erros são impostos no lado do cliente (usando o JavaScript e o jQuery) e no lado do servidor (caso um usuário tenha o JavaScript desabilitado).

Uma vantagem significativa é que você não precisa alterar uma única linha de código na classe `MoviesController` ou na exibição `Create.cshtml` para habilitar essa interface do usuário de validação. O controlador e as exibições criados anteriormente neste tutorial selecionaram automaticamente as regras de validação especificadas com atributos de validação nas propriedades da classe de modelo `Movie`. Teste a validação usando o método de ação `Edit` e a mesma validação é aplicada.

Os dados de formulário não serão enviados para o servidor enquanto houver erros de validação do lado do cliente. Verifique isso colocando um ponto de interrupção no método `HTTP Post` usando a [ferramenta Fiddler](#) ou as [ferramentas do Desenvolvedor F12](#).

## Como funciona a validação

Talvez você esteja se perguntando como a interface do usuário de validação foi gerada sem atualizações do código no controlador ou nas exibições. O código a seguir mostra os dois métodos `Create`.

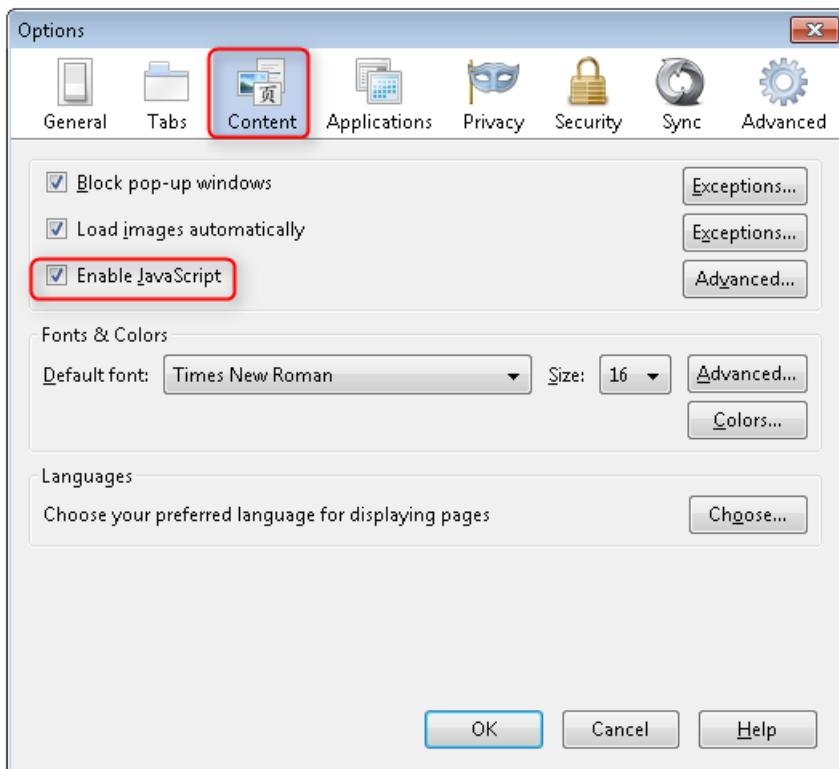
```
// GET: Movies/Create
public IActionResult Create()
{
    return View();
}

// POST: Movies/Create
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create(
    [Bind("ID,Title,ReleaseDate,Genre,Price, Rating")] Movie movie)
{
    if (ModelState.IsValid)
    {
        _context.Add(movie);
        await _context.SaveChangesAsync();
        return RedirectToAction("Index");
    }
    return View(movie);
}
```

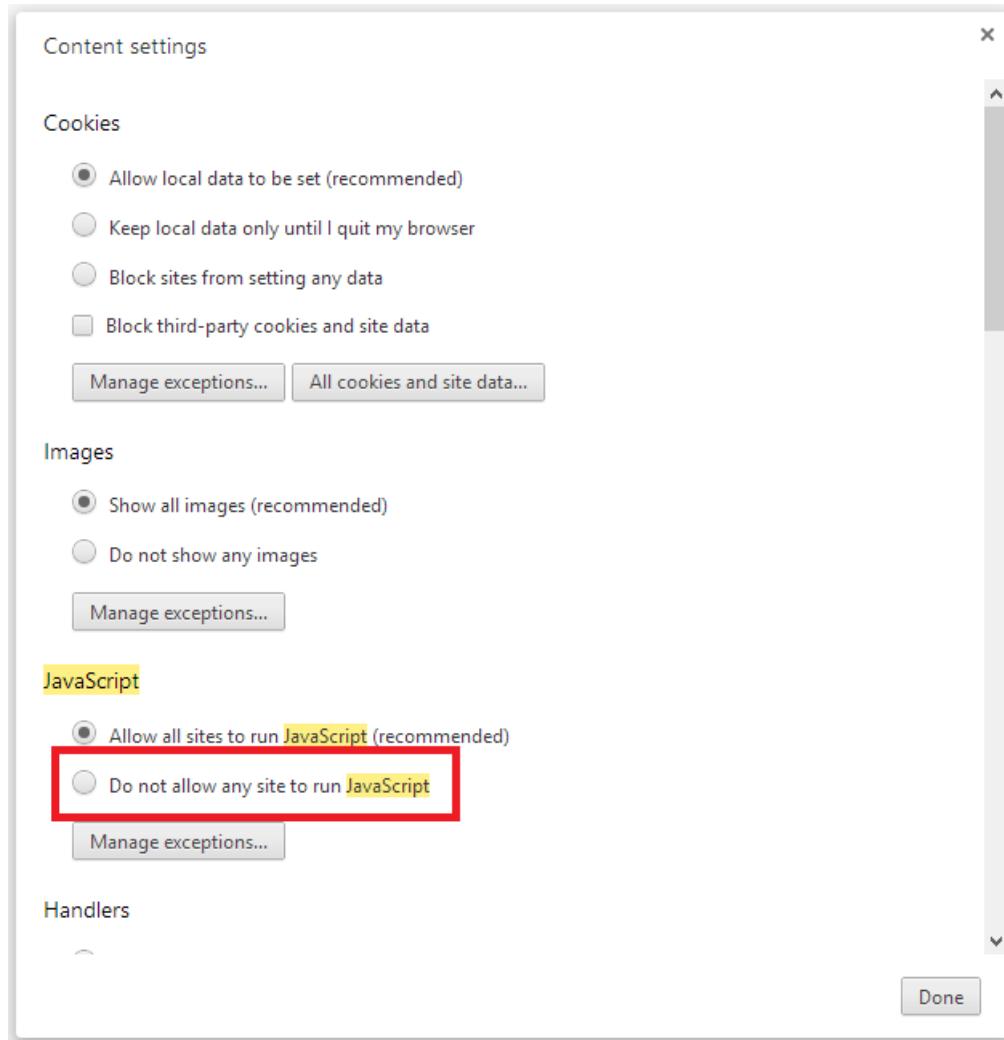
O primeiro método de ação (HTTP GET) `Create` exibe o formulário Criar inicial. A segunda versão (`[HttpPost]`) manipula a postagem de formulário. O segundo método `Create` (a versão `[HttpPost]`) chama `ModelState.IsValid` para verificar se o filme tem erros de validação. A chamada a esse método avalia os atributos de validação que foram aplicados ao objeto. Se o objeto tiver erros de validação, o método `Create` exibirá o formulário novamente. Se não houver erros, o método salvará o novo filme no banco de dados. Em nosso exemplo de filme, o formulário não é postado no servidor quando há erros de validação detectados no lado do cliente; o segundo método `Create` nunca é chamado quando há erros de validação do lado do cliente. Se você desabilitar o JavaScript no navegador, a validação do cliente será desabilitada e você poderá testar o método `Create` HTTP POST `ModelState.IsValid` detectando erros de validação.

Defina um ponto de interrupção no método `[HttpPost] Create` e verifique se o método nunca é chamado; a validação do lado do cliente não enviará os dados de formulário quando forem detectados erros de validação. Se você desabilitar o JavaScript no navegador e, em seguida, enviar o formulário com erros, o ponto de interrupção será atingido. Você ainda pode obter uma validação completa sem o JavaScript.

A imagem a seguir mostra como desabilitar o JavaScript no navegador FireFox.



A imagem a seguir mostra como desabilitar o JavaScript no navegador Chrome.



Depois de desabilitar o JavaScript, poste os dados inválidos e execute o depurador em etapas.

```

74         // POST: Movies/Create
75         // To protect from overposting attacks, please enable t
76         // more details see http://go.microsoft.com/fwlink/?LinkID=185464
77         [HttpPost]
78         [ValidateAntiForgeryToken]
79         public async Task<ActionResult> Create([Bind("ID,Title")]
80         {
81             if (ModelState.IsValid)
82             {
83                 _context.Add(movie);
84                 await _context.SaveChangesAsync();
85                 return RedirectToAction("Index");
86             }
87             return View(movie);
88         }
89     
```

A parte do modelo de exibição `Create.cshtml` é mostrada na seguinte marcação:

```

<h4>Movie</h4>


---



<form asp-action="Create">
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
    <div class="form-group">
        <label asp-for="Title" class="control-label"></label>
        <input asp-for="Title" class="form-control" />
        <span asp-validation-for="Title" class="text-danger"></span>
    </div>


```

@\*Markup removed for brevity.\*@

a marcação anterior é usada pelos métodos de ação para exibir o formulário inicial e exibi-lo novamente em caso de erro.

O [Auxiliar de Marcação de Entrada](#) usa os atributos de `DataAnnotations` e produz os atributos HTML necessários para a Validação do jQuery no lado do cliente. O [Auxiliar de Marcação de Validação](#) exibe erros de validação. Consulte [Validação](#) para obter mais informações.

O que é realmente interessante nessa abordagem é que o controlador nem o modelo de exibição `Create` sabem nada sobre as regras de validação reais que estão sendo impostas ou as mensagens de erro específicas exibidas. As regras de validação e as cadeias de caracteres de erro são especificadas somente na classe `Movie`. Essas mesmas regras de validação são aplicadas automaticamente à exibição `Edit` e a outros modelos de exibição que podem ser criados e que editam o modelo.

Quando você precisar alterar a lógica de validação, faça isso exatamente em um lugar, adicionando atributos de validação ao modelo (neste exemplo, a classe `Movie`). Você não precisa se preocupar se diferentes partes do aplicativo estão inconsistentes com a forma como as regras são impostas – toda a lógica de validação será definida em um lugar e usada em todos os lugares. Isso mantém o código muito limpo e torna-o mais fácil de manter e desenvolver. Além disso, isso significa que você respeitará totalmente o princípio DRY.

## Usando atributos `DataType`

Abra o arquivo `Movie.cs` e examine a classe `Movie`. O namespace `System.ComponentModel.DataAnnotations` fornece atributos de formatação, além do conjunto interno de atributos de validação. Já aplicamos um valor de enumeração `DataType` à data de lançamento e aos campos de preço. O código a seguir mostra as propriedades

`ReleaseDate` e `Price` com o atributo `DataType` apropriado.

```
[Display(Name = "Release Date")]
[DataType(DataType.Date)]
public DateTime ReleaseDate { get; set; }

[Range(1, 100)]
[DataType(DataType.Currency)]
public decimal Price { get; set; }
```

Os atributos `DataType` fornecem dicas apenas para que o mecanismo de exibição formate os dados (e fornece atributos como `<a>` para as URLs e `<a href="mailto:EmailAddress.com">` para o email). Use o atributo `RegularExpression` para validar o formato dos dados. O atributo `DataType` é usado para especificar um tipo de dados mais específico do que o tipo intrínseco de banco de dados; eles não são atributos de validação. Nesse caso, apenas desejamos acompanhar a data, não a hora. A Enumeração `DataType` fornece muitos tipos de dados, como `Date`, `Time`, `PhoneNumber`, `Currency`, `EmailAddress` e muito mais. O atributo `DataType` também pode permitir que o aplicativo forneça automaticamente recursos específicos a um tipo. Por exemplo, um link `mailto:` pode ser criado para `DataType.EmailAddress` e um seletor de data pode ser fornecido para `DataType.Date` em navegadores que dão suporte a HTML5. Os atributos `DataType` emitem atributos `data-` HTML 5 (ou "data dash") que são reconhecidos pelos navegadores HTML 5. Os atributos `DataType` **não** fornecem nenhuma validação.

`DataType.Date` não especifica o formato da data exibida. Por padrão, o campo de dados é exibido de acordo com os formatos padrão com base nas `CultureInfo` do servidor.

O atributo `DisplayFormat` é usado para especificar explicitamente o formato de data:

```
[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
public DateTime ReleaseDate { get; set; }
```

A configuração `ApplyFormatInEditMode` especifica que a formatação também deve ser aplicada quando o valor é exibido em uma caixa de texto para edição. (Talvez você não deseje isso em alguns campos – por exemplo, para valores de moeda, provavelmente, você não deseja exibir o símbolo de moeda na caixa de texto para edição.)

Você pode usar o atributo `DisplayFormat` por si só, mas geralmente é uma boa ideia usar o atributo `DataType`. O atributo `DataType` transmite a semântica dos dados, ao invés de apresentar como renderizá-lo em uma tela e oferece os seguintes benefícios que você não obtém com `DisplayFormat`:

- O navegador pode habilitar os recursos do HTML5 (por exemplo, mostrar um controle de calendário, o símbolo de moeda apropriado à localidade, links de email, etc.)
- Por padrão, o navegador renderizará os dados usando o formato correto de acordo com a localidade.
- O atributo `DataType` pode permitir que o MVC escolha o modelo de campo correto para renderizar os dados (o `DisplayFormat`, se usado por si só, usa o modelo de cadeia de caracteres).

#### NOTE

A validação do jQuery não funciona com os atributos `Range` e `DateTime`. Por exemplo, o seguinte código sempre exibirá um erro de validação do lado do cliente, mesmo quando a data estiver no intervalo especificado:

```
[Range(typeof(DateTime), "1/1/1966", "1/1/2020")]
```

Você precisará desabilitar a validação de data do jQuery para usar o atributo `Range` com `DateTime`. Geralmente, não é uma boa prática compilar datas rígidas nos modelos e, portanto, o uso do atributo `Range` e de `DateTime` não é recomendado.

O seguinte código mostra como combinar atributos em uma linha:

```
public class Movie
{
    public int Id { get; set; }

    [StringLength(60, MinimumLength = 3)]
    public string Title { get; set; }

    [Display(Name = "Release Date"), DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z\s-]*$"), Required, StringLength(30)]
    public string Genre { get; set; }

    [Range(1, 100), DataType(DataType.Currency)]
    [Column(TypeName = "decimal(18, 2)")]
    public decimal Price { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z0-9\s-]*$"), StringLength(5)]
    public string Rating { get; set; }
}
```

Na próxima parte da série, examinaremos o aplicativo e faremos algumas melhorias nos métodos `Details` e `Delete` gerados automaticamente.

## Recursos adicionais

- [Trabalhando com formulários](#)
- [Globalização e localização](#)
- [Introdução aos auxiliares de marcação](#)
- [Auxiliares de marca de autor](#)

[ANTERIOR](#)

[PRÓXIMO](#)

# Examine os métodos Details e Delete de um aplicativo ASP.NET Core

04/02/2019 • 5 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

Abra o controlador Movie e examine o método `Details`:

```
// GET: Movies/Details/5
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var movie = await _context.Movie
        .FirstOrDefaultAsync(m => m.Id == id);
    if (movie == null)
    {
        return NotFound();
    }

    return View(movie);
}
```

O mecanismo de scaffolding MVC que criou este método de ação adiciona um comentário mostrando uma solicitação HTTP que invoca o método. Nesse caso, é uma solicitação GET com três segmentos de URL, o controlador `Movies`, o método `Details` e um valor `id`. Lembre-se que esses segmentos são definidos em `Startup.cs`.

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});
```

O EF facilita a pesquisa de dados usando o método `FirstOrDefaultAsync`. Um recurso de segurança importante interno do método é que o código verifica se o método de pesquisa encontrou um filme antes de tentar fazer algo com ele. Por exemplo, um hacker pode introduzir erros no site alterando a URL criada pelos links de `http://localhost:xxxx/Movies/Details/1` para algo como `http://localhost:xxxx/Movies/Details/12345` (ou algum outro valor que não representa um filme real). Se você não marcou um filme nulo, o aplicativo gerará uma exceção.

Examine os métodos `Delete` e `DeleteConfirmed`.

```

// GET: Movies/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var movie = await _context.Movie
        .FirstOrDefaultAsync(m => m.Id == id);
    if (movie == null)
    {
        return NotFound();
    }

    return View(movie);
}

// POST: Movies/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var movie = await _context.Movie.FindAsync(id);
    _context.Movie.Remove(movie);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

```

Observe que o método `HTTP GET Delete` não exclui o filme especificado, mas retorna uma exibição do filme em que você pode enviar (`HttpPost`) a exclusão. A execução de uma operação de exclusão em resposta a uma solicitação GET (ou, de fato, a execução de uma operação de edição, criação ou qualquer outra operação que altera dados) abre uma falha de segurança.

O método `[HttpPost]` que exclui os dados é chamado `DeleteConfirmed` para fornecer ao método HTTP POST um nome ou uma assinatura exclusiva. As duas assinaturas de método são mostradas abaixo:

```

// GET: Movies/Delete/5
public async Task<IActionResult> Delete(int? id)
{

```

```

// POST: Movies/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{

```

O CLR (Common Language Runtime) exige que os métodos sobrecarregados tenham uma assinatura de parâmetro exclusiva (mesmo nome de método, mas uma lista diferente de parâmetros). No entanto, aqui você precisa de dois métodos `Delete` – um para GET e outro para POST – que têm a mesma assinatura de parâmetro. (Ambos precisam aceitar um único inteiro como parâmetro.)

Há duas abordagens para esse problema e uma delas é fornecer aos métodos nomes diferentes. Foi isso o que o mecanismo de scaffolding fez no exemplo anterior. No entanto, isso apresenta um pequeno problema: o ASP.NET mapeia os segmentos de uma URL para os métodos de ação por nome e, se você renomear um método, o roteamento normalmente não conseguirá encontrar esse método. A solução é o que você vê no exemplo, que é adicionar o atributo `ActionName("Delete")` ao método `DeleteConfirmed`. Esse atributo executa o mapeamento para o sistema de roteamento, de modo que uma URL que inclui `/Delete/` para uma solicitação POST encontre o

método `DeleteConfirmed`.

Outra solução alternativa comum para métodos que têm nomes e assinaturas idênticos é alterar artificialmente a assinatura do método POST para incluir um parâmetro extra (não utilizado). Foi isso o que fizemos em uma postagem anterior quando adicionamos o parâmetro `notUsed`. Você pode fazer o mesmo aqui para o método

`[HttpPost] Delete :`

```
// POST: Movies/Delete/6
[ValidateAntiForgeryToken]
public async Task<IActionResult> Delete(int id, bool notUsed)
```

## Publicar no Azure

Para obter informações sobre como implantar no Azure, consulte [Tutorial: criar um aplicativo Web do .NET Core e do Banco de Dados SQL no Serviço de Aplicativo do Azure](#).

[ANTERIOR](#)

# Exibições no ASP.NET Core MVC

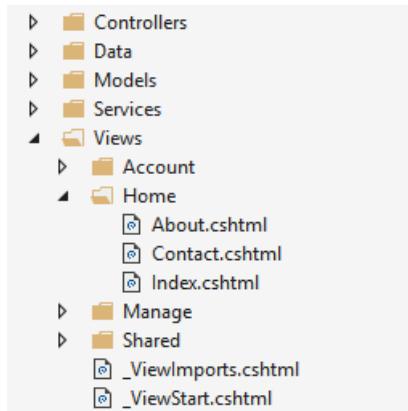
26/12/2018 • 26 minutes to read • [Edit Online](#)

Por [Steve Smith](#) e [Luke Latham](#)

Este documento explica as exibições usadas em aplicativos do ASP.NET Core MVC. Para obter informações sobre páginas do Razor, consulte [Introdução a Páginas do Razor](#).

No padrão MVC (Modelo-Exibição-Controlador), a *exibição* cuida da apresentação de dados do aplicativo e da interação com o usuário. Uma exibição é um modelo HTML com [marcação Razor](#) inserida. A marcação Razor é um código que interage com a marcação HTML para produzir uma página da Web que é enviada ao cliente.

No ASP.NET Core MVC, as exibições são arquivos `.cshtml` que usam a [linguagem de programação C#](#) na marcação Razor. Geralmente, arquivos de exibição são agrupados em pastas nomeadas para cada um dos [controladores](#) do aplicativo. As pastas são armazenadas em uma pasta chamada *Views* na raiz do aplicativo:



O controlador *Home* é representado por uma pasta *Home* dentro da pasta *Views*. A pasta *Home* contém as exibições das páginas da Web *About*, *Contact* e *Index* (home page). Quando um usuário solicita uma dessas três páginas da Web, ações do controlador *Home* determinam qual das três exibições é usada para compilar e retornar uma página da Web para o usuário.

Use [layouts](#) para fornecer seções de páginas da Web consistentes e reduzir repetições de código. Layouts geralmente contêm o cabeçalho, elementos de navegação e menu e o rodapé. O cabeçalho e o rodapé geralmente contêm marcações repetitivas para muitos elementos de metadados, bem como links para ativos de script e estilo. Layouts ajudam a evitar essa marcação repetitiva em suas exibições.

[Exibições parciais](#) reduzem a duplicação de código gerenciando as partes reutilizáveis das exibições. Por exemplo, uma exibição parcial é útil para uma biografia do autor que aparece em várias exibições em um site de blog. Uma biografia do autor é um conteúdo de exibição comum e não requer que um código seja executado para produzi-lo para a página da Web. O conteúdo da biografia do autor é disponibilizado para a exibição usando somente a associação de modelos, de modo que usar uma exibição parcial para esse tipo de conteúdo é ideal.

[Componentes de exibição](#) são semelhantes a exibições parciais no sentido em que permitem reduzir códigos repetitivos, mas são adequados para conteúdos de exibição que requerem que um código seja executado no servidor para renderizar a página da Web. Componentes de exibição são úteis quando o conteúdo renderizado requer uma interação com o banco de dados, como para o carrinho de compras de um site. Os componentes de exibição não ficam limitados à associação de modelos para produzir a saída da página da Web.

## Benefícios do uso de exibições

As exibições ajudam a estabelecer um design **SoC (Separação de Interesses)** dentro de um aplicativo MVC, separando a marcação da interface do usuário de outras partes do aplicativo. Seguir um design de SoC faz com que seu aplicativo seja modular, o que fornece vários benefícios:

- A manutenção do aplicativo é mais fácil, porque ele é melhor organizado. Geralmente, as exibições são agrupadas segundo os recursos do aplicativo. Isso facilita encontrar exibições relacionadas ao trabalhar em um recurso.
- As partes do aplicativo ficam acopladas de forma flexível. Você pode compilar e atualizar as exibições do aplicativo separadamente da lógica de negócios e dos componentes de acesso a dados. É possível modificar os modos de exibição do aplicativo sem precisar necessariamente atualizar outras partes do aplicativo.
- É mais fácil testar as partes da interface do usuário do aplicativo porque as exibições são unidades separadas.
- Devido à melhor organização, é menos provável que você repita accidentalmente seções da interface do usuário.

## Criando uma exibição

Exibições que são específicas de um controlador são criadas na pasta `Views/[NomeDoControlador]`. Exibições que são compartilhadas entre controladores são colocadas na pasta `Views/Shared`. Para criar uma exibição, adicione um novo arquivo e dê a ele o mesmo nome que o da ação de seu controlador associado, com a extensão de arquivo `.cshtml`. Para criar uma exibição correspondente à ação `About` no controlador `Home`, crie um arquivo `About.cshtml` na pasta `Views/Home`:

```
@{
    ViewData["Title"] = "About";
}
<h2>@ViewData["Title"].</h2>
<h3>@ViewData["Message"]</h3>

<p>Use this area to provide additional information.</p>
```

A marcação *Razor* começa com o símbolo `@`. Execute instruções em C# colocando o código C# dentro de **blocos de código Razor** entre chaves (`{ ... }`). Por exemplo, consulte a atribuição de "About" para `ViewData["Title"]` mostrado acima. É possível exibir valores em HTML simplesmente referenciando o valor com o símbolo `@`. Veja o conteúdo dos elementos `<h2>` e `<h3>` acima.

O conteúdo da exibição mostrado acima é apenas uma parte da página da Web inteira que é renderizada para o usuário. O restante do layout da página e outros aspectos comuns da exibição são especificados em outros arquivos de exibição. Para saber mais, consulte o [tópico sobre Layout](#).

## Como controladores especificam exibições

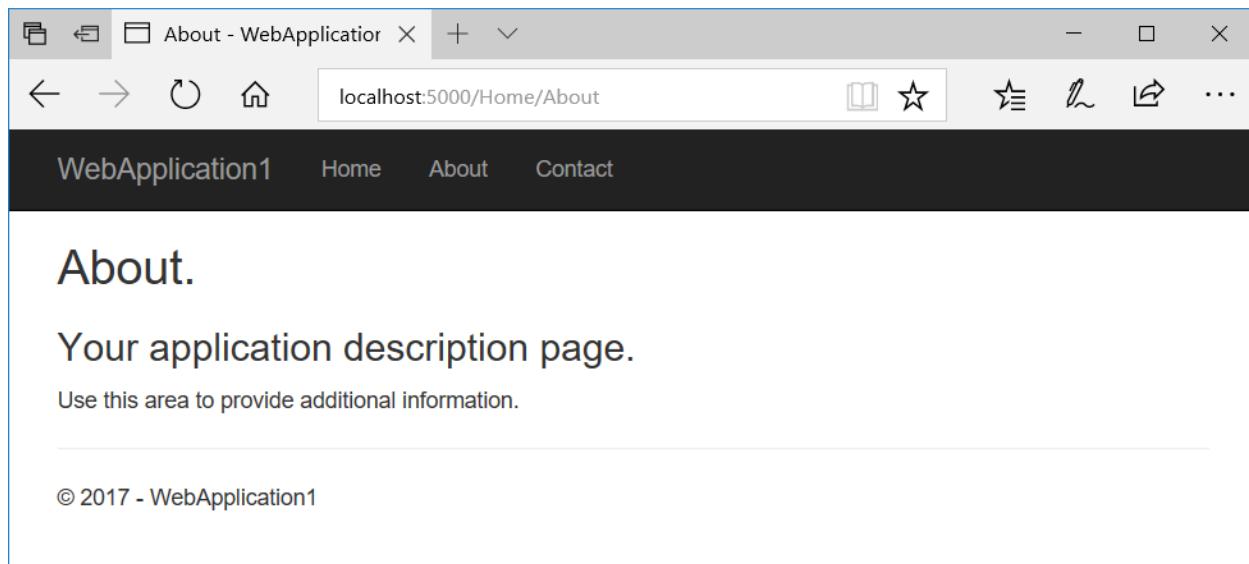
Normalmente, as exibições são retornadas de ações como um `ViewResult`, que é um tipo de `ActionResult`. O método de ação pode criar e retornar um `ViewResult` diretamente, mas normalmente isso não é feito. Como a maioria dos controladores herdam de `Controller`, basta usar o método auxiliar `View` para retornar o `ViewResult`:

`HomeController.cs`

```
public IActionResult About()
{
    ViewData["Message"] = "Your application description page.";

    return View();
}
```

Quando essa ação é retornada, a exibição *About.cshtml* mostrada na seção anterior é renderizada como a seguinte página da Web:



O método auxiliar `View` tem várias sobrecargas. Opcionalmente, você pode especificar:

- Uma exibição explícita a ser retornada:

```
return View("Orders");
```

- Um [modelo](#) a ser passado para a exibição:

```
return View(Orders);
```

- Um modo de exibição e um modelo:

```
return View("Orders", Orders);
```

## Descoberta de exibição

Quando uma ação retorna uma exibição, um processo chamado *descoberta de exibição* ocorre. Esse processo determina qual arquivo de exibição é usado com base no nome da exibição.

O comportamento padrão do método `View ( return View(); )` é retornar uma exibição com o mesmo nome que o método de ação do qual ela é chamada. Por exemplo, o nome do método `ActionResult` de *About* do controlador é usado para pesquisar um arquivo de exibição chamado *About.cshtml*. Primeiro, o tempo de execução pesquisa pela exibição na pasta *Views/[NomeDoControlador]*. Se não encontrar uma exibição correspondente nela, ele procura pela exibição na pasta *Shared*.

Não importa se você retornar implicitamente o `ViewResult` com `return View();` ou se passar explicitamente o nome de exibição para o método `view` com `return View("<ViewName>");`. Nos dois casos, a descoberta de exibição pesquisa por um arquivo de exibição correspondente nesta ordem:

1. *Views/[ControllerName]/[ViewName].cshtml*
2. *Views/Shared/[NomeDaExibição].cshtml*

Um caminho de arquivo de exibição pode ser fornecido em vez de um nome de exibição. Se um caminho absoluto que começa na raiz do aplicativo (ou é iniciado por "/" ou "~ /") estiver sendo usado, a extensão *.cshtml* deverá ser especificada:

```
return View("Views/Home/About.cshtml");
```

Você também pode usar um caminho relativo para especificar exibições em diretórios diferentes sem a extensão `.cshtml`. Dentro do `HomeController`, você pode retornar a exibição `Index` de suas exibições `Manage` com um caminho relativo:

```
return View("../Manage/Index");
```

De forma semelhante, você pode indicar o atual diretório específico do controlador com o prefixo `./`:

```
return View("./About");
```

[Exibições parciais](#) e [componentes de exibição](#) usam mecanismos de descoberta semelhantes (mas não idênticos).

É possível personalizar a convenção padrão de como as exibições ficam localizadas dentro do aplicativo usando um [IViewLocationExpander](#) personalizado.

A descoberta de exibição depende da localização de arquivos de exibição pelo nome do arquivo. Se o sistema de arquivos subjacente diferenciar maiúsculas de minúsculas, os nomes de exibição provavelmente diferenciarão maiúsculas de minúsculas. Para fins de compatibilidade de sistemas operacionais, padronize as maiúsculas e minúsculas dos nomes de controladores e de ações e dos nomes de arquivos e pastas de exibição. Se encontrar um erro indicando que não é possível encontrar um arquivo de exibição ao trabalhar com um sistema de arquivos que diferencia maiúsculas de minúsculas, confirme que o uso de maiúsculas e minúsculas é correspondente entre o arquivo de exibição solicitado e o nome do arquivo de exibição real.

Siga a melhor prática de organizar a estrutura de arquivos de suas exibições de forma a refletir as relações entre controladores, ações e exibições para facilidade de manutenção e clareza.

## Passando dados para exibições

Passe dados para exibições usando várias abordagens:

- Dados fortemente tipados: `viewmodel`
- Dados fracamente tipados
  - `ViewData` (`ViewDataAttribute`)
  - `ViewBag`

### Dados fortemente tipados (`viewmodel`)

A abordagem mais robusta é especificar um tipo de [modelo](#) na exibição. Esse modelo é conhecido como `viewmodel`. Você passa uma instância do tipo `viewmodel` para a exibição da ação.

Usar um `viewmodel` para passar dados para uma exibição permite que a exibição tire proveito da verificação de tipo *forte*. *Tipagem forte* (ou *fortemente tipado*) significa que cada variável e constante têm um tipo definido explicitamente (por exemplo, `string`, `int` ou `DateTime`). A validade dos tipos usados em uma exibição é verificada em tempo de compilação.

O [Visual Studio](#) e o [Visual Studio Code](#) listam membros de classe fortemente tipados usando um recurso chamado [IntelliSense](#). Quando quiser ver as propriedades de um `viewmodel`, digite o nome da variável para o `viewmodel`, seguido por um ponto final (`.`). Isso ajuda você a escrever código mais rapidamente e com menos erros.

Especifique um modelo usando a diretiva `@model`. Use o modelo com `@Model`:

```

@model WebApplication1.ViewModels.Address

<h2>Contact</h2>
<address>
    @Model.Street<br>
    @Model.City, @Model.State @Model.PostalCode<br>
    <abbr title="Phone">P:</abbr> 425.555.0100
</address>

```

Para fornecer o modelo à exibição, o controlador o passa como um parâmetro:

```

public IActionResult Contact()
{
    ViewData["Message"] = "Your contact page.";

    var viewModel = new Address()
    {
        Name = "Microsoft",
        Street = "One Microsoft Way",
        City = "Redmond",
        State = "WA",
        PostalCode = "98052-6399"
    };

    return View(viewModel);
}

```

Não há restrições quanto aos tipos de modelo que você pode fornecer a uma exibição. Recomendamos o uso de viewmodels do tipo POCO (objeto CRL básico) com pouco ou nenhum comportamento (métodos) definido. Geralmente, classes de.viewmodel são armazenadas na pasta *Models* ou em uma pasta *ViewModels* separada na raiz do aplicativo. O.viewmodel *Address* usado no exemplo acima é um.viewmodel POCO armazenado em um arquivo chamado *Address.cs*:

```

namespace WebApplication1.ViewModels
{
    public class Address
    {
        public string Name { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string PostalCode { get; set; }
    }
}

```

Nada impede que você use as mesmas classes para seus tipos de.viewmodel e seus tipos de.modelo de negócios. No entanto, o uso de.modelos separados permite que suas.exibições variem independentemente das partes de lógica de negócios e de acesso a dados do aplicativo. A separação de.modelos e.viewmodels também oferece benefícios de segurança quando os.modelos usam [associação de.modelos](#) e [validação](#) para dados enviados ao aplicativo pelo usuário.

### Dados fracamente tipados (ViewData, atributo ViewData e ViewBag)

`ViewBag` não está disponível nas Páginas do Razor.

Além de.exibições fortemente tipadas, as.exibições têm acesso a uma coleção de dados *fracamente tipados* (também chamada de *tipagem flexível*). Diferente dos.tipos fortes, ter *tipos fracos* (ou *tipos flexíveis*) significa que você não declara explicitamente o tipo dos dados que está usando. Você pode usar a coleção de dados fracamente tipados para transmitir pequenas quantidades de dados para dentro e para fora dos controladores e

das exibições.

PASSAR DADOS ENTRE...	EXEMPLO
Um controlador e uma exibição	Preencher uma lista suspensa com os dados.
Uma exibição e uma <a href="#">exibição de layout</a>	Definir o conteúdo do elemento <title> na exibição de layout de um arquivo de exibição.
Uma <a href="#">exibição parcial</a> e uma exibição	Um widget que exibe dados com base na página da Web que o usuário solicitou.

Essa coleção pode ser referenciada por meio das propriedades `ViewData` ou `ViewBag` em controladores e exibições. A propriedade `ViewData` é um dicionário de objetos fracamente tipados. A propriedade `ViewBag` é um wrapper em torno de `ViewData` que fornece propriedades dinâmicas à coleção de `ViewData` subjacente.

`ViewData` e `ViewBag` são resolvidos dinamicamente em tempo de execução. Uma vez que não oferecem verificação de tipo em tempo de compilação, geralmente ambos são mais propensos a erros do que quando um viewmodel é usado. Por esse motivo, alguns desenvolvedores preferem nunca usar `ViewData` e `ViewBag` ou usá-los o mínimo possível.

## ViewData

`ViewData` é um objeto  [ViewDataDictionary](#) acessado por meio de chaves `string`. Dados de cadeias de caracteres podem ser armazenados e usados diretamente, sem a necessidade de conversão, mas você precisa converter os valores de outros objetos `ViewData` em tipos específicos quando extraí-los. Você pode usar `ViewData` para passar dados de controladores para exibições e dentro das exibições, incluindo [exibições parciais](#) e [layouts](#).

A seguir, temos um exemplo que define valores para uma saudação e um endereço usando `ViewData` em uma ação:

```
public IActionResult SomeAction()
{
    ViewData["Greeting"] = "Hello";
    ViewData["Address"] = new Address()
    {
        Name = "Steve",
        Street = "123 Main St",
        City = "Hudson",
        State = "OH",
        PostalCode = "44236"
    };

    return View();
}
```

Trabalhar com os dados em uma exibição:

```

@{
    // Since Address isn't a string, it requires a cast.
    var address = ViewData["Address"] as Address;
}

@ ViewData["Greeting"] World!

<address>
    @address.Name<br>
    @address.Street<br>
    @address.City, @address.State @address.PostalCode
</address>

```

## Atributo ViewData

Outra abordagem que usa o [ViewDataDictionary](#) é [ViewDataAttribute](#). As propriedades nos controladores ou nos modelos da Página do Razor decoradas com `[ViewData]` têm seus valores armazenados e carregados do dicionário.

No exemplo a seguir, o controlador Home contém uma propriedade `Title` decorada com `[ViewData]`. O método `About` define o título para a exibição About:

```

public class HomeController : Controller
{
    [ViewData]
    public string Title { get; set; }

    public IActionResult About()
    {
        Title = "About Us";
        ViewData["Message"] = "Your application description page.";

        return View();
    }
}

```

Na exibição About, acesse a propriedade `Title` como uma propriedade de modelo:

```
<h1>@Model.Title</h1>
```

No layout, o título é lido a partir do dicionário ViewData:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>@ViewData["Title"] - WebApplication</title>
    ...

```

## ViewBag

`ViewBag` não está disponível nas Páginas do Razor.

`ViewBag` é um objeto [DynamicViewData](#) que fornece acesso dinâmico aos objetos armazenados em `ViewData`. Pode ser mais conveniente trabalhar com `ViewBag`, pois ele não requer uma conversão. O exemplo a seguir mostra como usar `ViewBag` com o mesmo resultado que o uso de `ViewData` acima:

```
public IActionResult SomeAction()
{
    ViewBag.Greeting = "Hello";
    ViewBag.Address = new Address()
    {
        Name = "Steve",
        Street = "123 Main St",
        City = "Hudson",
        State = "OH",
        PostalCode = "44236"
    };
    return View();
}
```

```
@ViewBag.Greeting World!

<address>
    @ViewBag.Address.Name<br>
    @ViewBag.Address.Street<br>
    @ViewBag.Address.City, @ViewBag.Address.State @ViewBag.Address.PostalCode
</address>
```

## Usando ViewData e ViewBag simultaneamente

`ViewBag` não está disponível nas Páginas do Razor.

Como `ViewData` e `ViewBag` fazem referência à mesma coleção `viewData` subjacente, você pode usar `ViewData` e `ViewBag`, além de misturá-los e combiná-los ao ler e gravar valores.

Defina o título usando `ViewBag` e a descrição usando `viewData` na parte superior de uma exibição `About.cshtml`:

```
@{
    Layout = "/Views/Shared/_Layout.cshtml";
    ViewBag.Title = "About Contoso";
    ViewData["Description"] = "Let us tell you about Contoso's philosophy and mission.";
}
```

Leia as propriedades, mas inverta o uso de `viewData` e `ViewBag`. No arquivo `_Layout.cshtml`, obtenha o título usando `ViewData` e a descrição usando `ViewBag`:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>@ViewData["Title"]</title>
    <meta name="description" content="@ViewBag.Description">
    ...

```

Lembre-se de que cadeias de caracteres não exigem uma conversão para `viewData`. Você pode usar `@ViewData["Title"]` sem converter.

Usar `ViewData` e `ViewBag` ao mesmo tempo funciona, assim como misturar e combinar e leitura e a gravação das propriedades. A seguinte marcação é renderizada:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>About Contoso</title>
    <meta name="description" content="Let us tell you about Contoso's philosophy and mission.">
    ...

```

## Resumo das diferenças entre ViewData e ViewBag

`ViewBag` não está disponível em páginas Razor.

- `ViewData`
  - Deriva de `ViewDataDictionary`, de forma que tem propriedades de dicionário que podem ser úteis, como `ContainsKey`, `Add`, `Remove` e `Clear`.
  - Chaves no dicionário são cadeias de caracteres, de forma que espaços em branco são permitidos. Exemplo: `ViewData["Some Key With Whitespace"]`
  - Qualquer tipo diferente de `string` deve ser convertido na exibição para usar `ViewData`.
- `ViewBag`
  - Deriva de `Dynamic ViewData`, de forma que permite a criação de propriedades dinâmicas usando a notação de ponto (`@ViewBag.SomeKey = <value or object>`) e nenhuma conversão é necessária. A sintaxe de `ViewBag` torna mais rápido adicioná-lo a controladores e exibições.
  - Mais simples de verificar quanto à presença de valores nulos. Exemplo: `@ViewBag.Person?.Name`

## Quando usar ViewData ou ViewBag

`ViewData` e `ViewBag` são abordagens igualmente válidas para passar pequenas quantidades de dados entre controladores e exibições. A escolha de qual delas usar é baseada na preferência. Você pode misturar e combinar objetos `ViewData` e `ViewBag`, mas é mais fácil ler e manter o código quando uma abordagem é usada de maneira consistente. Ambas as abordagens são resolvidas dinamicamente em tempo de execução e, portanto, são propensas a causar erros de tempo de execução. Algumas equipes de desenvolvimento as evitam.

### Exibições dinâmicas

Exibições que não declaram um tipo de modelo usando `@model`, mas que têm uma instância de modelo passada a elas (por exemplo, `return View(Address);`) podem referenciar as propriedades da instância dinamicamente:

```
<address>
    @Model.Street<br>
    @Model.City, @Model.State @Model.PostalCode<br>
    <abbr title="Phone">P:</abbr> 425.555.0100
</address>
```

Esse recurso oferece flexibilidade, mas não oferece proteção de compilação ou IntelliSense. Se a propriedade não existir, a geração da página da Web falhará em tempo de execução.

## Mais recursos das exibições

[Auxiliares de Marca](#) facilitam a adição do comportamento do lado do servidor às marcações HTML existentes. O uso de Auxiliares de marca evita a necessidade de escrever código personalizado ou auxiliares em suas exibições. Auxiliares de marca são aplicados como atributos a elementos HTML e são ignorados por editores que não podem processá-los. Isso permite editar e renderizar a marcação da exibição em várias ferramentas.

É possível gerar uma marcação HTML personalizada com muitos auxiliares HTML internos. Uma lógica de interface do usuário mais complexa pode ser tratada por [Componentes de exibição](#). Componentes de exibição fornecem o mesmo SoC oferecido por controladores e exibições. Eles podem eliminar a necessidade de ações e

exibições que lidam com os dados usados por elementos comuns da interface do usuário.

Como muitos outros aspectos do ASP.NET Core, as exibições dão suporte à [injeção de dependência](#), permitindo que serviços sejam [injetados em exibições](#).

# Exibições parciais no ASP.NET Core

30/10/2018 • 16 minutes to read • [Edit Online](#)

Por [Steve Smith](#), [Luke Latham](#), [Maher JENDOUBI](#), [Rick Anderson](#) e [Scott Sauber](#)

Uma exibição parcial é um arquivo de marcação [Razor \(.cshtml\)](#) que renderiza a saída HTML *dentro* da saída processada de outro arquivo de marcação.

O termo *exibição parcial* é usado durante o desenvolvimento de um aplicativo MVC, no qual os arquivos de marcação são chamados de *exibições*, ou de um aplicativo Razor Pages, no qual os arquivos de marcação são chamados de *páginas*. Este tópico refere-se genericamente a exibições do MVC e a páginas do Razor Pages como *arquivos de marcação*.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Quando usar exibições parciais

Exibições parciais são uma maneira eficaz de:

- Dividir arquivos de marcação grandes em componentes menores.

Aproveitar a vantagem de trabalhar com cada parte isolada em uma exibição parcial em um arquivo de marcação grande e complexo, composto por diversas partes lógicas. O código no arquivo de marcação é gerenciável porque a marcação contém somente a estrutura de página geral e as referências a exibições parciais.

- Reduzir a duplicação de conteúdo de marcação comum em arquivos de marcação.

Quando os mesmos elementos de marcação são usados nos arquivos de marcação, uma exibição parcial remove a duplicação de conteúdo de marcação em um arquivo de exibição parcial. Quando a marcação é alterada na exibição parcial, ela atualiza a saída renderizada dos arquivos de marcação que usam a exibição parcial.

As exibições parciais não podem ser usadas para manter os elementos de layout comuns. Os elementos de layout comuns precisam ser especificados nos arquivos [\\_Layout.cshtml](#).

Não use uma exibição parcial em que a lógica de renderização complexa ou a execução de código são necessárias para renderizar a marcação. Em vez de uma exibição parcial, use um [componente de exibição](#).

## Declarar exibições parciais

Uma exibição parcial é um arquivo de marcação [.cshtml](#) mantido dentro da pasta *Exibições* (MVC) ou *Páginas* (Razor Pages).

No ASP.NET Core MVC, um [ViewResult](#) do controlador é capaz de retornar uma exibição ou uma exibição parcial. Uma funcionalidade semelhante está planejada para o Razor Pages no ASP.NET Core 2.2. No Razor Pages, um [PageModel](#) pode retornar um [PartialViewResult](#). A referência e a renderização de exibições parciais são descritas na seção [Referenciar uma exibição parcial](#).

Ao contrário da exibição do MVC ou renderização de página, uma exibição parcial não executa [\\_ViewStart.cshtml](#). Para obter mais informações sobre [\\_ViewStart.cshtml](#), consulte [Layout no ASP.NET Core](#).

Nomes de arquivos de exibição parcial geralmente começam com um sublinhado (`_`). Essa convenção de nomenclatura não é obrigatória, mas ajuda a diferenciar visualmente as exibições parciais das exibições e das

páginas.

Uma exibição parcial é um arquivo de marcação `.cshtml` mantido dentro da pasta `Exibições`.

Um `ViewResult` do controlador é capaz de retornar uma exibição ou uma exibição parcial.

Ao contrário da renderização de exibição do MVC, uma exibição parcial não executa `_ViewStart.cshtml`. Para obter mais informações sobre `_ViewStart.cshtml`, consulte [Layout no ASP.NET Core](#).

Nomes de arquivos de exibição parcial geralmente começam com um sublinhado (`_`). Essa convenção de nomenclatura não é obrigatória, mas ajuda a diferenciar visualmente as exibições parciais das exibições.

## Referenciar uma exibição parcial

Há várias maneiras de referenciar uma exibição parcial em um arquivo de marcação. É recomendável que os aplicativos usem uma das seguintes abordagens de renderização assíncrona:

- [Auxiliar de marcação parcial](#)
- [Auxiliar de HTML assíncrono](#)

Há duas maneiras de referenciar uma exibição parcial em um arquivo de marcação:

- [Auxiliar de HTML assíncrono](#)
- [Auxiliar de HTML síncrono](#)

É recomendável que os aplicativos usem o [Auxiliar de HTML assíncrono](#).

### Auxiliar de marca parcial

O [Auxiliar de Marca Parcial](#) exige o ASP.NET Core 2.1 ou posterior.

O Auxiliar de Marca Parcial renderiza o conteúdo de forma assíncrona e usa uma sintaxe semelhante a HTML:

```
<partial name="_PartialName" />
```

Quando houver uma extensão de arquivo, o Auxiliar de Marca fará referência a uma exibição parcial que precisa estar na mesma pasta que o arquivo de marcação que chama a exibição parcial:

```
<partial name="_PartialName.cshtml" />
```

O exemplo a seguir faz referência a uma exibição parcial da raiz do aplicativo. Caminhos que começam com um til-barra (`~/`) ou uma barra (`/`) referem-se à raiz do aplicativo:

### Páginas do Razor

```
<partial name="~/Pages/Folder/_PartialName.cshtml" />
<partial name="/Pages/Folder/_PartialName.cshtml" />
```

### MVC

```
<partial name="~/Views/Folder/_PartialName.cshtml" />
<partial name="/Views/Folder/_PartialName.cshtml" />
```

O exemplo a seguir faz referência a uma exibição parcial com um caminho relativo:

```
<partial name="..../Account/_PartialName.cshtml" />
```

Para obter mais informações, consulte [Auxiliar de marca parcial no ASP.NET Core](#).

### Auxiliar HTML assíncrono

Ao usar um Auxiliar HTML, a melhor prática é usar `PartialAsync`. `PartialAsync` retorna um tipo `IHtmlContent` encapsulado em um `Task<TResult>`. O método é referenciado prefixando a chamada em espera com um caractere `@`:

```
@await Html.PartialAsync("_PartialName")
```

Quando houver a extensão de arquivo, o Auxiliar de HTML fará referência a uma exibição parcial que precisa estar na mesma pasta que o arquivo de marcação que chama a exibição parcial:

```
@await Html.PartialAsync("_PartialName.cshtml")
```

O exemplo a seguir faz referência a uma exibição parcial da raiz do aplicativo. Caminhos que começam com um til-barra (`~/`) ou uma barra (`/`) referem-se à raiz do aplicativo:

### Páginas do Razor

```
@await Html.PartialAsync("~/Pages/Folder/_PartialName.cshtml")
@await Html.PartialAsync("/Pages/Folder/_PartialName.cshtml")
```

### MVC

```
@await Html.PartialAsync("~/Views/Folder/_PartialName.cshtml")
@await Html.PartialAsync("/Views/Folder/_PartialName.cshtml")
```

O exemplo a seguir faz referência a uma exibição parcial com um caminho relativo:

```
@await Html.PartialAsync("../Account/_LoginPartial.cshtml")
```

Como alternativa, é possível renderizar uma exibição parcial com `RenderPartialAsync`. Esse método não retorna um `IHtmlContent`. Ele transmite a saída renderizada diretamente para a resposta. Como o método não retorna nenhum resultado, ele precisa ser chamado dentro de um bloco de código Razor:

```
@{
    await Html.RenderPartialAsync("_AuthorPartial");
}
```

Como `RenderPartialAsync` transmite conteúdo renderizado, ele apresenta melhor desempenho em alguns cenários. Em situações cruciais para o desempenho, submeta a página a benchmark usando ambas as abordagens e use aquela que gera uma resposta mais rápida.

### Auxiliar de HTML assíncrono

`Partial` e `RenderPartial` são os equivalentes síncronos de `PartialAsync` e `RenderPartialAsync`, respectivamente. Os equivalentes síncronos não são recomendados porque há cenários em que eles realizam deadlock. Os métodos síncronos estão programados para serem removidos em uma versão futura.

## IMPORTANT

Se for necessário executar código, use um [componente de exibição](#) em vez de uma exibição parcial.

Chamar `Partial` ou `RenderPartial` resulta em um aviso do analisador do Visual Studio. Por exemplo, a presença de `Partial` produz a seguinte mensagem de aviso:

Uso de `IHtmlHelper.Partial` pode resultar em deadlocks de aplicativo. Considere usar o Auxiliar de Marca Parcial ou `IHtmlHelper.PartialAsync`.

Substitua as chamadas para `@Html.Partial` por `@await Html.PartialAsync` ou o [Auxiliar de Marca Parcial](#). Para obter mais informações sobre a migração do auxiliar de marca parcial, consulte [Migrar de um auxiliar HTML](#).

## Descoberta de exibição parcial

Quando uma exibição parcial é referenciada pelo nome sem uma extensão de arquivo, os seguintes locais são pesquisados na ordem indicada:

### Páginas do Razor

1. Pasta da página em execução no momento
2. Grafo do diretório acima da pasta da página
3. `/Shared`
4. `/Pages/Shared`
5. `/Views/Shared`

### MVC

1. `/Areas/<Area-Name>/Views/<Controller-Name>`
  2. `/Areas/<Area-Name>/Views/Shared`
  3. `/Views/Shared`
  4. `/Pages/Shared`
- 
1. `/Areas/<Area-Name>/Views/<Controller-Name>`
  2. `/Areas/<Area-Name>/Views/Shared`
  3. `/Views/Shared`

As convenções a seguir se aplicam à descoberta de exibição parcial:

- Diferentes exibições parciais com o mesmo nome de arquivo são permitidos quando as exibições parciais estão em pastas diferentes.
- Ao referenciar uma exibição parcial pelo nome sem uma extensão de arquivo e a exibição parcial está presente na pasta do chamador e na pasta *Compartilhada*, a exibição parcial na pasta do chamador fornece a exibição parcial. Se a exibição parcial não existir na pasta do chamador, ela será fornecida pela pasta *Compartilhada*. Exibições parciais na pasta *Compartilhada* são chamadas de *exibições parciais compartilhadas* ou *exibições parciais padrão*.
- Exibições parciais podem ser *encadeadas*—uma exibição parcial pode chamar outra se uma referência circular não tiver sido formada por chamadas. Caminhos relativos sempre são relativos ao arquivo atual, não à raiz ou ao pai do arquivo.

## NOTE

Um `Razor` `section` definido em uma exibição parcial é invisível para arquivos de marcação pai. A `section` só é visível para a exibição parcial na qual ela está definida.

## Acessar dados de exibições parciais

Quando uma exibição parcial é instanciada, ela recebe uma *cópia* do dicionário `ViewData` do pai. As atualizações feitas nos dados dentro da exibição parcial não são persistidas na exibição pai. Alterações a `ViewData` em uma exibição parcial são perdidas quando a exibição parcial retorna.

O exemplo a seguir demonstra como passar uma instância de  `ViewDataDictionary` para uma exibição parcial:

```
@await Html.PartialAsync("_PartialName", customViewData)
```

Você pode passar um modelo para uma exibição parcial. O modelo pode ser um objeto personalizado. Você pode passar um modelo com `PartialAsync` (renderiza um bloco de conteúdo para o chamador) ou `RenderPartialAsync` (transmite o conteúdo para a saída):

```
@await Html.PartialAsync("_PartialName", model)
```

## Páginas do Razor

A marcação a seguir no aplicativo de exemplo é da página `Pages/ArticlesRP/ReadRP.cshtml`. A página contém duas exibições parciais. A segunda exibição parcial passa um modelo e `ViewData` para a exibição parcial. A sobrecarga do construtor  `ViewDataDictionary` é usada para passar um novo dicionário `ViewData`, retendo ainda o dicionário `ViewData` existente.

```
@model ReadRPMModel

<h2>@Model.Article.Title</h2>
/* Pass the author's name to Pages\Shared\_AuthorPartialRP.cshtml */@
@await Html.PartialAsync("../Shared/_AuthorPartialRP", Model.Article.AuthorName)
@Model.Article.PublicationDate

/* Loop over the Sections and pass in a section and additional ViewData to
the strongly typed Pages\ArticlesRP\_ArticleSectionRP.cshtml partial view. */@
 @{
    var index = 0;

    @foreach (var section in Model.Article.Sections)
    {
        @await Html.PartialAsync("_ArticleSectionRP", section,
            new ViewDataDictionary(ViewData)
            {
                { "index", index }
            })
        index++;
    }
}
```

`Pages/Shared/_AuthorPartialRP.cshtml` é a primeira exibição parcial referenciada pelo arquivo de marcação `ReadRP.cshtml`:

```

@model string
<div>
    <h3>@Model</h3>
    This partial view from /Pages/Shared/_AuthorPartialRP.cshtml.
</div>

```

*Pages/ArticlesRP/\_ArticleSectionRP.cshtml* é a segunda exibição parcial referenciada pelo arquivo de marcação *ReadRP.cshtml*:

```

@using PartialViewsSample.ViewModels
@model ArticleSection

<h3>@Model.Title Index: @ViewData["index"]</h3>
<div>
    @Model.Content
</div>

```

## MVC

A marcação a seguir no aplicativo de exemplo mostra a exibição *Views/Articles/Read.cshtml*. A exibição contém duas exibições parciais. A segunda exibição parcial passa um modelo e `ViewData` para a exibição parcial. A sobrecarga do construtor  `ViewDataDictionary` é usada para passar um novo dicionário  `ViewData`, retendo ainda o dicionário  `ViewData` existente.

```

@model PartialViewsSample.ViewModels.Article

<h2>@Model.Title</h2>
/* Pass the author's name to Views\Shared\_AuthorPartial.cshtml */
@await Html.PartialAsync("_AuthorPartial", Model.AuthorName)
@Model.PublicationDate

/* Loop over the Sections and pass in a section and additional ViewData to
   the strongly typed Views\Articles\_ArticleSection.cshtml partial view. */
@foreach (var section in Model.Sections)
{
    @await Html.PartialAsync("_ArticleSection", section,
                           new ViewDataDictionary(ViewData)
    {
        { "index", index }
    })

    index++;
}
}

```

*Views/Shared/\_AuthorPartial.cshtml* é a primeira exibição parcial referenciada pelo arquivo de marcação *ReadRP.cshtml*:

```

@model string
<div>
    <h3>@Model</h3>
    This partial view from /Views/Shared/_AuthorPartial.cshtml.
</div>

```

*Views/Articles/\_ArticleSection.cshtml* é a segunda exibição parcial referenciada pelo arquivo de marcação *Read.cshtml*:

```
@using PartialViewsSample.ViewModels  
@model ArticleSection  
  
<h3>@Model.Title Index: @ViewData["index"]</h3>  
<div>  
    @Model.Content  
</div>
```

No tempo de execução, as parciais são renderizadas para a saída renderizada do arquivo de marcação pai que, por sua vez, é renderizada dentro do *\_Layout.cshtml* compartilhado. A primeira exibição parcial renderiza a data de publicação e o nome do autor do artigo:

Abraham Lincoln

Esta exibição parcial do <caminho do arquivo de exibição parcial compartilhada>. 19/11/1863 12:00:00  
AM

A segunda exibição parcial renderiza as seções do artigo:

Índice da seção um: 0

Pontuação de quatro e há sete anos...

Índice da seção dois: 1

Agora estamos envolvidos em uma grande guerra civil, testando...

Índice da seção três: 2

Mas, em um sentido mais amplo, não podemos dedicar...

## Recursos adicionais

- [Referência da sintaxe Razor para ASP.NET Core](#)
- [Auxiliares de Marca no ASP.NET Core](#)
- [Auxiliar de marca parcial no ASP.NET Core](#)
- [Componentes de exibição no ASP.NET Core](#)
- [Áreas no ASP.NET Core](#)
  
- [Referência da sintaxe Razor para ASP.NET Core](#)
- [Componentes de exibição no ASP.NET Core](#)
- [Áreas no ASP.NET Core](#)

# Tratar solicitações com controladores no ASP.NET Core MVC

25/06/2018 • 10 minutes to read • [Edit Online](#)

Por [Steve Smith](#) e [Scott Addie](#)

Controladores, ações e resultados da ação são uma parte fundamental de como os desenvolvedores criam aplicativos usando o ASP.NET Core MVC.

## O que é um controlador?

Um controlador é usado para definir e agrupar um conjunto de ações. Uma ação (ou *método de ação*) é um método em um controlador que manipula solicitações. Os controladores agrupam ações semelhantes de forma lógica. Essa agregação de ações permite que conjuntos de regras comuns, como roteamento, cache e autorização, sejam aplicados em conjunto. As solicitações são mapeadas para ações por meio de [roteamento](#).

Por convenção, as classes do controlador:

- Residem na pasta *Controllers* no nível raiz do projeto
- Herdam de `Microsoft.AspNetCore.Mvc.Controller`

Um controlador é uma classe que pode ser instanciada, em que, pelo menos, uma das seguintes condições é verdadeira:

- O nome da classe tem "Controller" como sufixo
- A classe herda de uma classe cujo nome tem "Controller" como sufixo
- A classe está decorada com o atributo `[Controller]`

Uma classe de controlador não deve ter um atributo `[NonController]` associado.

Os controladores devem seguir o [Princípio de Dependências Explícitas](#). Há duas abordagens para implementar esse princípio. Se várias ações do controlador exigem o mesmo serviço, considere o uso da [injeção de construtor](#) para solicitar essas dependências. Se o serviço é necessário para um único método de ação, considere o uso da [Injeção de Ação](#) para solicitar a dependência.

Dentro do padrão **Model-View-Controller**, um controlador é responsável pelo processamento inicial da solicitação e criação de uma instância do modelo. Em geral, as decisões de negócios devem ser tomadas dentro do modelo.

O controlador usa o resultado do processamento do modelo (se houver) e retorna a exibição correta e seus dados da exibição associada ou o resultado da chamada à API. Saiba mais em [Visão geral do ASP.NET Core MVC](#) e em [Introdução ao ASP.NET Core MVC e ao Visual Studio](#).

O controlador é uma abstração no nível da interface do usuário. Suas responsabilidades são garantir que os dados de solicitação sejam válidos e escolher qual exibição (ou resultado de uma API) deve ser retornada. Em aplicativos bem fatorados, ele não inclui diretamente o acesso a dados ou a lógica de negócios. Em vez disso, o controlador delega essas responsabilidades a serviços.

## Definindo ações

Métodos públicos em um controlador, exceto aqueles decorados com o atributo `[NonAction]`, são ações.

Parâmetros em ações são associados aos dados de solicitação e validados usando a [associação de modelos](#). A

validação de modelo ocorre em tudo o que é associado ao modelo. O valor da propriedade `ModelState.IsValid` indica se a associação de modelos e a validação foram bem-sucedidas.

Métodos de ação devem conter uma lógica para mapear uma solicitação para um interesse de negócios. Normalmente, interesses de negócios devem ser representados como serviços acessados pelo controlador por meio da [injeção de dependência](#). Em seguida, as ações mapeiam o resultado da ação de negócios para um estado do aplicativo.

As ações podem retornar qualquer coisa, mas frequentemente retornam uma instância de `IActionResult` (ou `Task<IActionResult>` para métodos assíncronos) que produz uma resposta. O método de ação é responsável por escolher o *tipo de resposta*. O resultado da ação é *responsável pela resposta*.

## Métodos auxiliares do controlador

Os controladores geralmente herdam de `Controller`, embora isso não seja necessário. A derivação de `Controller` fornece acesso a três categorias de métodos auxiliares:

### 1. Métodos que resultam em um corpo de resposta vazio

Nenhum cabeçalho de resposta HTTP `Content-Type` é incluído, pois o corpo da resposta não tem nenhum conteúdo a ser descrito.

Há dois tipos de resultado nessa categoria: Redirecionamento e Código de Status HTTP.

- **Código de Status HTTP**

Esse tipo retorna um código de status HTTP. Alguns métodos auxiliares desse tipo são `BadRequest`, `NotFound` e `Ok`. Por exemplo, `return BadRequest();` produz um código de status 400 quando executado. Quando métodos como `BadRequest`, `NotFound` e `Ok` estão sobrecarregados, eles deixam de se qualificar como respondentes do Código de Status HTTP, pois a negociação de conteúdo está em andamento.

- **Redirecionamento**

Esse tipo retorna um redirecionamento para uma ação ou um destino (usando `Redirect`, `LocalRedirect`, `RedirectToAction` ou `RedirectToRoute`). Por exemplo,  
`return RedirectToAction("Complete", new {id = 123});` redireciona para `Complete`, passando um objeto anônimo.

O tipo de resultado do Redirecionamento é diferente do tipo do Código de Status HTTP, principalmente na adição de um cabeçalho de resposta HTTP `Location`.

### 2. Métodos que resultam em um corpo de resposta não vazio com um tipo de conteúdo predefinido

A maioria dos métodos auxiliares dessa categoria inclui uma propriedade `ContentType`, permitindo que você defina o cabeçalho de resposta `Content-Type` para descrever o corpo da resposta.

Há dois tipos de resultado nessa categoria: [Exibição](#) e [Resposta Formatada](#).

- **Exibir**

Esse tipo retorna uma exibição que usa um modelo para renderizar HTML. Por exemplo,  
`return View(customer);` passa um modelo para a exibição para associação de dados.

- **Resposta Formatada**

Esse tipo retorna JSON ou um formato de troca de dados semelhante para representar um objeto de uma maneira específica. Por exemplo, `return Json(customer);` serializa o objeto fornecido no formato JSON.

Outros métodos comuns desse tipo incluem `File`, `PhysicalFile` e `VirtualFile`. Por exemplo,  
`return PhysicalFile(customerFilePath, "text/xml");` retorna um arquivo XML descrito por um valor do cabeçalho de resposta `Content-Type` igual a "text/xml".

### 3. Métodos que resultam em um corpo de resposta não vazio formatado em um tipo de conteúdo negociado com o cliente

Essa categoria é mais conhecida como **Negociação de Conteúdo**. A Negociação de conteúdo aplica-se sempre que uma ação retorna um tipo [ObjectResult](#) ou algo diferente de uma implementação [IActionResult](#). Uma ação que retorna uma implementação não [IActionResult](#) (por exemplo, `object`) também retorna uma Resposta Formatada.

Alguns métodos auxiliares desse tipo incluem `BadRequest`, `CreatedAtRoute` e `Ok`. Exemplos desses métodos incluem `return BadRequest(modelState);`, `return CreatedAtRoute("routename", values, newobject);` e `return Ok(value);`, respectivamente. Observe que `BadRequest` e `Ok` fazem a negociação de conteúdo apenas quando um valor é passado; sem que um valor seja passado, eles atuam como tipos de resultado do Código de Status HTTP. Por outro lado, o método `CreatedAtRoute` sempre faz a negociação de conteúdo, pois todas as suas sobrecargas exigem que um valor seja passado.

### Interesses paralelos

Normalmente, os aplicativos compartilham partes de seu fluxo de trabalho. Exemplos incluem um aplicativo que exige autenticação para acessar o carrinho de compras ou um aplicativo que armazena dados em cache em algumas páginas. Para executar a lógica antes ou depois de um método de ação, use um *filtro*. O uso de [Filtros em interesses paralelos](#) pode reduzir a duplicação, possibilitando que elas sigam o [princípio DRY \(Don't Repeat Yourself\)](#).

A maioria dos atributos de filtro, como `[Authorize]`, pode ser aplicada no nível do controlador ou da ação, dependendo do nível desejado de granularidade.

O tratamento de erro e o cache de resposta costumam ser interesses paralelos:

- [Tratar erros](#)
- [Cache de resposta](#)

Muitos interesses paralelos podem ser abordados com filtros ou um [middleware](#) personalizado.

# Roteamento para ações do controlador no ASP.NET Core

30/01/2019 • 59 minutes to read • [Edit Online](#)

Por [Ryan Nowak](#) e [Rick Anderson](#)

O ASP.NET Core MVC usa o [middleware](#) de Roteamento para fazer as correspondências das URLs de solicitações de entrada e mapeá-las para ações. As rotas são definidas em atributos ou no código de inicialização. Elas descrevem como deve ser feita a correspondência entre caminhos de URL e ações. As rotas também são usadas para gerar URLs (para links) enviados em resposta.

As ações são roteadas convencionalmente ou segundo os atributos. Colocar uma rota no controlador ou na ação faz com que ela seja roteada segundo o atributo. Para obter mais informações, consulte [Roteamento misto](#).

Este documento explicará as interações entre o MVC e o roteamento e como aplicativos MVC comuns usam recursos de roteamento. Consulte [Roteamento](#) para obter detalhes sobre o roteamento avançado.

## Configurando o middleware de Roteamento

No método `Configurar`, você poderá ver código semelhante a:

```
app.UseMvc(routes =>
{
    routes.MapRoute("default", "{controller=Home}/{action=Index}/{id?}");
});
```

Dentro da chamada para `UseMvc`, `MapRoute` é usado para criar uma única rota, que chamaremos de rota `default`. A maioria dos aplicativos MVC usa uma rota com um modelo semelhante à rota `default`.

O modelo de rota `"{controller=Home}/{action=Index}/{id?}"` pode corresponder a um caminho de URL como `/Products/Details/5` e extrai os valores de rota `{ controller = Products, action = Details, id = 5 }` gerando tokens para o caminho. O MVC tentará localizar um controlador chamado `ProductsController` e executar a ação `Details`:

```
public class ProductsController : Controller
{
    public IActionResult Details(int id) { ... }
}
```

Observe que, neste exemplo, o model binding usaria o valor de `id = 5` para definir o parâmetro `id` como `5` ao invocar essa ação. Consulte [Model binding](#) para obter mais detalhes.

Usando a rota `default`:

```
routes.MapRoute("default", "{controller=Home}/{action=Index}/{id?}");
```

O modelo da rota:

- `{controller=Home}` define `Home` como o `controller` padrão

- `{action=Index}` define `Index` como o `action` padrão
- `{id?}` define `id` como opcional

Parâmetros de rota opcionais e padrão não precisam estar presentes no caminho da URL para que haja uma correspondência. Consulte [Referência de modelo de rota](#) para obter uma descrição detalhada da sintaxe do modelo de rota.

"`{controller=Home}/{action=Index}/{id?}"` pode corresponder ao caminho da URL `/` e produzirá os valores de rota `{ controller = Home, action = Index }`. Os valores de `controller` e `action` usam os valores padrão, `id` não produz um valor, uma vez que não há nenhum segmento correspondente no caminho da URL. O MVC usaria esses valores de rota para selecionar a ação `HomeController` e `Index`:

```
public class HomeController : Controller
{
    public IActionResult Index() { ... }
}
```

Usando essa definição de controlador e modelo de rota, a ação `HomeController.Index` seria executada para qualquer um dos caminhos de URL a seguir:

- `/Home/Index/17`
- `/Home/Index`
- `/Home`
- `/`

O método de conveniência `UseMvcWithDefaultRoute`:

```
app.UseMvcWithDefaultRoute();
```

Pode ser usado para substituir:

```
app.UseMvc(routes =>
{
    routes.MapRoute("default", "{controller=Home}/{action=Index}/{id?}");
});
```

`UseMvc` e `UseMvcWithDefaultRoute` adicionam uma instância de `RouterMiddleware` ao pipeline de middleware. O MVC não interage diretamente com o middleware e usa o roteamento para tratar das solicitações. O MVC é conectado às rotas por meio de uma instância de `MvcRouteHandler`. O código dentro de `UseMvc` é semelhante ao seguinte:

```
var routes = new RouteBuilder(app);

// Add connection to MVC, will be hooked up by calls to MapRoute.
routes.DefaultHandler = new MvcRouteHandler(...);

// Execute callback to register routes.
// routes.MapRoute("default", "{controller=Home}/{action=Index}/{id?}");

// Create route collection and add the middleware.
app.UseRouter(routes.Build());
```

`UseMvc` não define diretamente nenhuma rota, ele adiciona um espaço reservado à coleção de rotas para a

rota `attribute`. A sobrecarga `UseMvc(Action<IRouteBuilder>)` permite adicionar suas próprias rotas e também dá suporte ao roteamento de atributos. `UseMvc` e todas as suas variações adicionam um espaço reservado à rota do atributo – o roteamento de atributos sempre está disponível, independentemente de como você configura `UseMvc`. `UseMvcWithDefaultRoute` define uma rota padrão e dá suporte ao roteamento de atributos. A seção [Roteamento de atributos](#) inclui mais detalhes sobre o roteamento de atributos.

## Roteamento convencional

A rota `default`:

```
routes.MapRoute("default", "{controller=Home}/{action=Index}/{id?}");
```

é um exemplo de *roteamento convencional*. Nós chamamos esse estilo de *roteamento convencional* porque ele estabelece uma *convenção* para caminhos de URL:

- o primeiro segmento do caminho é mapeado para o nome do controlador,
- o segundo é mapeado para o nome da ação,
- o terceiro segmento é usado para uma `id` opcional usado para mapear para uma entidade de modelo.

Usando essa rota `default`, o caminho da URL `/Products/List` é mapeado para a ação `ProductsController.List` e `/Blog/Article/17` é mapeado para `BlogController.Article`. Esse mapeamento é baseado **somente** nos nomes do controlador e da ação e não é baseado em namespaces, locais de arquivos de origem ou parâmetros de método.

### TIP

Usar o roteamento convencional com a rota padrão permite compilar o aplicativo rapidamente sem precisar criar um novo padrão de URL para cada ação que você definir. Para um aplicativo com ações de estilo CRUD, ter consistência para as URLs em seus controladores pode ajudar a simplificar seu código e a tornar sua interface do usuário mais previsível.

### WARNING

O `id` é definido como opcional pelo modelo de rota, o que significa que suas ações podem ser executadas sem a ID fornecida como parte da URL. Normalmente, o que acontecerá se `id` for omitido da URL é que ele será definido como `0` pelo model binding e, dessa forma, não será encontrada no banco de dados nenhuma entidade correspondente a `id == 0`. O roteamento de atributos pode lhe proporcionar controle refinado para tornar a ID obrigatória para algumas ações e não para outras. Por convenção, a documentação incluirá parâmetros opcionais, como `id`, quando for provável que eles apareçam no uso correto.

## Várias rotas

É possível adicionar várias rotas dentro de `UseMvc` adicionando mais chamadas para `MapRoute`. Fazer isso permite que você defina várias convenções ou que adicione rotas convencionais dedicadas a uma ação específica, como:

```

app.UseMvc(routes =>
{
    routes.MapRoute("blog", "blog/{*article}",
        defaults: new { controller = "Blog", action = "Article" });
    routes.MapRoute("default", "{controller=Home}/{action=Index}/{id?}");
});

```

A rota `blog` aqui é uma *rota convencional dedicada*, o que significa que ela usa o sistema de roteamento convencional, mas é dedicada a uma ação específica. Como `controller` e `action` não aparecem no modelo de rota como parâmetros, eles só podem ter os valores padrão e, portanto, essa rota sempre será mapeada para a ação `BlogController.Article`.

As rotas na coleção de rotas são ordenadas e serão processadas na ordem em que forem adicionadas. Portanto, neste exemplo, a rota `blog` será tentada antes da rota `default`.

#### **NOTE**

*Rotas convencionais dedicadas* geralmente usam parâmetros de rota que capturam tudo, como `{*article}`, para capturar a parte restante do caminho da URL. Isso pode fazer com que uma rota fique "muito ambiciosa", ou seja, que faça a correspondência com URLs que deveriam ser correspondidas com outras rotas. Coloque as rotas "ambiciosas" mais adiante na tabela de rotas para solucionar esse problema.

#### **Fallback**

Como parte do processamento de solicitações, o MVC verificará se os valores das rotas podem ser usados para encontrar um controlador e uma ação em seu aplicativo. Se os valores das rotas não corresponderem a uma ação, a rota não será considerada correspondente e a próxima rota será tentada. Isso é chamado de *fallback* e sua finalidade é simplificar casos em que rotas convencionais se sobreponem.

#### **Desambiguação de ações**

Quando duas ações correspondem por meio do roteamento, o MVC precisa resolver a ambiguidade para escolher a "melhor" candidata ou lançar uma exceção. Por exemplo:

```

public class ProductsController : Controller
{
    public IActionResult Edit(int id) { ... }

    [HttpPost]
    public IActionResult Edit(int id, Product product) { ... }
}

```

Esse controlador define duas ações que fariam a correspondência entre caminho da URL `/Products/Edit/17` e os dados da rota `{ controller = Products, action = Edit, id = 17 }`. Este é um padrão comum para controladores MVC em que `Edit(int)` mostra um formulário para editar um produto e `Edit(int, Product)` processa o formulário postado. Para que isso seja possível, o MVC precisa escolher `Edit(int, Product)` quando a solicitação é um `POST` HTTP e `Edit(int)` quando o verbo HTTP é qualquer outra coisa.

O `HttpPostAttribute` (`[HttpPost]`) é uma implementação de `IActionConstraint` que só permite que a ação seja selecionada quando o verbo HTTP é `POST`. A presença de um `IActionConstraint` faz do `Edit(int, Product)` uma "melhor" correspondência do que `Edit(int)`, portanto, `Edit(int, Product)` será tentado primeiro.

Você só precisará gravar implementações personalizadas de `IActionConstraint` em cenários especializados, mas é importante compreender a função de atributos como `HttpPostAttribute` – atributos semelhantes são definidos para outros verbos HTTP. No roteamento convencional, é comum que ações usem o mesmo nome

de ação quando fazem parte de um fluxo de trabalho de `show form -> submit form`. A conveniência desse padrão ficará mais aparente após você revisar a seção [Noções básicas sobre IActionConstraint](#).

Se várias rotas corresponderem e o MVC não puder encontrar uma rota "melhor", ele gerará um `AmbiguousActionException`.

### Nomes de rotas

As cadeias de caracteres `"blog"` e `"default"` nos exemplos a seguir são nomes de rotas:

```
app.UseMvc(routes =>
{
    routes.MapRoute("blog", "blog/{*article}",
        defaults: new { controller = "Blog", action = "Article" });
    routes.MapRoute("default", "{controller=Home}/{action=Index}/{id?}");
});
```

Os nomes de rotas dão a uma rota um nome lógico, de modo que a rota nomeada possa ser usada para geração de URL. Isso simplifica muito a criação de URLs quando a ordenação de rotas poderia complicá-la. Nomes de rotas devem ser exclusivos no nível do aplicativo.

Os nomes de rotas não têm impacto sobre a correspondência de URLs ou o tratamento de solicitações; eles são usados apenas para a geração de URLs. [Roteamento](#) tem informações mais detalhadas sobre geração de URLs, incluindo a geração de URLs em auxiliares específicos do MVC.

## Roteamento de atributo

O roteamento de atributo usa um conjunto de atributos para mapear ações diretamente para modelos de rota. No exemplo a seguir, `app.UseMvc();` é usado no método `Configure` e nenhuma rota é passada. O `HomeController` corresponderá a um conjunto de URLs semelhantes ao que a rota padrão `{controller=Home}/{action=Index}/{id?}` corresponderia:

```
public class HomeController : Controller
{
    [Route("")]
    [Route("Home")]
    [Route("Home/Index")]
    public IActionResult Index()
    {
        return View();
    }
    [Route("Home/About")]
    public IActionResult About()
    {
        return View();
    }
    [Route("Home/Contact")]
    public IActionResult Contact()
    {
        return View();
    }
}
```

A ação `HomeController.Index()` será executada para qualquer um dos caminhos de URL `/`, `/Home` ou `/Home/Index`.

#### NOTE

Este exemplo destaca uma diferença importante de programação entre o roteamento de atributo e o roteamento convencional. O roteamento de atributo requer mais entradas para especificar uma rota; a rota padrão convencional manipula as rotas de forma mais sucinta. No entanto, o roteamento de atributo permite (e exige) o controle preciso de quais modelos de rota se aplicam a cada ação.

Com o roteamento de atributo, o nome do controlador e os nomes de ação não desempenham **nenhuma** função quanto a qual ação é selecionada. Este exemplo corresponderá as mesmas URLs que o exemplo anterior.

```
public class MyDemoController : Controller
{
    [Route("")]
    [Route("Home")]
    [Route("Home/Index")]
    public IActionResult MyIndex()
    {
        return View("Index");
    }
    [Route("Home/About")]
    public IActionResult MyAbout()
    {
        return View("About");
    }
    [Route("Home/Contact")]
    public IActionResult MyContact()
    {
        return View("Contact");
    }
}
```

#### NOTE

Os modelos de rota acima não definem parâmetros de rota para `action`, `area` e `controller`. Na verdade, esses parâmetros de rota não são permitidos em rotas de atributo. Uma vez que o modelo de rota já está associado a uma ação, não faria sentido analisar o nome da ação da URL.

## Roteamento de atributo com atributos `Http[Verb]`

O roteamento de atributo também pode usar atributos `Http[Verb]`, como `HttpPostAttribute`. Todos esses atributos podem aceitar um modelo de rota. Este exemplo mostra duas ações que correspondem ao mesmo modelo de rota:

```
[HttpGet("/products")]
public IActionResult ListProducts()
{
    // ...
}

[HttpPost("/products")]
public IActionResult CreateProduct(...)
{
    // ...
}
```

Para um caminho de URL como `/products`, a ação `ProductsApi.ListProducts` será executada quando o verbo

HTTP for `GET` e `ProductsApi.CreateProduct` será executado quando o verbo HTTP for `POST`. Primeiro, o roteamento de atributo faz a correspondência da URL com o conjunto de modelos de rota definidos por atributos de rota. Quando um modelo de rota for correspondente, restrições de `IActionConstraint` serão aplicadas para determinar quais ações podem ser executadas.

#### TIP

Ao compilar uma API REST, é raro que você queira usar `[Route(...)]` em um método de ação. É melhor usar o `Http*Verb*Attributes` mais específico para ser preciso quanto ao que tem suporte de sua API. Espera-se que clientes de APIs REST saibam quais caminhos e verbos HTTP são mapeados para operações lógicas específicas.

Como uma rota de atributo se aplica a uma ação específica, é fácil fazer com que parâmetros sejam obrigatórios como parte da definição do modelo de rota. Neste exemplo, `id` é obrigatório como parte do caminho da URL.

```
public class ProductsApiController : Controller
{
    [HttpGet("/products/{id}", Name = "Products_List")]
    public IActionResult GetProduct(int id) { ... }
}
```

A ação `ProductsApi.GetProduct(int)` será executada para um caminho de URL como `/products/3`, mas não para um caminho de URL como `/products`. Consulte [Roteamento](#) para obter uma descrição completa de modelos de rota e as opções relacionadas.

## Nome da rota

O código a seguir define um *nome da rota* como `Products_List`:

```
public class ProductsApiController : Controller
{
    [HttpGet("/products/{id}", Name = "Products_List")]
    public IActionResult GetProduct(int id) { ... }
}
```

Nomes de rota podem ser usados para gerar uma URL com base em uma rota específica. Nomes de rota não têm impacto sobre o comportamento de correspondência da URL e são usados somente para geração de URLs. Nomes de rotas devem ser exclusivos no nível do aplicativo.

#### NOTE

Compare isso com a *rota padrão* convencional, que define o parâmetro `id` como opcional (`{id?}`). Essa capacidade de especificar APIs de forma específica tem vantagens, como permitir que `/products` e `/products/5` sejam expedidos para ações diferentes.

## Combinando rotas

Para tornar o roteamento de atributo menos repetitivo, os atributos de rota no controlador são combinados com atributos de rota nas ações individuais. Modelos de rota definidos no controlador precedem modelos de rota nas ações. Colocar um atributo de rota no controlador faz com que **todas** as ações no controlador usem o roteamento de atributo.

```
[Route("products")]
public class ProductsApiController : Controller
{
    [HttpGet]
    public IActionResult ListProducts() { ... }

    [HttpGet("{id}")]
    public ActionResult GetProduct(int id) { ... }
}
```

Neste exemplo, o caminho de URL `/products` pode corresponder a `ProductsApi.ListProducts` e o caminho de URL `/products/5` pode corresponder a `ProductsApi.GetProduct(int)`. Essas duas ações são correspondentes somente ao `GET` HTTP porque são decoradas com o `HttpGetAttribute`.

Modelos de rota aplicados a uma ação, que começam com `/` ou `~/`, não são combinados com modelos de rota aplicados ao controlador. Este exemplo corresponde a um conjunto de caminhos de URL semelhante à *rota padrão*.

```
[Route("Home")]
public class HomeController : Controller
{
    [Route("")]] // Combines to define the route template "Home"
    [Route("Index")] // Combines to define the route template "Home/Index"
    [Route("/")] // Doesn't combine, defines the route template ""
    public IActionResult Index()
    {
        ViewData["Message"] = "Home index";
        var url = Url.Action("Index", "Home");
        ViewData["Message"] = "Home index" + "var url = Url.Action; = " + url;
        return View();
    }

    [Route("About")] // Combines to define the route template "Home/About"
    public IActionResult About()
    {
        return View();
    }
}
```

## Ordenando rotas de atributos

Diferente de rotas convencionais, que são executadas em uma ordem definida, o roteamento de atributo cria uma árvore e faz a correspondência de todas as rotas simultaneamente. O comportamento é como se as entradas de rota fossem colocadas em uma ordem ideal; as rotas mais específicas têm uma chance de ser executadas antes das rotas mais gerais.

Por exemplo, uma rota como `blog/search/{topic}` é mais específica que uma rota como `blog/*article`. Em termos da lógica, a rota `blog/search/{topic}` é "executada" primeiro, por padrão, porque essa é a única ordem que faz sentido. Usando o roteamento convencional, o desenvolvedor é responsável por colocar as rotas na ordem desejada.

Rotas de atributos podem configurar uma ordem, usando a propriedade `Order` de todos os atributos de rota fornecidos pela estrutura. As rotas são processadas segundo uma classificação crescente da propriedade `Order`. A ordem padrão é `0`. Uma rota definida usando `Order = -1` será executada antes de rotas que não definem uma ordem. Uma rota definida usando `Order = 1` será executada após a ordem das rotas padrão.

#### TIP

Evite depender de `Order`. Se o seu espaço de URL exigir valores de ordem explícita para fazer o roteamento corretamente, provavelmente ele também será confuso para os clientes. De modo geral, o roteamento de atributos selecionará a rota correta com a correspondência de URL. Se a ordem padrão usada para a geração de URL não estiver funcionando, usar o nome da rota como uma substituição geralmente será mais simples do que aplicar a propriedade `Order`.

Roteamento do Razor Pages e do controlador do MVC compartilham uma implementação. Para saber mais sobre Ordem de Rota, confira os tópicos do Razor Pages disponíveis em [Convenções de rota e aplicativo do Razor Pages: Ordem de Rota](#).

## Substituição de token em modelos de rota ([controlador] [ação] [área])

Para conveniência, as rotas de atributo dão suporte à *substituição de token* colocando um token entre chaves quadradas (`[ ]`). Os tokens `[action]`, `[area]` e `[controller]` são substituídos pelos valores do nome da ação, do nome da área e do nome do controlador da ação em que a rota é definida. No exemplo a seguir, as ações correspondem a caminhos de URL conforme descrito nos comentários:

```
[Route("[controller]/[action]")]
public class ProductsController : Controller
{
    [HttpGet] // Matches '/Products/List'
    public IActionResult List() {
        // ...
    }

    [HttpGet("{id}")]
    // Matches '/Products/Edit/{id}'
    public IActionResult Edit(int id) {
        // ...
    }
}
```

A substituição de token ocorre como a última etapa da criação das rotas de atributo. O exemplo acima se comportará da mesma forma que o código a seguir:

```
public class ProductsController : Controller
{
    [HttpGet("[controller]/[action]")]
    // Matches '/Products/List'
    public IActionResult List() {
        // ...
    }

    [HttpGet("[controller]/[action]/[id]")]
    // Matches '/Products/Edit/{id}'
    public IActionResult Edit(int id) {
        // ...
    }
}
```

Rotas de atributo também podem ser combinadas com herança. Isso é especialmente eficiente em combinação com a substituição de token.

```
[Route("api/[controller]")]
public abstract class MyBaseController : Controller { ... }

public class ProductsController : MyBaseController
{
    [HttpGet] // Matches '/api/Products'
    public IActionResult List() { ... }

    [HttpPut("{id}")] // Matches '/api/Products/{id}'
    public IActionResult Edit(int id) { ... }
}
```

A substituição de token também se aplica a nomes de rota definidos por rotas de atributo.

`[Route("[controller]/[action]", Name="[controller]_[action]")]` gera um nome de rota exclusivo para cada ação.

Para corresponder ao delimitador de substituição de token literal `[` ou `]`, faça seu escape repetindo o caractere (`[[` ou `]]`).

### Usar um transformador de parâmetro para personalizar a substituição de token

A substituição do token pode ser personalizada usando um transformador de parâmetro. Um transformador de parâmetro implementa `IOutboundParameterTransformer` e transforma o valor dos parâmetros. Por exemplo, um transformador de parâmetro `SlugifyParameterTransformer` personalizado muda o valor de rota `SubscriptionManagement` para `subscription-management`.

O `RouteTokenTransformerConvention` é uma convenção de modelo de aplicativo que:

- Aplica um transformador de parâmetro a todas as rotas de atributo em um aplicativo.
- Personaliza os valores de token de rota de atributo conforme eles são substituídos.

```
public class SubscriptionManagementController : Controller
{
    [HttpGet("[controller]/[action]")] // Matches '/subscription-management/list-all'
    public IActionResult ListAll() { ... }
}
```

O `RouteTokenTransformerConvention` está registrado como uma opção em `ConfigureServices`.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(options =>
    {
        options.Conventions.Add(new RouteTokenTransformerConvention(
            new SlugifyParameterTransformer()));
    });
}

public class SlugifyParameterTransformer : IOutboundParameterTransformer
{
    public string TransformOutbound(object value)
    {
        if (value == null) { return null; }

        // Slugify value
        return Regex.Replace(value.ToString(), "([a-z])([A-Z])", "$1-$2").ToLower();
    }
}
```

## Várias rotas

O roteamento de atributo dá suporte à definição de várias rotas que atingem a mesma ação. O uso mais comum desse recurso é para simular o comportamento da *rota convencional padrão*, conforme mostrado no exemplo a seguir:

```
[Route("[controller]")]
public class ProductsController : Controller
{
    [Route("")]      // Matches 'Products'
    [Route("Index")] // Matches 'Products/Index'
    public IActionResult Index()
}
```

Colocar vários atributos de rota no controlador significa que cada um deles será combinado com cada um dos atributos de rota nos métodos de ação.

```
[Route("Store")]
[Route("[controller]")]
public class ProductsController : Controller
{
    [HttpPost("Buy")]      // Matches 'Products/Buy' and 'Store/Buy'
    [HttpPost("Checkout")] // Matches 'Products/Checkout' and 'Store/Checkout'
    public IActionResult Buy()
}
```

Quando vários atributos de rota (que implementam `IActionConstraint`) são colocados em uma ação, cada restrição da ação combina com o modelo de rota do atributo que a definiu.

```
[Route("api/[controller]")]
public class ProductsController : Controller
{
    [HttpPut("Buy")]      // Matches PUT 'api/Products/Buy'
    [HttpPost("Checkout")] // Matches POST 'api/Products/Checkout'
    public IActionResult Buy()
}
```

#### TIP

Embora o uso de várias rotas em ações possa parecer eficaz, é melhor manter o espaço de URL de seu aplicativo simples e bem definido. Use várias rotas em ações somente quando for necessário; por exemplo, para dar suporte a clientes existentes.

### Especificando parâmetros opcionais, valores padrão e restrições da rota de atributo

Rotas de atributo dão suporte à mesma sintaxe embutida que as rotas convencionais para especificar parâmetros opcionais, valores padrão e restrições.

```
[HttpPost("product/{id:int}")]
public IActionResult ShowProduct(int id)
{
    // ...
}
```

Consulte [Referência de modelo de rota](#) para obter uma descrição detalhada da sintaxe do modelo de rota.

### Atributos de rota personalizados usando `IRouteTemplateProvider`

Todos os atributos de rota fornecidos na estrutura (`[Route(...)]`, `[HttpGet(...)]` etc.) implementam a

interface `IRouteTemplateProvider`. O MVC procura atributos em classes de controlador e métodos de ação quando o aplicativo é iniciado e usa aqueles que implementam `IRouteTemplateProvider` para criar o conjunto inicial de rotas.

Você pode implementar `IRouteTemplateProvider` para definir seus próprios atributos de rota. Cada `IRouteTemplateProvider` permite definir uma única rota com um nome, uma ordem e um modelo de rota personalizado:

```
public class MyApiControllerAttribute : Attribute, IRouteTemplateProvider
{
    public string Template => "api/[controller]";

    public int? Order { get; set; }

    public string Name { get; set; }
}
```

O atributo do exemplo acima configura automaticamente o `Template` como `"api/[controller]"` quando `[MyApiController]` é aplicado.

### Usando o Modelo de Aplicativo para personalizar rotas de atributo

O *modelo de aplicativo* é um modelo de objeto criado durante a inicialização com todos os metadados usados pelo MVC para rotear e executar suas ações. O *modelo de aplicativo* inclui todos os dados reunidos dos atributos de rota (por meio de `IRouteTemplateProvider`). Você pode escrever *convenções* para modificar o modelo do aplicativo no momento da inicialização para personalizar o comportamento do roteamento. Esta seção mostra um exemplo simples de personalização de roteamento usando o modelo de aplicativo.

```

using Microsoft.AspNetCore.Mvc.ApplicationModels;
using System.Linq;
using System.Text;
public class NamespaceRoutingConvention : IControllerModelConvention
{
    private readonly string _baseNamespace;

    public NamespaceRoutingConvention(string baseNamespace)
    {
        _baseNamespace = baseNamespace;
    }

    public void Apply(ControllerModel controller)
    {
        var hasRouteAttributes = controller.Selectors.Any(selector =>
            selector.AttributeRouteModel != null);
        if (hasRouteAttributes)
        {
            // This controller manually defined some routes, so treat this
            // as an override and not apply the convention here.
            return;
        }

        // Use the namespace and controller name to infer a route for the controller.
        //
        // Example:
        //
        // controller.ControllerTypeInfo ->      "My.Application.Admin.UsersController"
        // baseNamespace ->                      "My.Application"
        //
        // template =>                          "Admin/[controller]"
        //
        // This makes your routes roughly line up with the folder structure of your project.
        //

        var namespc = controller.ControllerType.Namespace;
        if (namespc == null)
            return;
        var template = new StringBuilder();
        template.Append(namespc, _baseNamespace.Length + 1,
            namespc.Length - _baseNamespace.Length - 1);
        template.Replace('.', '/');
        template.Append("/[controller]");

        foreach (var selector in controller.Selectors)
        {
            selector.AttributeRouteModel = new AttributeRouteModel()
            {
                Template = template.ToString()
            };
        }
    }
}

```

## Roteamento misto: roteamento de atributo versus roteamento convencional

Aplicativos MVC podem combinar o uso do roteamento convencional e do roteamento de atributo. É comum usar rotas convencionais para controladores que servem páginas HTML para navegadores e usar o roteamento de atributo para controladores que servem APIs REST.

As ações são roteadas convencionalmente ou segundo os atributos. Colocar uma rota no controlador ou na ação faz com que ela seja roteada segundo o atributo. Ações que definem rotas de atributo não podem ser acessadas por meio das rotas convencionais e vice-versa. **Qualquer** atributo de rota no controlador faz com

que todas as ações no atributo de controlador sejam roteadas.

#### NOTE

O que diferencia os dois tipos de sistemas de roteamento é o processo aplicado após uma URL corresponder a um modelo de rota. No roteamento convencional, os valores de rota da correspondência são usados para escolher a ação e o controlador em uma tabela de pesquisa com todas as ações roteadas convencionais. No roteamento de atributo, cada modelo já está associado a uma ação e nenhuma pesquisa adicional é necessária.

## Segmentos complexos

Segmentos complexos (por exemplo, `[Route("/dog{token}cat")]`), são processados combinando literais da direita para a esquerda de uma maneira não Greedy. Veja [o código-fonte](#) para obter uma descrição. Para obter mais informações, confira [esta edição](#).

## Geração de URL

Aplicativos MVC podem usar os recursos de geração de URL do roteamento para gerar links de URL para ações. Gerar URLs elimina a necessidade de codificar URLs, tornando seu código mais robusto e sustentável. Esta seção tem como foco os recursos de geração de URL fornecidos pelo MVC e só aborda as noções básicas de como a geração de URL funciona. Consulte [Roteamento](#) para obter uma descrição detalhada da geração de URL.

A interface `IUrlHelper` é a parte subjacente da infraestrutura entre o MVC e o roteamento para geração de URL. Você encontrará uma instância de `IUrlHelper` disponível por meio da propriedade `Url` em controladores, exibições e componentes de exibição.

Neste exemplo, a interface `IUrlHelper` é usada por meio a propriedade `Controller.Url` para gerar uma URL para outra ação.

```
using Microsoft.AspNetCore.Mvc;

public class UrlGenerationController : Controller
{
    public IActionResult Source()
    {
        // Generates /UrlGeneration/Destination
        var url = Url.Action("Destination");
        return Content($"Go check out {url}, it's really great.");
    }

    public IActionResult Destination()
    {
        return View();
    }
}
```

Se o aplicativo estiver usando a rota convencional padrão, o valor da variável `url` será a cadeia de caracteres do caminho de URL `/UrlGeneration/Destination`. Esse caminho de URL é criado pelo roteamento combinando os valores de rota da solicitação atual (valores de ambiente) com os valores passados para `Url.Action` e substituindo esses valores no modelo de rota:

```
ambient values: { controller = "UrlGeneration", action = "Source" }
values passed to Url.Action: { controller = "UrlGeneration", action = "Destination" }
route template: {controller}/{action}/{id?}

result: /UrlGeneration/Destination
```

Cada parâmetro de rota no modelo de rota tem seu valor substituído por nomes correspondentes com os valores e os valores de ambiente. Um parâmetro de rota que não tem um valor pode usar um valor padrão se houver um ou pode ser ignorado se for opcional (como no caso de `id` neste exemplo). A geração de URL falhará se qualquer parâmetro de rota obrigatório não tiver um valor correspondente. Se a geração de URL falhar para uma rota, a rota seguinte será tentada até que todas as rotas tenham sido tentadas ou que uma correspondência seja encontrada.

O exemplo de `Url.Action` acima pressupõe que o roteamento seja convencional, mas a geração de URL funciona de forma semelhante com o roteamento de atributo, embora os conceitos sejam diferentes. Com o roteamento convencional, os valores de rota são usados para expandir um modelo e os valores de rota para `controller` e `action` normalmente são exibidos no modelo – isso funciona porque as URLs correspondidas pelo roteamento aderem a uma *convenção*. No roteamento de atributo, os valores de rota para `controller` e `action` não podem ser exibidos no modelo; em vez disso, eles são usados para pesquisar o modelo a ser usado.

Este exemplo usa o roteamento de atributo:

```
// In Startup class
public void Configure(IApplicationBuilder app)
{
    app.UseMvc();
}

using Microsoft.AspNetCore.Mvc;

public class UrlGenerationController : Controller
{
    [HttpGet("")]
    public IActionResult Source()
    {
        var url = Url.Action("Destination"); // Generates /custom/url/to/destination
        return Content($"Go check out {url}, it's really great.");
    }

    [HttpGet("custom/url/to/destination")]
    public IActionResult Destination() {
        return View();
    }
}
```

O MVC cria uma tabela de pesquisa de todas as ações de atributo roteadas e faz a correspondência dos valores de `controller` e `action` para selecionar o modelo de rota a ser usado para geração de URL. Na amostra acima, `custom/url/to/destination` é gerado.

## Gerando URLs pelo nome da ação

`Url.Action` (`IUrlHelper` . `Action`) e todas as sobrecargas relacionadas são baseadas na ideia de que você deseja especificar ao que está vinculando, especificando um nome do controlador e um nome da ação.

#### NOTE

Ao usar `Url.Action`, os valores de rota atuais para `controller` e `action` são especificados para você – o valor de `controller` e `action` fazem parte de *valores de ambiente* e de *valores*. O método `Url.Action` sempre usa os valores atuais de `action` e `controller` e gera um caminho de URL que roteia para a ação atual.

O roteamento tenta usar os valores em valores de ambiente para preencher informações que você não forneceu ao gerar uma URL. Usando uma rota como `{a}/{b}/{c}/{d}` e valores de ambiente `{ a = Alice, b = Bob, c = Carol, d = David }`, o roteamento tem informações suficientes para gerar uma URL sem valores adicionais – uma vez que todos os parâmetros de rota têm um valor. Se você tiver adicionado o valor `{ d = Donovan }`, o valor `{ d = David }` será ignorado e o caminho de URL gerado será `Alice/Bob/Carol/Donovan`.

#### WARNING

Caminhos de URL são hierárquicos. No exemplo acima, se você tiver adicionado o valor `{ c = Cheryl }`, ambos os valores `{ c = Carol, d = David }` serão ignorados. Nesse caso, não teremos mais um valor para `d` e a geração de URL falhará. Você precisaria especificar o valor desejado de `c` e `d`. Você pode esperar se deparar com esse problema com a rota padrão (`{controller}/{action}/{id?}`) – mas raramente encontrará esse comportamento na prática, pois `Url.Action` sempre especificará explicitamente um valor de `controller` e `action`.

Sobrecargas maiores de `Url.Action` também usam um objeto adicional de *valores de rota* para fornecer valores para parâmetros de rota diferentes de `controller` e `action`. É mais comum ver isso com `id` como `Url.Action("Buy", "Products", new { id = 17 })`. Por convenção, o objeto de *valores de rota* geralmente é um objeto de tipo anônimo, mas também pode ser um `IDictionary<>` ou um *objeto .NET simples*. Qualquer valor de rota adicional que não corresponder aos parâmetros de rota será colocado na cadeia de caracteres de consulta.

```
using Microsoft.AspNetCore.Mvc;

public class TestController : Controller
{
    public IActionResult Index()
    {
        // Generates /Products/Buy/17?color=red
        var url = Url.Action("Buy", "Products", new { id = 17, color = "red" });
        return Content(url);
    }
}
```

#### TIP

Para criar uma URL absoluta, use uma sobrecarga que aceita um `protocol`:

```
Url.Action("Buy", "Products", new { id = 17 }, protocol: Request.Scheme)
```

## Gerando URLs pela rota

O código acima demonstrou a geração de uma URL passando o nome do controlador e da ação. `IUrlHelper` também fornece a família de métodos `Url.RouteUrl`. Esses métodos são semelhantes a `Url.Action`, mas não copiam os valores atuais de `action` e `controller` para os valores de rota. O uso mais comum é especificar um nome de rota para usar uma rota específica para gerar a URL, geralmente *sem* especificar um nome de controlador ou de ação.

```

using Microsoft.AspNetCore.Mvc;

public class UrlGenerationController : Controller
{
    [HttpGet("")]
    public IActionResult Source()
    {
        var url = Url.RouteUrl("Destination_Route"); // Generates /custom/url/to/destination
        return Content($"See {url}, it's really great.");
    }

    [HttpGet("custom/url/to/destination", Name = "Destination_Route")]
    public IActionResult Destination()
    {
        return View();
    }
}

```

## Gerar URLs em HTML

`IHtmlHelper` fornece os métodos `Html.BeginForm` e `Html.ActionLink` de `HtmlHelper` para gerar elementos `<form>` e `<a>` respectivamente. Esses métodos usam o método `Url.Action` para gerar uma URL e aceitam argumentos semelhantes. Os complementos `Url.RouteUrl` para `HtmlHelper` são `Html.BeginRouteForm` e `Html.RouteLink`, que têm uma funcionalidade semelhante.

TagHelpers geram URLs por meio do TagHelper `form` e do TagHelper `a`. Ambos usam `IUrlHelper` para sua implementação. Consulte [Trabalhando com Formulários](#) para obter mais informações.

Nos modos de exibição, o `IUrlHelper` está disponível por meio da propriedade `url` para qualquer geração de URL ad hoc não abordada acima.

## Gerando URLs nos resultados da ação

Os exemplos acima mostraram o uso de `IUrlHelper` em um controlador, enquanto o uso mais comum em um controlador é gerar uma URL como parte do resultado de uma ação.

As classes base  `ControllerBase` e  `Controller` fornecem métodos de conveniência para resultados de ação que fazem referência a outra ação. Um uso típico é para redirecionar após aceitar a entrada do usuário.

```

public IActionResult Edit(int id, Customer customer)
{
    if (ModelState.IsValid)
    {
        // Update DB with new details.
        return RedirectToAction("Index");
    }
    return View(customer);
}

```

Os métodos de fábrica dos resultados da ação seguem um padrão semelhante aos métodos em `IUrlHelper`.

## Caso especial para rotas convencionais dedicadas

O roteamento convencional pode usar um tipo especial de definição de rota chamado *rota convencional dedicada*. No exemplo a seguir, a rota chamada `blog` é uma rota convencional dedicada.

```

app.UseMvc(routes =>
{
    routes.MapRoute("blog", "blog/{*article}",
        defaults: new { controller = "Blog", action = "Article" });
    routes.MapRoute("default", "{controller=Home}/{action=Index}/{id?}");
});

```

Usando essas definições de rota, `Url.Action("Index", "Home")` gerará o caminho de URL `/` com a rota `default`, mas por quê? Você poderia imaginar que os valores de rota `{ controller = Home, action = Index }` seriam suficientes para gerar uma URL usando `blog` e o resultado seria `/blog?action=Index&controller=Home`.

Rotas convencionais dedicadas dependem de um comportamento especial de valores padrão que não têm um parâmetro de rota correspondente que impeça que a rota seja "muito ambiciosa" com a geração de URLs. Nesse caso, os valores padrão são `{ controller = Blog, action = Article }` e nem `controller` ou `action` aparece como um parâmetro de rota. Quando o roteamento executa a geração de URL, os valores fornecidos devem corresponder aos valores padrão. A geração de URL usando `blog` falhará porque os valores de `{ controller = Home, action = Index }` não correspondem a `{ controller = Blog, action = Article }`. O roteamento, então, faz o fallback para tentar `default`, que é bem-sucedido.

## Áreas

Áreas são um recurso do MVC usado para organizar funcionalidades relacionadas em um grupo como um namespace de roteamento (para ações do controlador) e estrutura de pasta (para exibições) separada. O uso de áreas permite que um aplicativo tenha vários controladores com o mesmo nome, desde que tenham áreas diferentes. O uso de áreas cria uma hierarquia para fins de roteamento, adicionando outro parâmetro de rota, `area` a `controller` e `action`. Esta seção aborda como o roteamento interage com as áreas. Consulte [Áreas](#) para obter detalhes sobre como as áreas são usadas com exibições.

O exemplo a seguir configura o MVC para usar a rota convencional padrão e uma *rota de área* para uma área chamada `Blog`:

```
app.UseMvc(routes =>
{
    routes.MapAreaRoute("blog_route", "Blog",
        "Manage/{controller}/{action}/{id?}");
    routes.MapRoute("default_route", "{controller}/{action}/{id?}");
});
```

Ao fazer a correspondência de um caminho de URL como `/Manage/Users/AddUser`, a primeira rota produzirá os valores de rota `{ area = Blog, controller = Users, action = AddUser }`. O valor de rota `area` é produzido por um valor padrão para `area`. De fato, a rota criada por `MapAreaRoute` é equivalente à seguinte:

```
app.UseMvc(routes =>
{
    routes.MapRoute("blog_route", "Manage/{controller}/{action}/{id?}",
        defaults: new { area = "Blog" }, constraints: new { area = "Blog" });
    routes.MapRoute("default_route", "{controller}/{action}/{id?}");
});
```

`MapAreaRoute` cria uma rota usando um valor padrão e a restrição para `area` usando o nome da área fornecido, nesse caso, `Blog`. O valor padrão garante que a rota sempre produza `{ area = Blog, ... }`, a restrição requer o valor `{ area = Blog, ... }` para geração de URL.

### TIP

O roteamento convencional é dependente da ordem. De modo geral, rotas com áreas devem ser colocadas mais no início na tabela de rotas, uma vez que são mais específicas que rotas sem uma área.

Usando o exemplo acima, os valores de rota corresponderiam à ação a seguir:

```

using Microsoft.AspNetCore.Mvc;

namespace MyApp.Namespace1
{
    [Area("Blog")]
    public class UsersController : Controller
    {
        public IActionResult AddUser()
        {
            return View();
        }
    }
}

```

O `[AreaAttribute]` é o que indica que um controlador faz parte de uma área; dizemos que esse controlador está na área `Blog`. Controladores sem um atributo `[Area]` não são membros de nenhuma área e **não** corresponderão quando o valor de rota `area` for fornecido pelo roteamento. No exemplo a seguir, somente o primeiro controlador listado pode corresponder aos valores de rota

```
{ area = Blog, controller = Users, action = AddUser }.
```

```

using Microsoft.AspNetCore.Mvc;

namespace MyApp.Namespace1
{
    [Area("Blog")]
    public class UsersController : Controller
    {
        public IActionResult AddUser()
        {
            return View();
        }
    }
}

```

```

using Microsoft.AspNetCore.Mvc;

namespace MyApp.Namespace2
{
    // Matches { area = Zebra, controller = Users, action = AddUser }
    [Area("Zebra")]
    public class UsersController : Controller
    {
        public IActionResult AddUser()
        {
            return View();
        }
    }
}

```

```
using Microsoft.AspNetCore.Mvc;

namespace MyApp.Namespace3
{
    // Matches { area = string.Empty, controller = Users, action = AddUser }
    // Matches { area = null, controller = Users, action = AddUser }
    // Matches { controller = Users, action = AddUser }
    public class UsersController : Controller
    {
        public IActionResult AddUser()
        {
            return View();
        }
    }
}
```

#### NOTE

O namespace de cada controlador é mostrado aqui para fins de integridade – caso contrário, os controladores teriam um conflito de nomenclatura e gerariam um erro do compilador. Namespaces de classe não têm efeito sobre o roteamento do MVC.

Os primeiros dois controladores são membros de áreas e correspondem somente quando seus respectivos nomes de área são fornecidos pelo valor de rota `area`. O terceiro controlador não é um membro de nenhuma área e só pode corresponder quando nenhum valor para `area` for fornecido pelo roteamento.

#### NOTE

Em termos de não corresponder a *nenhum valor*, a ausência do valor de `area` é equivalente ao valor de `area` ser nulo ou uma cadeia de caracteres vazia.

Ao executar uma ação dentro de uma área, o valor de rota para `area` estará disponível como um *valor de ambiente* para o roteamento usar para geração de URL. Isso significa que, por padrão, as áreas atuam como se fossem *autoadesivas* para a geração de URL, como demonstrado no exemplo a seguir.

```
app.UseMvc(routes =>
{
    routes.MapAreaRoute("duck_route", "Duck",
        "Manage/{controller}/{action}/{id?}");
    routes.MapRoute("default", "Manage/{controller=Home}/{action=Index}/{id?}");
});
```

```

using Microsoft.AspNetCore.Mvc;

namespace MyApp.Namespace4
{
    [Area("Duck")]
    public class UsersController : Controller
    {
        public IActionResult GenerateURLInArea()
        {
            // Uses the 'ambient' value of area
            var url = Url.Action("Index", "Home");
            // returns /Manage
            return Content(url);
        }

        public IActionResult GenerateURLOutsideOfArea()
        {
            // Uses the empty value for area
            var url = Url.Action("Index", "Home", new { area = "" });
            // returns /Manage/Home/Index
            return Content(url);
        }
    }
}

```

## Entendendo `IActionConstraint`

### NOTE

Esta seção é uma análise aprofundada dos elementos internos da estrutura e de como o MVC escolhe uma ação para ser executada. Um aplicativo típico não precisará de um `IActionConstraint` personalizado

Provavelmente, você já usou `IActionConstraint` mesmo que não esteja familiarizado com a interface. O atributo `[HttpGet]` e atributos `[Http-VERB]` semelhantes implementam `IActionConstraint` para limitar a execução de um método de ação.

```

public class ProductsController : Controller
{
    [HttpGet]
    public IActionResult Edit() { }

    public IActionResult Edit(...) { }
}

```

Presumindo a rota convencional padrão, o caminho de URL `/Products/Edit` produziria os valores `{ controller = Products, action = Edit }`, que corresponderiam a **ambas** as ações mostradas aqui. Na terminologia `IActionConstraint`, diríamos que essas duas ações são consideradas candidatas – uma vez que ambas correspondem aos dados da rota.

Quando for executado, `HttpGetAttribute` indicará que `Edit()` corresponde a *GET* e não corresponde a nenhum outro verbo HTTP. A ação `Edit(...)` não tem restrições definidas e, portanto, corresponderá a qualquer verbo HTTP. Sendo assim, supondo um `POST`, `Edit(...)` será correspondente. Mas, para um `GET`, ambas as ações ainda podem corresponder – no entanto, uma ação com um `IActionConstraint` sempre é considerada *melhor* que uma ação sem. Assim, como `Edit()` tem `[HttpGet]`, ela é considerada mais específica e será selecionada se as duas ações puderem corresponder.

Conceitualmente, `IActionConstraint` é uma forma de *sobrecarga*, mas em vez de uma sobrecarga de

métodos com o mesmo nome, trata-se da sobrecarga entre ações que correspondem à mesma URL. O roteamento de atributo também usa `IActionConstraint` e pode fazer com que ações de controladores diferentes sejam consideradas candidatas.

## Implementando `IActionConstraint`

A maneira mais simples de implementar um `IActionConstraint` é criar uma classe derivada de `System.Attribute` e colocá-la em suas ações e controladores. O MVC descobrirá automaticamente qualquer `IActionConstraint` que for aplicado como atributo. Você pode usar o modelo de aplicativo para aplicar restrições e, provavelmente, essa é a abordagem mais flexível, pois permite a você faça uma metaprogramação de como elas são aplicadas.

No exemplo a seguir, uma restrição escolhe uma ação com base em um *código de país* dos dados de rota. O [exemplo completo no GitHub](#).

```
public class CountrySpecificAttribute : Attribute, IActionConstraint
{
    private readonly string _countryCode;

    public CountrySpecificAttribute(string countryCode)
    {
        _countryCode = countryCode;
    }

    public int Order
    {
        get
        {
            return 0;
        }
    }

    public bool Accept(ActionConstraintContext context)
    {
        return string.Equals(
            context.RouteContext.RouteData.Values["country"].ToString(),
            _countryCode,
            StringComparison.OrdinalIgnoreCase);
    }
}
```

Você é responsável por implementar o método `Accept` e por escolher uma "ordem" na qual a restrição deve ser executada. Nesse caso, o método `Accept` retorna `true` para indicar que a ação é correspondente quando o valor de rota `country` é correspondente. Isso é diferente de um `RouteValueAttribute`, pois permite o fallback para uma ação não atribuída. O exemplo mostra que se você definir uma ação `en-US`, um código de país como `fr-FR` fará o fallback para um controlador mais genérico que não tem `[CountrySpecific(...)]` aplicado.

A propriedade `Order` decide de qual *estágio* a restrição faz parte. Restrições de ação são executadas em grupos com base no `Order`. Por exemplo, todos atributos de método HTTP fornecidos pela estrutura usam o mesmo valor de `Order` para que sejam executados no mesmo estágio. Você pode ter tantos estágios quantos forem necessários para implementar suas políticas desejadas.

### TIP

Para decidir o valor para `Order`, considere se sua restrição deve ou não deve ser aplicada antes de métodos HTTP. Números inferiores são executados primeiro.

# Uploads de arquivos no ASP.NET Core

30/10/2018 • 13 minutes to read • [Edit Online](#)

Por [Steve Smith](#)

Ações do ASP.NET MVC dão suporte ao upload de um ou mais arquivos usando model binding simples para arquivos menores ou streaming para arquivos maiores.

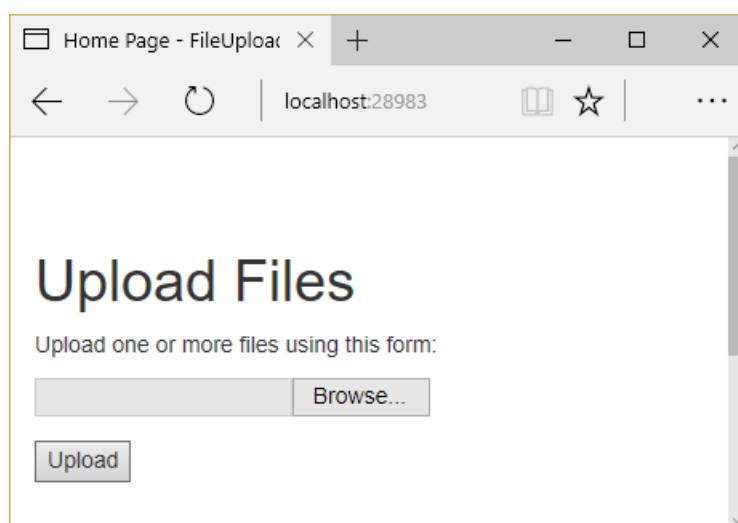
[Exibir ou baixar a amostra do GitHub](#)

## Upload de arquivos pequenos com o model binding

Para carregar arquivos pequenos, você pode usar um formulário HTML com várias partes ou construir uma solicitação POST usando JavaScript. Um formulário de exemplo usando Razor, que dá suporte a vários arquivos carregados, é mostrado abaixo:

```
<form method="post" enctype="multipart/form-data" asp-controller="UploadFiles" asp-action="Index">
    <div class="form-group">
        <div class="col-md-10">
            <p>Upload one or more files using this form:</p>
            <input type="file" name="files" multiple />
        </div>
    </div>
    <div class="form-group">
        <div class="col-md-10">
            <input type="submit" value="Upload" />
        </div>
    </div>
</form>
```

Para dar suporte a uploads de arquivos, os formulários HTML devem especificar um `enctype` igual a `multipart/form-data`. O elemento de entrada `files` mostrado acima dá suporte ao upload de vários arquivos. Omita o atributo `multiple` neste elemento de entrada para permitir o upload de apenas um arquivo. A marcação acima é renderizada em um navegador como:



Os arquivos individuais carregados no servidor podem ser acessados por meio do [Model binding](#) usando a interface `IFormFile`. `IFormFile` tem esta estrutura:

```
public interface IFormFile
{
    string ContentType { get; }
    string ContentDisposition { get; }
    IHeaderDictionary Headers { get; }
    long Length { get; }
    string Name { get; }
    string FileName { get; }
    Stream OpenReadStream();
    void CopyTo(Stream target);
    Task CopyToAsync(Stream target, CancellationToken cancellationToken = null);
}
```

## WARNING

Não dependa ou confie na propriedade `FileName` sem validação. A propriedade `FileName` deve ser usada somente para fins de exibição.

Ao fazer upload de arquivos usando model binding e a interface `IFormFile`, o método de ação pode aceitar um único `IFormFile` ou um `IEnumerable<IFormFile>` (ou `List<IFormFile>`) que representa vários arquivos. O exemplo a seguir executa um loop em um ou mais arquivos carregados, salva-os no sistema de arquivos local e retorna o número total e o tamanho dos arquivos carregados.

**Aviso:** o seguinte código usa `GetTempFileName`, que gerará um `IOException` se mais de 65.535 arquivos forem criados sem excluir os arquivos temporários anteriores. Um aplicativo real deve excluir arquivos temporários ou usar `GetTempPath` e `GetRandomFileName` para criar nomes de arquivo temporários. O limite de 65.535 arquivos é por servidor, então outro aplicativo no servidor poderá usar todos os 65.535 arquivos.

```
[HttpPost("UploadFiles")]
public async Task<IActionResult> Post(List<IFormFile> files)
{
    long size = files.Sum(f => f.Length);

    // full path to file in temp location
    var filePath = Path.GetTempFileName();

    foreach (var formFile in files)
    {
        if (formFile.Length > 0)
        {
            using (var stream = new FileStream(filePath, FileMode.Create))
            {
                await formFile.CopyToAsync(stream);
            }
        }
    }

    // process uploaded files
    // Don't rely on or trust the FileName property without validation.

    return Ok(new { count = files.Count, size, filePath });
}
```

Arquivos carregados usando a técnica `IFormFile` são armazenados em buffer na memória ou no disco no servidor Web antes de serem processados. Dentro do método de ação, o conteúdo de `IFormFile` podem ser acessado como um fluxo. Além do sistema de arquivos local, os arquivos podem ser transmitidos para o [Armazenamento de Blobs do Azure](#) ou para o [Entity Framework](#).

Para armazenar dados de arquivo binário em um banco de dados usando o Entity Framework, defina uma

propriedade do tipo `byte[]` na entidade:

```
public class ApplicationUser : IdentityUser
{
    public byte[] AvatarImage { get; set; }
}
```

Especifique uma propriedade de viewmodel do tipo `IFormFile`:

```
public class RegisterViewModel
{
    // other properties omitted

    public IFormFile AvatarImage { get; set; }
}
```

#### NOTE

`IFormFile` pode ser usado diretamente como um parâmetro de método de ação ou como uma propriedade de viewmodel, como mostrado acima.

Copie o `IFormFile` para um fluxo e salve-o na matriz de bytes:

```
// POST: /Account/Register
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Register(RegisterViewModel model)
{
    ViewData["ReturnUrl"] = returnUrl;
    if (ModelState.IsValid)
    {
        var user = new ApplicationUser {
            UserName = model.Email,
            Email = model.Email
        };
        using (var memoryStream = new MemoryStream())
        {
            await model.AvatarImage.CopyToAsync(memoryStream);
            user.AvatarImage = memoryStream.ToArray();
        }
        // additional logic omitted

        // Don't rely on or trust the model.AvatarImage.FileName property
        // without validation.
    }
}
```

#### NOTE

Tenha cuidado ao armazenar dados binários em bancos de dados relacionais, pois isso pode afetar negativamente o desempenho.

## Upload de arquivos grandes usando streaming

Se o tamanho ou a frequência dos uploads de arquivos estiver causando problemas de recursos para o aplicativo, considere transmitir o upload dos arquivos por streaming em vez de armazená-los completamente em buffer, como na abordagem de model binding mostrada acima. Embora o uso de `IFormFile` e do model binding seja uma

solução muito mais simples, o streaming requer a execução de algumas etapas para ser implementado corretamente.

#### NOTE

Qualquer arquivo armazenado em buffer que exceder 64KB será movido do RAM para um arquivo temporário em disco no servidor. Os recursos (disco, RAM) usados pelos uploads de arquivos dependem do número e do tamanho dos uploads de arquivos simultâneos. Streaming não é tanto uma questão de desempenho, e sim de escala. Se você tentar armazenar muitos uploads em buffer, seu site falhará quando ficar sem memória ou sem espaço em disco.

O exemplo a seguir demonstra como usar JavaScript/Angular para fazer o streaming para uma ação do controlador. O token antifalsificação do arquivo é gerado usando um atributo de filtro personalizado e passada nos cabeçalhos HTTP em vez do corpo da solicitação. Como um método de ação processa os dados carregados diretamente, o model binding é desabilitado por outro filtro. Dentro da ação, o conteúdo do formulário é lido usando um `MultipartReader`, que lê cada `MultipartSection` individual, processando o arquivo ou armazenando o conteúdo conforme apropriado. Após todas as seções serem lidas, a ação executa seu próprio model binding.

A ação inicial carrega o formulário e salva um token antifalsificação em um cookie (por meio do atributo `GenerateAntiforgeryTokenCookieForAjax`):

```
[HttpGet]
[GenerateAntiforgeryTokenCookieForAjax]
public IActionResult Index()
{
    return View();
}
```

O atributo usa o suporte interno [Antifalsificação](#) do ASP.NET Core para definir um cookie com um token de solicitação:

```
public class GenerateAntiforgeryTokenCookieForAjaxAttribute : ActionFilterAttribute
{
    public override void OnActionExecuted(ActionExecutedContext context)
    {
        var antiforgery = context.HttpContext.RequestServices.GetService<IAntiforgery>();

        // We can send the request token as a JavaScript-readable cookie,
        // and Angular will use it by default.
        var tokens = antiforgery.GetAndStoreTokens(context.HttpContext);
        context.HttpContext.Response.Cookies.Append(
            "XSRF-TOKEN",
            tokens.RequestToken,
            new CookieOptions() { HttpOnly = false });
    }
}
```

O Angular passa automaticamente um token antifalsificação em um cabeçalho de solicitação chamado `X-XSRF-TOKEN`. O aplicativo ASP.NET Core MVC é configurado para se referir a esse cabeçalho em sua configuração em `Startup.cs`:

```
public void ConfigureServices(IServiceCollection services)
{
    // Angular's default header name for sending the XSRF token.
    services.AddAntiforgery(options => options.HeaderName = "X-XSRF-TOKEN");

    services.AddMvc();
}
```

O atributo `DisableFormValueModelBinding`, mostrado abaixo, é usado para desabilitar o model binding para o método de ação `Upload`.

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method)]
public class DisableFormValueModelBindingAttribute : Attribute, IResourceFilter
{
    public void OnResourceExecuting(ResourceExecutingContext context)
    {
        var factories = context.ValueProviderFactories;
        factories.RemoveType<FormValueProviderFactory>();
        factories.RemoveType<JQueryFormValueProviderFactory>();
    }

    public void OnResourceExecuted(ResourceExecutedContext context)
    {
    }
}
```

Como o model binding é desabilitado, o método de ação `Upload` não aceita parâmetros. Ele trabalha diretamente com a propriedade `Request` de `ControllerBase`. Um `MultipartReader` é usado para ler cada seção. O arquivo é salvo com um nome de arquivo GUID e os dados de chave/valor são armazenados em um `KeyValueAccumulator`. Após todas as seções terem sido lidas, o conteúdo do `KeyValueAccumulator` é usado para associar os dados do formulário a um tipo de modelo.

O método `Upload` completo é mostrado abaixo:

**Aviso:** o seguinte código usa `GetTempFileName`, que gerará um `IOException` se mais de 65.535 arquivos forem criados sem excluir os arquivos temporários anteriores. Um aplicativo real deve excluir arquivos temporários ou usar `GetTempPath` e `GetRandomFileName` para criar nomes de arquivo temporários. O limite de 65.535 arquivos é por servidor, então outro aplicativo no servidor poderá usar todos os 65.535 arquivos.

```
// 1. Disable the form value model binding here to take control of handling
//      potentially large files.
// 2. Typically antiforgery tokens are sent in request body, but since we
//      do not want to read the request body early, the tokens are made to be
//      sent via headers. The antiforgery token filter first looks for tokens
//      in the request header and then falls back to reading the body.
[HttpPost]
[DisableFormValueModelBinding]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Upload()
{
    if (!MultipartRequestHelper.IsMultipartContentType(Request.ContentType))
    {
        return BadRequest($"Expected a multipart request, but got {Request.ContentType}");
    }

    // Used to accumulate all the form url encoded key value pairs in the
    // request.
    var formAccumulator = new KeyValueAccumulator();
    string targetFilePath = null;

    var boundary = MultipartRequestHelper.GetBoundary(
        MediaTypeHeaderValue.Parse(Request.ContentType),
        _defaultFormOptions.MultipartBoundaryLengthLimit);
    var reader = new MultipartReader(boundary, HttpContext.Request.Body);

    var section = await reader.ReadNextSectionAsync();
    while (section != null)
    {
        ContentDispositionHeaderValue contentDisposition;
        var hasContentDispositionHeader = ContentDispositionHeaderValue.TryParse(section.ContentDisposition,
            out contentDisposition);
```

```

if (hasContentDispositionHeader)
{
    if (MultipartRequestHelper.HasFileContentDisposition(contentDisposition))
    {
        targetFilePath = Path.GetTempFileName();
        using (var targetStream = System.IO.File.Create(targetFilePath))
        {
            await section.Body.CopyToAsync(targetStream);

            _logger.LogInformation($"Copied the uploaded file '{targetFilePath}'");
        }
    }
    else if (MultipartRequestHelper.HasFormDataContentDisposition(contentDisposition))
    {
        // Content-Disposition: form-data; name="key"
        //
        // value

        // Do not limit the key name length here because the
        // multipart headers length limit is already in effect.
        var key = HeaderUtilities.RemoveQuotes(contentDisposition.Name);
        var encoding = GetEncoding(section);
        using (var streamReader = new StreamReader(
            section.Body,
            encoding,
            detectEncodingFromByteOrderMarks: true,
            bufferSize: 1024,
            leaveOpen: true))
        {
            // The value length limit is enforced by MultipartBodyLengthLimit
            var value = await streamReader.ReadToEndAsync();
            if (String.Equals(value, "undefined", StringComparison.OrdinalIgnoreCase))
            {
                value = String.Empty;
            }
            formAccumulator.Append(key, value);

            if (formAccumulator.ValueCount > _defaultFormOptions.ValueCountLimit)
            {
                throw new InvalidDataException($"Form key count limit
{_defaultFormOptions.ValueCountLimit} exceeded.");
            }
        }
    }
}

// Drains any remaining section body that has not been consumed and
// reads the headers for the next section.
section = await reader.ReadNextSectionAsync();
}

// Bind form data to a model
var user = new User();
var formValueProvider = new FormValueProvider(
    BindingSource.Form,
    new FormCollection(formAccumulator.GetResults()),
    CultureInfo.CurrentCulture);

var bindingSuccessful = await TryUpdateModelAsync(user, prefix: "",
    valueProvider: formValueProvider);
if (!bindingSuccessful)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
}

```

```
var uploadedData = new UploadedData()
{
    Name = user.Name,
    Age = user.Age,
    Zipcode = user.Zipcode,
    FilePath = targetFilePath
};
return Json(uploadedData);
}
```

## Solução de problemas

Abaixo, são listados alguns problemas comuns encontrados ao trabalhar com o upload de arquivos e suas possíveis soluções.

### Erro não encontrado inesperado com o IIS

O erro a seguir indica que o upload do arquivo excede o `maxAllowedContentLength` configurado do servidor:

```
HTTP 404.13 - Not Found
The request filtering module is configured to deny a request that exceeds the request content length.
```

A configuração padrão é `30000000`, que é aproximadamente 28,6 MB. O valor pode ser personalizado editando `web.config`:

```
<system.webServer>
  <security>
    <requestFiltering>
      <!-- This will handle requests up to 50MB -->
      <requestLimits maxAllowedContentLength="52428800" />
    </requestFiltering>
  </security>
</system.webServer>
```

Essa configuração só se aplica ao IIS. Esse comportamento não ocorre por padrão quando a hospedagem é feita no Kestrel. Para obter mais informações, consulte [Limites de solicitação <requestLimits>](#).

### Exceção de referência nula com IFormFile

Se o controlador estiver aceitando arquivos carregados usando `IFormFile`, mas você observar que o valor sempre é nulo, confirme que seu formulário HTML está especificando um valor de `enctype` igual a `multipart/form-data`.

Se esse atributo não estiver definido no elemento `<form>`, o upload do arquivo não ocorrerá e os argumentos `IFormFile` associados serão nulos.

# Injeção de dependência em controladores no ASP.NET Core

30/10/2018 • 9 minutes to read • [Edit Online](#)

Por [Steve Smith](#)

Controladores do ASP.NET Core MVC devem solicitar suas dependências explicitamente por meio de seus construtores. Em algumas instâncias, ações individuais do controlador podem exigir um serviço e pode não fazer sentido solicitá-lo no nível do controlador. Nesse caso, você também pode optar por injetar um serviço como um parâmetro no método de ação.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Injeção de dependência

A injeção de dependência é uma técnica que segue o [Princípio de inversão de dependência](#), permitindo que aplicativos sejam compostos por módulos acoplados de forma flexível. O ASP.NET Core tem suporte interno para a [injeção de dependência](#), o que facilita a manutenção e o teste de aplicativos.

## Injeção de construtor

O suporte interno do ASP.NET Core para injeção de dependência baseada no construtor se estende para controladores MVC. Simplesmente adicionando um tipo de serviço ao seu controlador como um parâmetro de construtor, o ASP.NET Core tentará resolver o tipo usando seu contêiner de serviço interno. Os serviços são normalmente, mas não sempre, definidos usando interfaces. Por exemplo, se seu aplicativo tiver lógica de negócios que depende da hora atual, você pode injetar um serviço que recupera a hora (em vez fazer o hard-coding), o que permite que seus testes sejam passados em implementações que usam uma hora definida.

```
using System;

namespace ControllerDI.Interfaces
{
    public interface IDateTime
    {
        DateTime Now { get; }
    }
}
```

Implementar uma interface como esta para que ele use o relógio do sistema em tempo de execução é simples:

```

using System;
using ControllerDI.Interfaces;

namespace ControllerDI.Services
{
    public class SystemDateTime : IDateTime
    {
        public DateTime Now
        {
            get { return DateTime.Now; }
        }
    }
}

```

Com isso em vigor, podemos usar o serviço em nosso controlador. Nesse caso, adicionamos alguma lógica para ao método `HomeController` `Index` para exibir uma saudação ao usuário com base na hora do dia.

```

using ControllerDI.Interfaces;
using Microsoft.AspNetCore.Mvc;

namespace ControllerDI.Controllers
{
    public class HomeController : Controller
    {
        private readonly IDateTime _dateTime;

        public HomeController(IDateTime dateTime)
        {
            _dateTime = dateTime;
        }

        public IActionResult Index()
        {
            var serverTime = _dateTime.Now;
            if (serverTime.Hour < 12)
            {
                ViewData["Message"] = "It's morning here - Good Morning!";
            }
            else if (serverTime.Hour < 17)
            {
                ViewData["Message"] = "It's afternoon here - Good Afternoon!";
            }
            else
            {
                ViewData["Message"] = "It's evening here - Good Evening!";
            }
            return View();
        }
    }
}

```

Se executarmos o aplicativo agora, provavelmente encontraremos um erro:

```

An unhandled exception occurred while processing the request.

InvalidOperationException: Unable to resolve service for type 'ControllerDI.Interfaces.IDateTime' while
attempting to activate 'ControllerDI.Controllers.HomeController'.
Microsoft.Extensions.DependencyInjection.ActivatorUtilities.GetService(IServiceProvider sp, Type type, Type
requiredBy, Boolean isDefaultParameterRequired)

```

Esse erro ocorre quando não configuramos um serviço no método `ConfigureServices` em nossa classe `Startup`. Para especificar que solicitações de `IDateTime` devem ser resolvidas usando uma instância de `SystemDateTime`,

adicone a linha realçada à lista abaixo para seu método `ConfigureServices`:

```
public void ConfigureServices(IServiceCollection services)
{
    // Add application services.
    services.AddTransient<IDateTime, SystemDateTime>();
}
```

#### NOTE

Esse serviço específico poderia ser implementado usando qualquer uma das várias opções diferentes de tempo de vida (`Transient`, `Scoped` ou `Singleton`). Consulte [Injeção de dependência](#) para entender como cada uma dessas opções de escopo afetará o comportamento de seu serviço.

Depois que o serviço tiver sido configurado, executar o aplicativo e navegar para a home page deve exibir a mensagem baseada em hora conforme o esperado:



#### A Message From The Server

It's afternoon here - Good Afternoon!

#### TIP

Confira [Testar a lógica do controlador](#) para saber como solicitar dependências explicitamente <http://deviq.com/explicit-dependencies-principle/> em controladores e facilitar o teste do código.

A injeção de dependência interna do ASP.NET Core dá suporte a apenas um construtor para classes que solicitam serviços. Se tiver mais de um construtor, você poderá receber uma exceção informando:

```
An unhandled exception occurred while processing the request.

InvalidOperationException: Multiple constructors accepting all given argument types have been found in type 'ControllerDI.Controllers.HomeController'. There should only be one applicable constructor.
Microsoft.Extensions.DependencyInjection.ActivatorUtilities.FindApplicableConstructor(Type instanceType,
Type[] argumentTypes, ConstructorInfo& matchingConstructor, Nullable`1[] parameterMap)
```

Como a mensagem de erro afirma, você pode corrigir esse problema usando um único construtor. Você também pode [substituir o contêiner de injeção de dependência padrão por uma implementação de terceiros](#), muitas das quais dão suporte a vários construtores.

## Injeção de ação com `FromServices`

Às vezes, você não precisa de um serviço para mais de uma ação em seu controlador. Nesse caso, talvez faça sentido injetar o serviço como um parâmetro do método de ação. Isso é feito marcando o parâmetro com o atributo `[FromServices]`, conforme mostrado aqui:

```

public IActionResult About([FromServices] IDateTime dateTime)
{
    ViewData["Message"] = "Currently on the server the time is " + dateTime.Now;

    return View();
}

```

## Acessando configurações de um controlador

Acessar definições de configuração ou do aplicativo de dentro de um controlador é um padrão comum. Esse acesso deve usar o padrão Opções descrito na [configuração](#). Geralmente, você não deve solicitar configurações diretamente de seu controlador usando a injeção de dependência. Uma abordagem melhor é solicitar uma instância de `IOptions<T>`, em que `T` é a classe de configuração de que você precisa.

Para trabalhar com o padrão de opções, você precisa criar uma classe que representa as opções, como esta:

```

namespace ControllerDI.Model
{
    public class SampleWebSettings
    {
        public string Title { get; set; }
        public int Updates { get; set; }
    }
}

```

Em seguida, você precisa configurar o aplicativo para usar o modelo de opções e adicionar sua classe de configuração à coleção de serviços em `ConfigureServices`:

```

public Startup(IHostingEnvironment env)
{
    var builder = new ConfigurationBuilder()
        .SetBasePath(env.ContentRootPath)
        .AddJsonFile("samplewebsettings.json");
    Configuration = builder.Build();
}

public IConfigurationRoot Configuration { get; set; }

// This method gets called by the runtime. Use this method to add services to the container.
// For more information on how to configure your application, visit http://go.microsoft.com/fwlink/?LinkID=398940
public void ConfigureServices(IServiceCollection services)
{
    // Required to use the Options<T> pattern
    services.AddOptions();

    // Add settings from configuration
    services.Configure<SampleWebSettings>(Configuration);

    // Uncomment to add settings from code
    //services.Configure<SampleWebSettings>(settings =>
    //{
    //    settings.Updates = 17;
    //});

    services.AddMvc();

    // Add application services.
    services.AddTransient<IDateTime, SystemDateTime>();
}

```

#### NOTE

Na lista acima, estamos configurando o aplicativo para ler as configurações de um arquivo no formato JSON. Você também pode definir as configurações inteiramente no código, como é mostrado no código comentado acima. Consulte [Configuração](#) para ver outras opções de configuração.

Após ter especificado um objeto de configuração fortemente tipado (nesse caso, `SampleWebSettings`) e tê-lo adicionado à coleção de serviços, você pode solicitá-lo de qualquer método de Ação ou Controlador solicitando uma instância de `IOptions<T>` (nesse caso, `IOptions<SampleWebSettings>`). O código a seguir mostra como alguém solicitaria as configurações de um controlador:

```
public class SettingsController : Controller
{
    private readonly SampleWebSettings _settings;

    public SettingsController(IOptions<SampleWebSettings> settingsOptions)
    {
        _settings = settingsOptions.Value;
    }

    public IActionResult Index()
    {
        ViewData["Title"] = _settings.Title;
        ViewData["Updates"] = _settings.Updates;
        return View();
    }
}
```

Seguir o padrão Opções permite que as definições e configurações sejam dissociadas umas das outras e garante que o controlador esteja seguindo a [separação de preocupações](#), uma vez que ele não precisa saber como nem onde encontrar as informações de configuração. Isso facilita a realização de teste de unidade no controlador usando a [Lógica do controlador de teste](#), porque não há nenhuma [adesão estática](#) ou criação de instância direta das classes de configuração dentro da classe do controlador.

# Injeção de dependência em exibições no ASP.NET Core

30/10/2018 • 7 minutes to read • [Edit Online](#)

Por [Steve Smith](#)

O ASP.NET Core dá suporte à [injeção de dependência](#) em exibições. Isso pode ser útil para serviços específicos a uma exibição, como localização ou dados necessários apenas para o preenchimento de elementos de exibição. Você deve tentar manter a [separação de interesses](#) entre os controladores e as exibições. A maioria dos dados exibida pelas exibições deve ser passada pelo controlador.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Um exemplo simples

Injete um serviço em uma exibição usando a diretiva `@inject`. Considere `@inject` como a adição de uma propriedade à exibição e o preenchimento da propriedade usando a DI.

A sintaxe de `@inject : @inject <type> <name>`

Um exemplo de `@inject` em ação:

```

@using System.Threading.Tasks
@using ViewInjectSample.Model
@using ViewInjectSample.Model.Services
@model IEnumerable<ToDoItem>
@inject StatisticsService StatsService
<!DOCTYPE html>
<html>
<head>
    <title>To Do Items</title>
</head>
<body>
    <div>
        <h1>To Do Items</h1>
        <ul>
            <li>Total Items: @StatsService.GetCount()</li>
            <li>Completed: @StatsService.GetCompletedCount()</li>
            <li>Avg. Priority: @StatsService.GetAveragePriority()</li>
        </ul>
        <table>
            <tr>
                <th>Name</th>
                <th>Priority</th>
                <th>Is Done?</th>
            </tr>
            @foreach (var item in Model)
            {
                <tr>
                    <td>@item.Name</td>
                    <td>@item.Priority</td>
                    <td>@item.IsDone</td>
                </tr>
            }
        </table>
    </div>
</body>
</html>

```

Essa exibição exibe uma lista de instâncias `ToDoItem`, junto com um resumo mostrando estatísticas gerais. O resumo é populado com base no `StatisticsService` injetado. Esse serviço é registrado para injeção de dependência em `ConfigureServices` em `Startup.cs`:

```

// For more information on how to configure your application, visit http://go.microsoft.com/fwlink/?LinkID=398940
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    services.AddTransient<IToDoItemRepository, ToDoItemRepository>();
    services.AddTransient<StatisticsService>();
    services.AddTransient<ProfileOptionsService>();
}

```

O `StatisticsService` faz alguns cálculos no conjunto de instâncias `ToDoItem`, que ele acessa por meio de um repositório:

```

using System.Linq;
using ViewInjectSample.Interfaces;

namespace ViewInjectSample.Model.Services
{
    public class StatisticsService
    {
        private readonly IToDoItemRepository _toDoItemRepository;

        public StatisticsService(IToDoItemRepository toDoItemRepository)
        {
            _toDoItemRepository = toDoItemRepository;
        }

        public int GetCount()
        {
            return _toDoItemRepository.List().Count();
        }

        public int GetCompletedCount()
        {
            return _toDoItemRepository.List().Count(x => x.IsDone);
        }

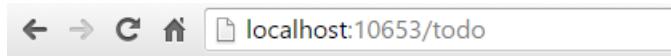
        public double GetAveragePriority()
        {
            if (_toDoItemRepository.List().Count() == 0)
            {
                return 0.0;
            }

            return _toDoItemRepository.List().Average(x => x.Priority);
        }
    }
}

```

O repositório de exemplo usa uma coleção em memória. A implementação mostrada acima (que opera em todos os dados na memória) não é recomendada para conjuntos de dados grandes e acessados remotamente.

A amostra exibe dados do modelo associado à exibição e o serviço injetado na exibição:



## To Do Items

- Total Items: 50
- Completed: 17
- Avg. Priority: 3

### Name Priority Is Done?

Task 1	1	True
Task 2	2	False
Task 3	3	False
Task 4	4	True
Task 5	5	False

## Populando os dados de pesquisa

A injeção de exibição pode ser útil para popular opções em elementos de interface do usuário, como listas suspensas. Considere um formulário de perfil do usuário que inclui opções para especificar o gênero, estado e outras preferências. A renderização desse formulário usando uma abordagem MVC padrão exigirá que o

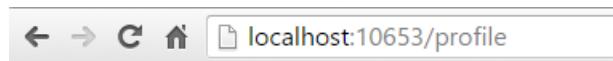
controlador solicite serviços de acesso a dados para cada um desses conjuntos de opções e, em seguida, popule um modelo ou `ViewBag` com cada conjunto de opções a ser associado.

Uma abordagem alternativa injeta os serviços diretamente na exibição para obter as opções. Isso minimiza a quantidade de código necessário para o controlador, movendo essa lógica de construção do elemento de exibição para a própria exibição. A ação do controlador para exibir um formulário de edição de perfil precisa apenas passar a instância de perfil para o formulário:

```
using Microsoft.AspNetCore.Mvc;
using ViewInjectSample.Model;

namespace ViewInjectSample.Controllers
{
    public class ProfileController : Controller
    {
        [Route("Profile")]
        public IActionResult Index()
        {
            // TODO: look up profile based on logged-in user
            var profile = new Profile()
            {
                Name = "Steve",
                FavColor = "Blue",
                Gender = "Male",
                State = new State("Ohio", "OH")
            };
            return View(profile);
        }
    }
}
```

O formulário HTML usado para atualizar essas preferências inclui listas suspensas para três das propriedades:



## Update Profile

Name:

Gender:

State:

Fav. Color:

Essas listas são populadas por um serviço que foi injetado na exibição:

```

@using System.Threading.Tasks
@using ViewInjectSample.Model.Services
@model ViewInjectSample.Model.Profile
@inject ProfileOptionsService Options
<!DOCTYPE html>
<html>
<head>
    <title>Update Profile</title>
</head>
<body>
<div>
    <h1>Update Profile</h1>
    Name: @Html.TextBoxFor(m => m.Name)
    <br/>
    Gender: @Html.DropDownList("Gender",
        Options.ListGenders().Select(g =>
            new SelectListItem() { Text = g, Value = g }))
    <br/>

    State: @Html.DropDownListFor(m => m.State.Code,
        Options.ListStates().Select(s =>
            new SelectListItem() { Text = s.Name, Value = s.Code}))
    <br />

    Fav. Color: @Html.DropDownList("FavColor",
        Options.ListColors().Select(c =>
            new SelectListItem() { Text = c, Value = c }))
</div>
</body>
</html>

```

O `ProfileOptionsService` é um serviço no nível da interface do usuário criado para fornecer apenas os dados necessários para esse formulário:

```

using System.Collections.Generic;

namespace ViewInjectSample.Model.Services
{
    public class ProfileOptionsService
    {
        public List<string> ListGenders()
        {
            // keeping this simple
            return new List<string>() {"Female", "Male"};
        }

        public List<State> ListStates()
        {
            // a few states from USA
            return new List<State>()
            {
                new State("Alabama", "AL"),
                new State("Alaska", "AK"),
                new State("Ohio", "OH")
            };
        }

        public List<string> ListColors()
        {
            return new List<string>() { "Blue", "Green", "Red", "Yellow" };
        }
    }
}

```

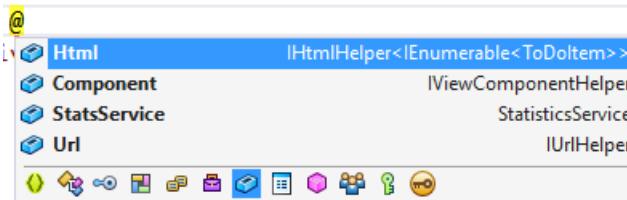
## IMPORTANT

Não se esqueça de registrar tipos que você solicita por meio de injeção de dependência no `Startup.ConfigureServices`.

Um tipo não registrado gera uma exceção em tempo de execução porque o provedor de serviços é consultado internamente por meio de `GetRequiredService`.

## Substituindo serviços

Além de injetar novos serviços, essa técnica também pode ser usada para substituir serviços injetados anteriormente em uma página. A figura abaixo mostra todos os campos disponíveis na página usada no primeiro exemplo:



Como você pode ver, os campos padrão incluem `Html`, `Component` e `Url` (bem como o `StatsService` que injetamos). Se, para a instância, você desejava substituir os Auxiliares HTML padrão por seus próprios, faça isso com facilidade usando `@inject`:

```
@using System.Threading.Tasks
@using ViewInjectSample.Helpers
@inject MyHtmlHelper Html
<!DOCTYPE html>
<html>
<head>
    <title>My Helper</title>
</head>
<body>
    <div>
        Test: @Html.Value
    </div>
</body>
</html>
```

Se deseja estender os serviços existentes, basta usar essa técnica herdando da implementação existente ou encapsulando-a com sua própria implementação.

## Consulte também

- Blog de Simon Timms: [Getting Lookup Data Into Your View](#) (Inserindo dados de pesquisa na exibição)

# Lógica do controlador de teste no ASP.NET Core

10/11/2018 • 19 minutes to read • [Edit Online](#)

Por [Steve Smith](#)

[Controladores](#) desempenham um papel central em qualquer aplicativo ASP.NET Core MVC. Assim, você precisa estar confiante de que os controladores se comportarão conforme o esperado. Testes automatizados podem detectar erros antes do aplicativo ser implantado em um ambiente de produção.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Testes de unidade da lógica do controlador

Os [testes de unidade](#) envolvem o teste de uma parte de um aplicativo isoladamente em relação à sua infraestrutura e às suas dependências. Quando a unidade está testando a lógica do controlador, somente o conteúdo de uma única ação é testada, não o comportamento de suas dependências ou da estrutura em si.

Configure testes de unidade de ações do controlador para se concentrarem no comportamento do controlador. Um teste de unidade do controlador evita cenários como [filtros](#), [roteamento](#) ou [model binding](#). Os testes que abrangem as interações entre os componentes que respondem coletivamente a uma solicitação são manipulados pelos [testes de integração](#). Para obter mais informações sobre os testes de integração, consulte [Testes de integração no ASP.NET Core](#).

Se você estiver escrevendo filtros e rotas personalizados, realize testes de unidade neles de forma isolada, não como parte dos testes em uma ação do controlador específica.

Para demonstrar testes de unidade do controlador, examine o controlador a seguir no aplicativo de exemplo. O controlador Home exibe uma lista de sessões de debate e permite que novas sessões sejam criadas com uma solicitação POST:

```

public class HomeController : Controller
{
    private readonly IBrainstormSessionRepository _sessionRepository;

    public HomeController(IBrainstormSessionRepository sessionRepository)
    {
        _sessionRepository = sessionRepository;
    }

    public async Task<IActionResult> Index()
    {
        var sessionList = await _sessionRepository.ListAsync();

        var model = sessionList.Select(session => new StormSessionViewModel()
        {
            Id = session.Id,
            DateCreated = session.DateCreated,
            Name = session.Name,
            IdeaCount = session.Ideas.Count
        });

        return View(model);
    }

    public class NewSessionModel
    {
        [Required]
        public string SessionName { get; set; }
    }

    [HttpPost]
    public async Task<IActionResult> Index(NewSessionModel model)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }
        else
        {
            await _sessionRepository.AddAsync(new BrainstormSession()
            {
                DateCreated = DateTimeOffset.Now,
                Name = model.SessionName
            });
        }

        return RedirectToAction(actionName: nameof(Index));
    }
}

```

O controlador anterior:

- Segue o [Princípio de Dependências Explícitas](#).
- Espera [DI \(injeção de dependência\)](#) para fornecer uma instância de `IBrainstormSessionRepository`.
- Pode ser testado com um serviço `IBrainstormSessionRepository` fictício usando uma estrutura de objeto fictício, como [Moq](#). Um *objeto fictício* é um objeto fabricado com um conjunto predeterminado de comportamentos de propriedade e de método usado para teste. Para saber mais, consulte [Introdução aos testes de integração](#).

O método `HTTP GET Index` não tem nenhum loop ou branch e chama apenas um método. O teste de unidade para esta ação:

- Imita o serviço `IBrainstormSessionRepository` usando o método `GetTestSessions`. `GetTestSessions` cria duas sessões de debate fictícias com datas e nomes de sessão.

- Executa o método `Index`.
- Faz declarações sobre o resultado retornado pelo método:
  - Um `ViewResult` é retornado.
  - O `ViewDataDictionary.Model` é um `StormSessionViewModel`.
  - Há duas sessões de debate armazenadas no `ViewDataDictionary.Model`.

```
[Fact]
public async Task Index_ReturnsAViewResult_WithAListOfBrainstormSessions()
{
    // Arrange
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.ListAsync())
        .ReturnsAsync(GetTestSessions());
    var controller = new HomeController(mockRepo.Object);

    // Act
    var result = await controller.Index();

    // Assert
    var viewResult = Assert.IsType<ViewResult>(result);
    var model = Assert.IsAssignableFrom<IEnumerable<StormSessionViewModel>>(
        viewResult.ViewData.Model);
    Assert.Equal(2, model.Count());
}
```

```
private List<BrainstormSession> GetTestSessions()
{
    var sessions = new List<BrainstormSession>();
    sessions.Add(new BrainstormSession()
    {
        DateCreated = new DateTime(2016, 7, 2),
        Id = 1,
        Name = "Test One"
    });
    sessions.Add(new BrainstormSession()
    {
        DateCreated = new DateTime(2016, 7, 1),
        Id = 2,
        Name = "Test Two"
    });
    return sessions;
}
```

Os testes método `HTTP POST Index` do controlador Home verificam se:

- Quando `ModelState.IsValid` é `false`, o método de ação retorna uma *400 Solicitação inválida* `ViewResult` com os dados apropriados.
- Quando `ModelState.IsValid` é `true`:
  - O método `Add` no repositório é chamado.
  - Um `RedirectToActionResult` é retornado com os argumentos corretos.

O estado de modelo inválido é testado por meio da adição de erros usando `AddModelError`, conforme mostrado no primeiro teste abaixo:

```

[Fact]
public async Task IndexPost_ReturnsBadRequestResult_WhenModelStateIsInvalid()
{
    // Arrange
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.ListAsync())
        .ReturnsAsync(GetTestSessions());
    var controller = new HomeController(mockRepo.Object);
    controller.ModelState.AddModelError("SessionName", "Required");
    var newSession = new HomeController.NewSessionModel();

    // Act
    var result = await controller.Index(newSession);

    // Assert
    var badRequestResult = Assert.IsType<BadRequestObjectResult>(result);
    Assert.IsType<SerializableError>(badRequestResult.Value);
}

[Fact]
public async Task IndexPost_ReturnsARedirectAndAddsSession_WhenModelStateIsValid()
{
    // Arrange
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.AddAsync(It.IsAny<BrainstormSession>()))
        .Returns(Task.CompletedTask)
        .Verifiable();
    var controller = new HomeController(mockRepo.Object);
    var newSession = new HomeController.NewSessionModel()
    {
        SessionName = "Test Name"
    };

    // Act
    var result = await controller.Index(newSession);

    // Assert
    var redirectToActionResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Null(redirectToActionResult.ControllerName);
    Assert.Equal("Index", redirectToActionResult.ActionName);
    mockRepo.Verify();
}

```

Quando `ModelState` não for válido, o mesmo `ViewResult` será retornado para uma solicitação GET. O teste não tenta passar um modelo inválido. Passar um modelo inválido não é uma abordagem válida, visto que o model binding não está em execução (embora um [teste de integração](#) use model binding). Nesse caso, o model binding não está sendo testado. Esses testes de unidade estão testando apenas o código no método de ação.

O segundo teste verifica se, quando o `ModelState` é válido:

- Um novo `BrainstormSession` é adicionado (por meio do repositório).
- O método retorna um `RedirectToActionResult` com as propriedades esperadas.

Chamadas fictícias que não são chamadas são normalmente ignoradas, mas a chamada a `Verifiable` no final da chamada de instalação permite a validação fictícia no teste. Isso é realizado com a chamada a `mockRepo.Verify`, que não será aprovada no teste se o método esperado não tiver sido chamado.

#### NOTE

A biblioteca do Moq usada neste exemplo possibilita a combinação de simulações verificáveis ou "estritas" com simulações não verificáveis (também chamadas de simulações "flexíveis" ou stubs). Saiba mais sobre como [personalizar o comportamento de Simulação com o Moq](#).

`SessionController` no exemplo de aplicativo exibe informações relacionadas a uma sessão de debate específica. O controlador inclui lógica para lidar com valores `id` inválidos (há dois cenários `return` no exemplo a seguir para abordar esses cenários). A última instrução `return` retorna um novo `StormSessionViewModel` para a exibição (`Controllers/SessionController.cs`):

```
public class SessionController : Controller
{
    private readonly IBrainstormSessionRepository _sessionRepository;

    public SessionController(IBrainstormSessionRepository sessionRepository)
    {
        _sessionRepository = sessionRepository;
    }

    public async Task<IActionResult> Index(int? id)
    {
        if (!id.HasValue)
        {
            return RedirectToAction(actionName: nameof(Index),
                controllerName: "Home");
        }

        var session = await _sessionRepository.GetByIdAsync(id.Value);
        if (session == null)
        {
            return Content("Session not found.");
        }

        var viewModel = new StormSessionViewModel()
        {
            DateCreated = session.DateCreated,
            Name = session.Name,
            Id = session.Id
        };

        return View(viewModel);
    }
}
```

Os testes de unidade incluem um teste para cada cenário `return` na ação `Index` do controlador `Session`:

```

[Fact]
public async Task IndexReturnsARedirectToIndexHomeWhenIdIsNull()
{
    // Arrange
    var controller = new SessionController(sessionRepository: null);

    // Act
    var result = await controller.Index(id: null);

    // Assert
    var redirectToActionResult =
        Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Home", redirectToActionResult.ControllerName);
    Assert.Equal("Index", redirectToActionResult.ActionName);
}

[Fact]
public async Task IndexReturnsContentWithSessionNotFoundWhenSessionNotFound()
{
    // Arrange
    int testSessionId = 1;
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync((BrainstormSession)null);
    var controller = new SessionController(mockRepo.Object);

    // Act
    var result = await controller.Index(testSessionId);

    // Assert
    var contentResult = Assert.IsType<ContentResult>(result);
    Assert.Equal("Session not found.", contentResult.Content);
}

[Fact]
public async Task IndexReturnsViewResultWithStormSessionViewModel()
{
    // Arrange
    int testSessionId = 1;
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync(GetTestSessions().FirstOrDefault(
            s => s.Id == testSessionId));
    var controller = new SessionController(mockRepo.Object);

    // Act
    var result = await controller.Index(testSessionId);

    // Assert
    var viewResult = Assert.IsType<ViewResult>(result);
    var model = Assert.IsType<StormSessionViewModel>(
        viewResult.ViewData.Model);
    Assert.Equal("Test One", model.Name);
    Assert.Equal(2, model.DateCreated.Day);
    Assert.Equal(testSessionId, model.Id);
}

```

Mudando para o controlador Ideas, o aplicativo expõe a funcionalidade como uma API Web na rota `api/ideas`:

- Uma lista de ideias (`IdeasDTO`) associada com uma sessão de debate é retornada pelo método `ForSession`.
- O método `Create` adiciona novas ideias a uma sessão.

```

[HttpGet("forsession/{sessionId}")]
public async Task<IActionResult> ForSession(int sessionId)
{
    var session = await _sessionRepository.GetByIdAsync(sessionId);
    if (session == null)
    {
        return NotFound(sessionId);
    }

    var result = session.Ideas.Select(idea => new IdeaDTO()
    {
        Id = idea.Id,
        Name = idea.Name,
        Description = idea.Description,
        DateCreated = idea.DateCreated
    }).ToList();

    return Ok(result);
}

[HttpPost("create")]
public async Task<IActionResult> Create([FromBody]NewIdeaModel model)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var session = await _sessionRepository.GetByIdAsync(model.SessionId);
    if (session == null)
    {
        return NotFound(model.SessionId);
    }

    var idea = new Idea()
    {
        DateCreated = DateTimeOffset.Now,
        Description = model.Description,
        Name = model.Name
    };
    session.AddIdea(idea);

    await _sessionRepository.UpdateAsync(session);

    return Ok(session);
}

```

Evite retornar entidades de domínio de negócios diretamente por meio de chamadas à API. Entidades de domínio:

- Geralmente incluem mais dados do que o cliente necessita.
- Acopla desnecessariamente o modelo de domínio interno do aplicativo à API exposta publicamente.

É possível executar o mapeamento entre entidades de domínio e os tipos retornados ao cliente:

- Manualmente com um LINQ `Select`, como o aplicativo de exemplo usa. Para saber mais, consulte [LINQ \(Consulta Integrada à Linguagem\)](#).
- Automaticamente com uma biblioteca, como [AutoMapper](#).

Em seguida, o aplicativo de exemplo demonstra os testes de unidade para os métodos de API `Create` e `ForSession` do controlador Ideas.

O aplicativo de exemplo contém dois testes `ForSession`. O primeiro teste determina se `ForSession` retorna um `NotFoundObjectResult` (HTTP não encontrado) para uma sessão inválida:

```

[Fact]
public async Task ForSession_ReturnsHttpNotFound_ForInvalidSession()
{
    // Arrange
    int testSessionId = 123;
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync((BrainstormSession)null);
    var controller = new IdeasController(mockRepo.Object);

    // Act
    var result = await controller.ForSession(testSessionId);

    // Assert
    var notFoundObjectResult = Assert.IsType<NotFoundObjectResult>(result);
    Assert.Equal(testSessionId, notFoundObjectResult.Value);
}

```

O segundo teste `ForSession` determina se `ForSession` retorna uma lista de sessão (`<List<IdeaDTO>>`) para uma sessão válida. As verificações também examinam a primeira ideia para confirmar se sua propriedade `Name` está correta:

```

[Fact]
public async Task ForSession_ReturnsIdeasForSession()
{
    // Arrange
    int testSessionId = 123;
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync(GetTestSession());
    var controller = new IdeasController(mockRepo.Object);

    // Act
    var result = await controller.ForSession(testSessionId);

    // Assert
    var okResult = Assert.IsType<OkObjectResult>(result);
    var returnValue = Assert.IsType<List<IdeaDTO>>(okResult.Value);
    var idea = returnValue.FirstOrDefault();
    Assert.Equal("One", idea.Name);
}

```

Para testar o comportamento do método `Create` quando o `ModelState` é inválido, o aplicativo de exemplo adiciona um erro de modelo ao controlador como parte do teste. Não tente testar a validação de modelos ou o model binding em testes de unidade—teste apenas o comportamento do método de ação quando houver um `ModelState` inválido:

```

[Fact]
public async Task Create_ReturnsBadRequest_GivenInvalidModel()
{
    // Arrange & Act
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    var controller = new IdeasController(mockRepo.Object);
    controller.ModelState.AddModelError("error", "some error");

    // Act
    var result = await controller.Create(model: null);

    // Assert
    Assert.IsType<BadRequestObjectResult>(result);
}

```

O segundo teste de `Create` depende do repositório retornar `null`, portanto, o repositório fictício é configurado para retornar `null`. Não é necessário criar um banco de dados de teste (na memória ou de outro tipo) e construir uma consulta que retornará esse resultado. O teste pode ser realizado em uma única instrução, como mostrado no código de exemplo:

```
[Fact]
public async Task Create_ReturnsHttpNotFound_ForInvalidSession()
{
    // Arrange
    int testSessionId = 123;
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync((BrainstormSession)null);
    var controller = new IdeasController(mockRepo.Object);

    // Act
    var result = await controller.Create(new NewIdeaModel());

    // Assert
    Assert.IsType<NotFoundObjectResult>(result);
}
```

O terceiro teste `Create`, `Create_ReturnsNewlyCreatedIdeaForSession`, verifica se o método `UpdateAsync` do repositório é chamado. A simulação é chamada com `Verifiable` e o método `Verify` do repositório fictício é chamado para confirmar se o método verificável é executado. Não é responsabilidade do teste de unidade garantir que o método `UpdateAsync` salve os dados—isso pode ser realizado com um teste de integração.

```
[Fact]
public async Task Create_ReturnsNewlyCreatedIdeaForSession()
{
    // Arrange
    int testSessionId = 123;
    string testName = "test name";
    string testDescription = "test description";
    var testSession = GetTestSession();
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync(testSession);
    var controller = new IdeasController(mockRepo.Object);

    var newIdea = new NewIdeaModel()
    {
        Description = testDescription,
        Name = testName,
        SessionId = testSessionId
    };
    mockRepo.Setup(repo => repo.UpdateAsync(testSession))
        .Returns(Task.CompletedTask)
        .Verifiable();

    // Act
    var result = await controller.Create(newIdea);

    // Assert
    var okResult = Assert.IsType<OkObjectResult>(result);
    var returnSession = Assert.IsType<BrainstormSession>(okResult.Value);
    mockRepo.Verify();
    Assert.Equal(2, returnSession.Ideas.Count());
    Assert.Equal(testName, returnSession.Ideas.LastOrDefault().Name);
    Assert.Equal(testDescription, returnSession.Ideas.LastOrDefault().Description);
}
```

## Testar ActionResult<T>

No ASP.NET Core 2.1 ou posterior, o `ActionResult<T>` (`ActionResult< TValue >`) permite que você retorne um tipo derivado de `ActionResult` ou retorne um tipo específico.

O aplicativo de exemplo inclui um método que retorna um `List<IdeaDTO>` para uma sessão `id` determinada. Se a sessão `id` não existir, o controlador retornará `NotFound`:

```
[HttpGet("forsessionactionresult/{sessionId}")]
[ProducesResponseType(200)]
[ProducesResponseType(404)]
public async Task<ActionResult<List<IdeaDTO>>> ForSessionActionResult(int sessionId)
{
    var session = await _sessionRepository.GetByIdAsync(sessionId);

    if (session == null)
    {
        return NotFound(sessionId);
    }

    var result = session.Ideas.Select(idea => new IdeaDTO()
    {
        Id = idea.Id,
        Name = idea.Name,
        Description = idea.Description,
        DateCreated = idea.DateCreated
    }).ToList();

    return result;
}
```

Dois testes do controlador `ForSessionActionResult` estão incluídos no `ApiIdeasControllerTests`.

O primeiro teste confirma se o controlador retorna um `ActionResult`, mas não uma lista de ideias inexistente para uma sessão `id` inexistente:

- O tipo `ActionResult<List<IdeaDTO>>` é `ActionResult`.
- O `Result` é um `NotFoundObjectResult`.

```
[Fact]
public async Task ForSessionActionResult_ReturnsNotFoundObjectResultForNonexistentSession()
{
    // Arrange
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    var controller = new IdeasController(mockRepo.Object);
    var nonExistentSessionId = 999;

    // Act
    var result = await controller.ForSessionActionResult(nonExistentSessionId);

    // Assert
    var actionResult = Assert.IsType<ActionResult<List<IdeaDTO>>>(result);
    Assert.IsType<NotFoundObjectResult>(actionResult.Result);
}
```

Para uma sessão `id` válida, o segundo teste confirma se o método retorna:

- Um `ActionResult` com um tipo `List<IdeaDTO>`.
- O `ActionResult<T>.Value` é um tipo `List<IdeaDTO>`.
- O primeiro item na lista é uma ideia válida que corresponde à ideia armazenada na sessão fictícia (obtida

chamando `GetTestSession` ).

```
[Fact]
public async Task ForSessionActionResult_ReturnsIdeasForSession()
{
    // Arrange
    int testSessionId = 123;
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync(GetTestSession());
    var controller = new IdeasController(mockRepo.Object);

    // Act
    var result = await controller.ForSessionActionResult(testSessionId);

    // Assert
    var actionResult = Assert.IsType<ActionResult<List<IdeaDTO>>>(result);
    var returnValue = Assert.IsType<List<IdeaDTO>>(actionResult.Value);
    var idea = returnValue.FirstOrDefault();
    Assert.Equal("One", idea.Name);
}
```

O aplicativo de exemplo também inclui um método para criar um novo `Idea` para uma determinada sessão. O controlador retorna:

- `BadRequest` para um modelo inválido.
- `NotFound` se a sessão não existir.
- `CreatedAtAction` quando a sessão for atualizada com a nova ideia.

```
[HttpPost("createactionresult")]
[ProducesResponseType(201)]
[ProducesResponseType(400)]
[ProducesResponseType(404)]
public async Task<ActionResult<BrainstormSession>> CreateActionResult([FromBody]NewIdeaModel model)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var session = await _sessionRepository.GetByIdAsync(model.SessionId);

    if (session == null)
    {
        return NotFound(model.SessionId);
    }

    var idea = new Idea()
    {
        DateCreated = DateTimeOffset.Now,
        Description = model.Description,
        Name = model.Name
    };
    session.AddIdea(idea);

    await _sessionRepository.UpdateAsync(session);

    return CreatedAtAction(nameof(CreateActionResult), new { id = session.Id }, session);
}
```

Três testes `CreateActionResult` estão incluídos no `ApiIdeasControllerTests`.

O primeiro texto confirma que um `BadRequest` é retornado para um modelo inválido.

```

[Fact]
public async Task CreateActionResult_ReturnsBadRequest_GivenInvalidModel()
{
    // Arrange & Act
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    var controller = new IdeasController(mockRepo.Object);
    controller.ModelState.AddModelError("error", "some error");

    // Act
    var result = await controller.CreateActionResult(model: null);

    // Assert
    var actionResult = Assert.IsType<ActionResult<BrainstormSession>>(result);
    Assert.IsType<BadRequestObjectResult>(actionResult.Result);
}

```

O segundo teste verifica se um `NotFound` será retornado se a sessão não existir.

```

[Fact]
public async Task CreateActionResult_ReturnsNotFoundObjectResultForNonexistentSession()
{
    // Arrange
    var nonExistentSessionId = 999;
    string testName = "test name";
    string testDescription = "test description";
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    var controller = new IdeasController(mockRepo.Object);

    var newIdea = new NewIdeaModel()
    {
        Description = testDescription,
        Name = testName,
        SessionId = nonExistentSessionId
    };

    // Act
    var result = await controller.CreateActionResult(newIdea);

    // Assert
    var actionResult = Assert.IsType<ActionResult<BrainstormSession>>(result);
    Assert.IsType<NotFoundObjectResult>(actionResult.Result);
}

```

Para uma sessão `id` válida, o teste final confirmará se:

- O método retorna um `ActionResult` com um tipo `BrainstormSession`.
- O `ActionResult<T>.Result` é um `CreatedAtActionResult`. `CreatedAtActionResult` é semelhante à resposta `201 Criado` com um cabeçalho `Location`.
- O `ActionResult<T>.Value` é um tipo `BrainstormSession`.
- A chamada fictícia para atualizar a sessão, `UpdateAsync(testSession)`, foi invocada. A chamada de método `Verifiable` é verificada por meio da execução de `mockRepo.Verify()` nas declarações.
- Dois objetos `Idea` são retornados para a sessão.
- O último item (o `Idea` adicionado pela chamada fictícia a `UpdateAsync`) corresponde ao `newIdea` adicionado à sessão no teste.

```

[Fact]
public async Task CreateActionResult_ReturnsNewlyCreatedIdeaForSession()
{
    // Arrange
    int testSessionId = 123;
    string testName = "test name";
    string testDescription = "test description";
    var testSession = GetTestSession();
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync(testSession);
    var controller = new IdeasController(mockRepo.Object);

    var newIdea = new NewIdeaModel()
    {
        Description = testDescription,
        Name = testName,
        SessionId = testSessionId
    };
    mockRepo.Setup(repo => repo.UpdateAsync(testSession))
        .Returns(Task.CompletedTask)
        .Verifiable();

    // Act
    var result = await controller.CreateActionResult(newIdea);

    // Assert
    var actionResult = Assert.IsType<ActionResult<BrainstormSession>>(result);
    var createdAtIndexResult = Assert.IsType<CreatedAtActionResult>(actionResult.Result);
    var returnValue = Assert.IsType<BrainstormSession>(createdAtIndexResult.Value);
    mockRepo.Verify();
    Assert.Equal(2, returnValue.Ideas.Count());
    Assert.Equal(testName, returnValue.Ideas.LastOrDefault().Name);
    Assert.Equal(testDescription, returnValue.Ideas.LastOrDefault().Description);
}

```

## Recursos adicionais

- [Testes de integração no ASP.NET Core](#)
- [Crie e execute testes de unidade com o Visual Studio.](#)
- [Princípio de Dependências Explícitas](#)

# Introdução aos Componentes Razor

04/02/2019 • 13 minutes to read • [Edit Online](#)

Por [Steve Sanderson](#), [Daniel Roth](#) e [Luke Latham](#)

## NOTE

O Razor Components do ASP.NET Core é compatível com ASP.NET Core 3.0 ou posterior.

Blazor é uma estrutura da Web sem suporte e experimental que não deve ser usada para cargas de trabalho de produção no momento.

Os *Componentes Razor* do ASP.NET Core executam o lado do servidor no ASP.NET Core, enquanto o *Blazor* (Componentes Razor que executam o lado do cliente) é uma estrutura da Web experimental do .NET que usa C#/Razor e HTML executados no navegador com o WebAssembly. O Blazor fornece todos os benefícios de uma estrutura de interface do usuário da Web do lado do cliente usando o .NET no cliente.

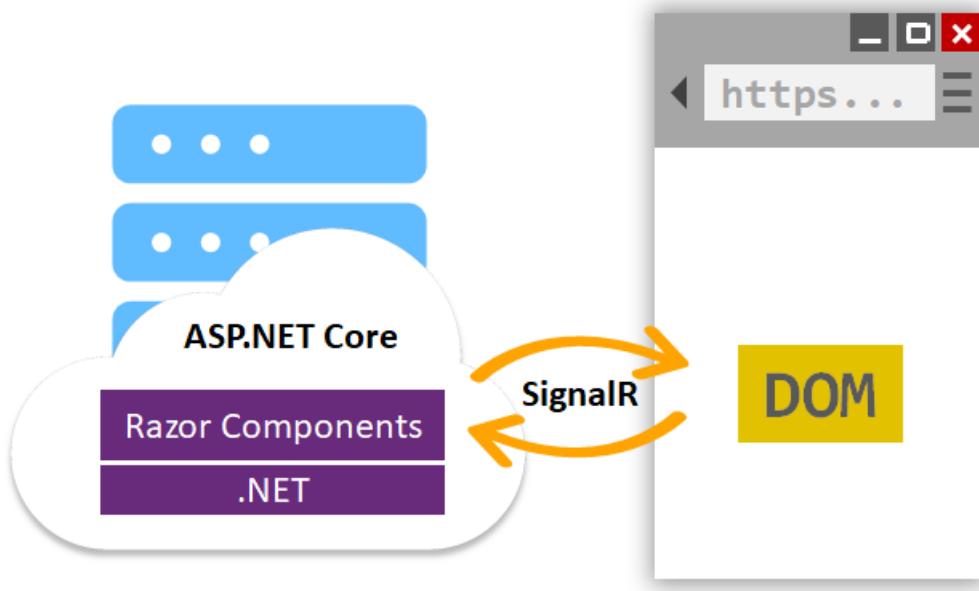
O desenvolvimento para a Web foi aprimorado de diversas maneiras ao longo dos anos, mas a criação de aplicativos Web modernos ainda apresenta desafios. O uso do .NET no navegador oferece muitas vantagens que podem ajudar a tornar o desenvolvimento para Web mais fácil e mais produtivo:

- **Estabilidade e consistência:** o .NET fornece estruturas de programação padronizadas entre plataformas, que são estáveis, completas e fáceis de usar.
- **Linguagens modernas inovadoras:** as linguagens do .NET estão melhorando constantemente com novos recursos de linguagem inovadores.
- **Ferramentas líderes do setor:** a família de produtos do Visual Studio fornece uma experiência fantástica de desenvolvimento no .NET entre plataformas, em Windows, Linux e macOS.
- **Velocidade e escalabilidade:** o .NET tem um forte histórico de desempenho, confiabilidade e segurança para o desenvolvimento de aplicativos. O uso do .NET como uma solução de pilha completa facilita a criação de aplicativos rápidos, confiáveis e seguros.
- **Desenvolvimento de pilha completa que aproveita as habilidades existentes:** os desenvolvedores C#/Razor usam as habilidades em C#/Razor que eles já têm para escrever o código do lado do cliente e compartilhar a lógica do lado do servidor e do lado do cliente entre os aplicativos.
- **Amplio suporte a navegadores:** os Componentes Razor renderizam a interface do usuário como uma marcação e um JavaScript comuns. O Blazor é executado no .NET, no navegador, usando padrões abertos da Web sem nenhum plug-in e nenhuma transpilação de código. O Blazor funciona em todos os navegadores da Web modernos, incluindo os navegadores móveis.

## Modelos de hospedagem

### Modelo de hospedagem do lado do servidor

Como os Componentes Razor desvinculam a lógica de renderização de um componente da maneira de aplicar as atualizações da interface do usuário, há flexibilidade na maneira de hospedar os Componentes Razor. Os Componentes Razor do ASP.NET Core no .NET Core 3.0 adicionam suporte para a hospedagem dos Componentes Razor no servidor, em um aplicativo ASP.NET Core, em que todas as atualizações da interface do usuário são realizadas por uma conexão SignalR. O tempo de execução realiza o envio de eventos da interface do usuário do navegador para o servidor e, depois de executar os componentes, aplica as atualizações na interface do usuário retornadas pelo servidor ao navegador. A mesma conexão também é usada para realizar as chamadas de interoperabilidade do JavaScript.



Para obter mais informações, consulte [Razor Components hosting models](#).

### Modelo de hospedagem do lado do cliente

A execução do código do .NET em navegadores da Web é possibilitada por uma tecnologia relativamente nova, o [WebAssembly](#) (abreviado como *wasm*). O WebAssembly é um padrão aberto da Web compatível com navegadores da Web sem plug-ins. O WebAssembly é um formato de código de bytes compacto, otimizado para download rápido e máxima velocidade de execução.

O código do WebAssembly pode acessar a funcionalidade completa do navegador por meio da interoperabilidade do JavaScript. Ao mesmo tempo, o código do WebAssembly é executado na mesma área restrita confiável que o JavaScript para impedir ações mal-intencionadas no computador cliente.

Quando um aplicativo Blazor é criado e executado em um navegador:

1. os arquivos C# e os arquivos Razor são compilados em assemblies do .NET.
2. os assemblies e o tempo de execução do .NET são baixados no navegador.
3. o Blazor usa o JavaScript para inicializar o tempo de execução do .NET e configura o tempo de execução para carregar as referências de assembly necessárias. A manipulação do DOM (Modelo de Objeto do Documento) e as chamadas à API do navegador são realizadas pelo tempo de execução do Blazor por meio da interoperabilidade do JavaScript.

Para dar suporte a navegadores mais antigos que não são compatíveis com o WebAssembly, você pode usar o [modelo de hospedagem do lado do servidor](#) dos Componentes Razor do ASP.NET Core.

Para obter mais informações, consulte [Razor Components hosting models](#).

## Componentes

Os aplicativos são criados com *componentes*. Um componente é uma parte da interface do usuário, como uma página, uma caixa de diálogo ou um formulário de entrada de dados. Os componentes podem ser aninhados, reutilizados e compartilhados entre os projetos.

Um *componente* é uma classe do .NET. A classe também pode ser escrita diretamente, como uma classe C# (\*.cs) ou, com mais frequência, na forma de uma página de marcação Razor (\*.cshtml).

O [Razor](#) é uma sintaxe para a combinação da marcação HTML com o código C#. O Razor foi projetado visando a produtividade do desenvolvedor, permitindo que ele mude entre a marcação e o C# no mesmo arquivo com o

suporte do [IntelliSense](#). A marcação a seguir é um exemplo de um componente básico de caixa de diálogo personalizada em um arquivo Razor (*DialogComponent.cshtml*):

```
<div>
    <h2>@Title</h2>
    @BodyContent
    <button onclick=@OnOK>OK</button>
</div>

@functions {
    public string Title { get; set; }
    public RenderFragment BodyContent { get; set; }
    public Action OnOK { get; set; }
}
```

Quando esse componente é usado em outro lugar no aplicativo, o IntelliSense acelera o desenvolvimento com o preenchimento de sintaxe e de parâmetro.

Os componentes podem:

- ser aninhados.
- ser criados com o Razor (\*.cshtml) ou com o código C# (\*.cs).
- ser compartilhados por meio de bibliotecas de classes.
- receber um teste de unidade sem precisar de um navegador DOM.

## Infraestrutura

Os Componentes Razor oferecem os principais recursos que a maioria dos aplicativos exige, como:

- Layouts
- Roteamento
- Injeção de dependência

Todos esses recursos são opcionais. Quando um desses recursos não é usado em um aplicativo, a implementação é eliminada do aplicativo quando ele é publicado pelo [Vinculador de IL \(linguagem intermediária\)](#).

## Compartilhamento de código e o .NET Standard

Os aplicativos podem referenciar e usar as bibliotecas [.NET Standard](#) existentes. O .NET Standard é uma especificação formal das APIs do .NET que são comuns entre as implementações do .NET. Há suporte para o .NET standard 2.0 ou superior. As APIs que não são aplicáveis em um navegador da Web (por exemplo, para acessar o sistema de arquivos, abrir um soquete, threading e outros recursos) geram a [PlatformNotSupportedException](#). As bibliotecas de classes do .NET Standard podem ser compartilhadas entre o código do servidor e em aplicativos baseados em navegador.

## Interoperabilidade do JavaScript

Para aplicativos que exigem bibliotecas JavaScript e APIs do navegador de terceiros, o WebAssembly foi projetado para interoperar com o JavaScript. Os Componentes Razor são capazes de usar qualquer biblioteca ou API que o JavaScript possa usar. O código C# pode chamar o código JavaScript, e o código JavaScript pode chamar o código C#. Para obter mais informações, confira [Interoperabilidade do JavaScript](#).

## Otimização

Para aplicativos do lado do cliente, o tamanho do conteúdo é crítico. O Blazor otimiza o tamanho do conteúdo para reduzir os tempos de download. Por exemplo, as partes não usadas dos assemblies do .NET são removidas durante

o processo de build, as respostas HTTP são compactadas e o tempo de execução do .NET e os assemblies são armazenados em cache no navegador.

Os Componentes Razor fornecem um tamanho de conteúdo ainda menor do que o Blazor mantendo os assemblies do .NET, o assembly do aplicativo e o lado do servidor do tempo de execução. Os aplicativos dos Componentes Razor fornecem somente marcação, scripts e folhas de estilos aos clientes.

## Implantação

Use o Blazor para criar um aplicativo do lado do cliente autônomo puro ou um aplicativo do ASP.NET Core de pilha completa que contenha os aplicativos cliente e os aplicativos para servidor:

- Em um **aplicativo autônomo do lado do cliente**, o aplicativo Blazor é compilado em uma pasta *dist* que contém apenas arquivos estáticos. Os arquivos podem ser hospedados no Serviço de Aplicativo do Azure, nas páginas do GitHub, no IIS (configurado como um servidor de arquivos estático), nos servidores do Nodejs e em vários outros servidores e serviços. O .NET não é necessário no servidor de produção.
- Em um **aplicativo do ASP.NET Core de pilha completa**, o código pode ser compartilhado entre os aplicativos cliente e os aplicativos para servidor. O aplicativo dos Componentes Razor do ASP.NET Core resultante, que atende à interface do usuário do lado do cliente e a outros pontos de extremidade de API do lado do servidor, pode ser criado e implantado em qualquer host de nuvem ou local compatível com o ASP.NET Core.

## Sugerir um recurso ou enviar um relatório de bug

Para fazer sugestões e enviar relatórios de bug, [abra um problema](#). Para obter ajuda geral e respostas da comunidade, participe da conversa no [Gitter](#).

## Recursos adicionais

- [WebAssembly](#)
- [Guia do C#](#)
- [Razor](#)
- [HTML](#)

# Estado de sessão e aplicativo no ASP.NET Core

30/01/2019 • 34 minutes to read • [Edit Online](#)

Por [Rick Anderson](#), [Steve Smith](#), [Diana LaRose](#) e [Luke Latham](#)

O HTTP é um protocolo sem estado. Sem realizar etapas adicionais, as solicitações HTTP são mensagens independentes que não mantêm os valores de usuário nem o estado do aplicativo. Este artigo descreve várias abordagens para preservar dados do usuário e estado de aplicativo entre solicitações.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Gerenciamento de estado

O estado pode ser armazenado usando várias abordagens. Cada abordagem é descrita posteriormente neste tópico.

ABORDAGEM DE ARMAZENAMENTO	MECANISMO DE ARMAZENAMENTO
<a href="#">Cookies</a>	Cookies HTTP (podem incluir dados armazenados usando o código de aplicativo do lado do servidor)
<a href="#">Estado de sessão</a>	Cookies HTTP e o código de aplicativo do lado do servidor
<a href="#">TempData</a>	Estado de sessão ou cookies HTTP
<a href="#">Cadeias de consulta</a>	Cadeias de caracteres de consulta HTTP
<a href="#">Campos ocultos</a>	Campos de formulário HTTP
<a href="#">HttpContext.Items</a>	Código do aplicativo do lado do servidor
<a href="#">Cache</a>	Código do aplicativo do lado do servidor
<a href="#">Injeção de dependência</a>	Código do aplicativo do lado do servidor

## Cookies

Cookies armazenam dados entre solicitações. Como os cookies são enviados com cada solicitação, seu tamanho deve ser reduzido ao mínimo. Idealmente, somente um identificador deve ser armazenado em um cookie com os dados armazenados pelo aplicativo. A maioria dos navegadores restringe o tamanho do cookie a 4.096 bytes. Somente um número limitado de cookies está disponível para cada domínio.

Uma vez que os cookies estão sujeitos à adulteração, eles devem ser validados pelo aplicativo. Os cookies podem ser excluídos por usuários e expirarem em clientes. No entanto, os cookies geralmente são a forma mais durável de persistência de dados no cliente.

Frequentemente, cookies são usados para personalização quando o conteúdo é personalizado para um usuário conhecido. O usuário é apenas identificado, e não autenticado, na maioria dos casos. O cookie pode armazenar o nome do usuário, o nome da conta ou a ID de usuário único (como um GUID). Você pode usar o cookie para acessar as configurações personalizadas do usuário, como cor preferida da tela de fundo do site.

Esteja ciente do [RGPD \(Regulamento Geral sobre a Proteção de Dados\) da União Europeia](#) ao emitir cookies e lidar com questões de privacidade. Para obter mais informações, veja [Suporte ao RGPD \(Regulamento Geral sobre a Proteção de Dados\) no ASP.NET Core](#).

## Estado de sessão

Estado de sessão é um cenário do ASP.NET Core para o armazenamento de dados de usuário enquanto o usuário procura um aplicativo Web. Estado de sessão usa um armazenamento mantido pelo aplicativo para que os dados persistam entre solicitações de um cliente. Os dados da sessão são apoiados por um cache e considerados dados efêmeros—o site deve continuar funcionando sem os dados da sessão. Os dados críticos do aplicativo devem ser armazenados no banco de dados do usuário e armazenados em cache na sessão apenas como uma otimização de desempenho.

### NOTE

A sessão não é compatível com os aplicativos [SignalR](#) porque um [Hub SignalR](#) pode ser executado independente de um contexto HTTP. Por exemplo, isso pode ocorrer quando uma solicitação de sondagem longa é mantida aberta por um hub além do tempo de vida do contexto HTTP da solicitação.

O ASP.NET Core mantém o estado de sessão fornecendo um cookie para o cliente que contém uma ID de sessão, que é enviada para o aplicativo com cada solicitação. O aplicativo usa a ID da sessão para buscar os dados da sessão.

Estado de sessão exibe os seguintes comportamentos:

- Uma vez que o cookie da sessão é específico ao navegador, não é possível compartilhar sessões entre navegadores.
- Cookies da sessão são excluídos quando a sessão do navegador termina.
- Se um cookie for recebido de uma sessão expirada, será criada uma nova sessão que usa o mesmo cookie de sessão.
- Sessões vazias não são mantidas—a sessão deve ter pelo menos um valor definido nela para que mantenha a sessão entre solicitações. Quando uma sessão não é mantida, uma nova ID de sessão é gerada para cada nova solicitação.
- O aplicativo mantém uma sessão por um tempo limitado após a última solicitação. O aplicativo define o tempo limite da sessão ou usa o valor padrão de 20 minutos. Estado de sessão é ideal para armazenar dados de usuário específicos para uma sessão em particular, mas em que os dados não requerem armazenamento permanente entre sessões.
- Dados da sessão são excluídos quando a implementação [ISession.Clear](#) é chamada ou quando a sessão expira.
- Não há nenhum mecanismo padrão para informar o código do aplicativo de que um navegador cliente foi fechado ou quando o cookie de sessão foi excluído ou expirou no cliente. Os modelos de páginas do ASP.NET Core MVC e Razor incluem suporte para [suporte do RGPD \(Regulamento Geral sobre a Proteção de Dados\)](#). [Os cookies de estado de sessão não são essenciais](#), o estado de sessão não funciona quando o acompanhamento está desabilitado.

### WARNING

Não armazene dados confidenciais no estado de sessão. O usuário não pode fechar o navegador e limpar o cookie de sessão. Alguns navegadores mantêm cookies de sessão válidos entre as janelas do navegador. Uma sessão não pode ser restringida a um único usuário—o próximo usuário pode continuar a procurar o aplicativo com o mesmo cookie de sessão.

O provedor de cache na memória armazena dados de sessão na memória do servidor em que o aplicativo reside. Em um cenário de farm de servidores:

- Use *sessões persistentes* para vincular cada sessão a uma instância de aplicativo específico em um servidor individual. O [Serviço de Aplicativo do Azure](#) usa o [ARR \(Application Request Routing\)](#) para impor as sessões persistentes por padrão. Entretanto, sessões autoadesivas podem afetar a escalabilidade e complicar atualizações de aplicativos Web. Uma abordagem melhor é usar um Redis ou cache distribuído do SQL Server, que não requer sessões persistentes. Para obter mais informações, consulte [O cache no ASP.NET Core distribuído](#).
- O cookie de sessão é criptografado por meio de [IDataProtector](#). A Proteção de Dados deve ser configurada corretamente para ler os cookies de sessão em cada computador. Para obter mais informações, veja [Proteção de dados do ASP.NET Core](#) e [Provedores de armazenamento de chaves](#).

## Configurar o estado de sessão

O pacote [Microsoft.AspNetCore.Session](#), incluído no [metapacote Microsoft.AspNetCore.App](#), fornece middleware para gerenciar o estado de sessão. Para habilitar o middleware da sessão, `Startup` deve conter:

O pacote [Microsoft.AspNetCore.Session](#) fornece middleware para gerenciar o estado de sessão. Para habilitar o middleware da sessão, `Startup` deve conter:

- Qualquer um dos caches de memória [IDistributedCache](#). A implementação `IDistributedCache` é usada como um repositório de backup para a sessão. Para obter mais informações, consulte [O cache no ASP.NET Core distribuído](#).
- Uma chamada para [AddSession](#) em `ConfigureServices`.
- Uma chamada para [UseSession](#) em `Configure`.

O código a seguir mostra como configurar o provedor de sessão na memória com uma implementação padrão na memória de `IDistributedCache`:

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.Configure<CookiePolicyOptions>(options =>
        {
            options.CheckConsentNeeded = context => true;
            options.MinimumSameSitePolicy = SameSiteMode.None;
        });

        services.AddDistributedMemoryCache();

        services.AddSession(options =>
        {
            // Set a short timeout for easy testing.
            options.IdleTimeout = TimeSpan.FromSeconds(10);
            options.Cookie.HttpOnly = true;
        });

        services.AddMvc()
            .SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Error");
            app.UseHsts();
        }

        app.UseHttpsRedirection();
        app.UseStaticFiles();
        app.UseCookiePolicy();
        app.UseSession();
        app.UseHttpContextItemsMiddleware();
        app.UseMvc();
    }
}
```

```

public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddDistributedMemoryCache();

        services.AddSession(options =>
        {
            // Set a short timeout for easy testing.
            options.IdleTimeout = TimeSpan.FromSeconds(10);
            options.CookieHttpOnly = true;
        });

        services.AddMvc();
    }

    public void Configure(IApplicationBuilder app)
    {
        app.UseSession();
        app.UseHttpContextItemsMiddleware();
        app.UseMvc(routes =>
        {
            routes.MapRoute(
                name: "default",
                template: "{controller=Home}/{action=Index}/{id?}");
        });
    }
}

```

A ordem do middleware é importante. No exemplo anterior, uma exceção `InvalidOperationException` ocorre quando `UseSession` é invocado após `UseMvc`. Para obter mais informações, veja [Ordenação de Middleware](#).

`HttpContext.Session` estará disponível depois que o estado de sessão for configurado.

`HttpContext.Session` não pode ser acessado antes que `useSession` tenha sido chamado.

Não é possível criar uma nova sessão com um novo cookie de sessão depois que o aplicativo começou a gravar no fluxo de resposta. A exceção é registrada no log do servidor Web e não é exibida no navegador.

### Carregue o estado de sessão de maneira assíncrona

O provedor de sessão padrão no ASP.NET Core carregará os registros da sessão do repositório de backup `IDistributedCache` subjacente de maneira assíncrona somente se o método `ISession.LoadAsync` for chamado explicitamente antes dos métodos `TryGetValue`, `Set` ou `Remove`. Se `LoadAsync` não for chamado primeiro, o registro de sessão subjacente será carregado de maneira síncrona, o que pode incorrer em uma penalidade de desempenho em escala.

Para que aplicativos imponham esse padrão, encapsule as implementações `DistributedSessionStore` e `DistributedSession` com versões que geram uma exceção se o método `LoadAsync` não for chamado antes de `TryGetValue`, `Set` ou `Remove`. Registre as versões encapsuladas no contêiner de serviços.

### Opções da sessão

Para substituir os padrões da sessão, use `SessionOptions`.

OPÇÃO	DESCRIÇÃO
-------	-----------

OPÇÃO	DESCRIÇÃO
Cookie	Determina as configurações usadas para criar o cookie. <code>Name</code> assume como padrão <code>SessionDefaults.CookieName</code> ( <code>.AspNetCore.Session</code> ). <code>Path</code> assume como padrão <code>SessionDefaults.CookiePath</code> ( <code>/</code> ). <code>SameSite</code> assume como padrão <code>SameSiteMode.Lax</code> ( <code>1</code> ). <code>HttpOnly</code> assume <code>true</code> como padrão. <code>IsEssential</code> assume <code>false</code> como padrão.
IdleTimeout	O <code>IdleTimeout</code> indica por quanto tempo a sessão pode ficar ociosa antes de seu conteúdo ser abandonado. Cada acesso à sessão redefine o tempo limite. Essa configuração aplica-se somente ao conteúdo da sessão, não ao cookie. O padrão é de 20 minutos.
IOTimeout	O tempo máximo permitido para carregar uma sessão do repositório ou para confirmá-la de volta para o repositório. Essa configuração pode se aplicar somente a operações assíncronas. Esse tempo limite pode ser desabilitado usando <code>InfiniteTimeSpan</code> . O padrão é 1 minuto.

A sessão usa um cookie para rastrear e identificar solicitações de um único navegador. Por padrão, esse cookie se chama `.AspNetCore.Session`, e usa um caminho de `/`. Uma vez que o padrão do cookie não especifica um domínio, ele não fica disponível para o script do lado do cliente na página (porque `HttpOnly` usa `true` como padrão).

OPÇÃO	DESCRIÇÃO
CookieDomain	Determina o domínio usado para criar o cookie. <code>CookieDomain</code> não é definido por padrão.
CookieHttpOnly	Determina se o navegador deve permitir que o cookie seja acessado pelo JavaScript do lado do cliente. O padrão é <code>true</code> , o que significa que o cookie é transmitido apenas às solicitações HTTP e não é disponibilizado para o script na página.
CookieName	Determina o nome do cookie usado para manter a ID da sessão. O padrão é <code>SessionDefaults.CookieName</code> ( <code>.AspNetCore.Session</code> ).
CookiePath	Determina o caminho usado para criar o cookie. Usa como padrão <code>SessionDefaults.CookiePath</code> ( <code>/</code> ).
CookieSecure	Determina se o cookie deve ser transmitido apenas em solicitações HTTPS. O padrão é <code>CookieSecurePolicy.None</code> ( <code>2</code> ).
IdleTimeout	O <code>IdleTimeout</code> indica por quanto tempo a sessão pode ficar ociosa antes de seu conteúdo ser abandonado. Cada acesso à sessão redefine o tempo limite. Observe que isso só se aplica ao conteúdo da sessão, não o cookie. O padrão é de 20 minutos.

A sessão usa um cookie para rastrear e identificar solicitações de um único navegador. Por padrão, esse cookie se chama `.AspNet.Session`, e usa um caminho de `/`.

Para substituir os padrões de sessão do cookie, use `SessionOptions`:

```

public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddDistributedMemoryCache();

    services.AddMvc()
        .SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    services.AddSession(options =>
    {
        options.Cookie.Name = ".AdventureWorks.Session";
        options.IdleTimeout = TimeSpan.FromSeconds(10);
    });
}

```

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddDistributedMemoryCache();

    services.AddSession(options =>
    {
        options.CookieName = ".AdventureWorks.Session";
        options.IdleTimeout = TimeSpan.FromSeconds(10);
    });

    services.AddMvc();
}

```

O aplicativo usa a propriedade `IdleTimeout` para determinar por quanto tempo uma sessão pode ficar ociosa antes de seu conteúdo no cache do servidor ser abandonado. Essa propriedade é independente da expiração do cookie. Cada solicitação que passa pelo [Middleware da Sessão](#) redefine o tempo limite.

Estado de sessão é *sem bloqueio*. Se duas solicitações tentarem simultaneamente modificar o conteúdo de uma sessão, a última solicitação substituirá a primeira. `Session` é implementado como uma *sessão coerente*, o que significa que todo o conteúdo é armazenado junto. Quando duas solicitações buscam modificar valores de sessão diferentes, a última solicitação pode substituir as alterações de sessão feitas pelo primeira.

## Definir e obter valores de Session

Estado de sessão é acessado de uma classe [PageModel](#) Razor Pages ou classe [Controlador](#) MVC com `HttpContext.Session`. Esta propriedade é uma implementação de `ISession`.

A implementação de `ISession` fornece vários métodos de extensão para definir e recuperar valores de inteiro e cadeia de caracteres. Os métodos de extensão estão no namespace [Microsoft.AspNetCore.Http](#) (adicone uma instrução `using Microsoft.AspNetCore.Http;` para acessar os métodos de extensão) quando o pacote [Microsoft.AspNetCore.Http.Extensions](#) for referenciado pelo projeto. Ambos os pacotes são incluídos no metapacote [Microsoft.AspNetCore.App](#).

A implementação de `ISession` fornece vários métodos de extensão para definir e recuperar valores de inteiro e cadeia de caracteres. Os métodos de extensão estão no namespace [Microsoft.AspNetCore.Http](#) (adicone uma instrução `using Microsoft.AspNetCore.Http;` para acessar os métodos de extensão) quando o pacote [Microsoft.AspNetCore.Http.Extensions](#) for referenciado pelo projeto.

Métodos de extensão `ISession`:

- [Get\(I Session, String\)](#)
- [GetInt32\(I Session, String\)](#)
- [GetString\(I Session, String\)](#)
- [SetInt32\(I Session, String, Int32\)](#)
- [SetString\(I Session, String, String\)](#)

O exemplo a seguir recupera o valor da sessão para a chave `IndexModel.SessionKeyName` (`_Name` no aplicativo de exemplo) em uma página do Razor Pages:

```
@page
@using Microsoft.AspNetCore.Http
@model IndexModel

...
Name: @HttpContext.Session.GetString(IndexModel.SessionKeyName)
```

O exemplo a seguir mostra como definir e obter um número inteiro e uma cadeia de caracteres:

```
public class IndexModel : PageModel
{
    public const string SessionKeyName = "_Name";
    public const string SessionKeyAge = "_Age";
    const string SessionKeyTime = "_Time";

    public string SessionInfo_Name { get; private set; }
    public string SessionInfo_Age { get; private set; }
    public string SessionInfo_CurrentTime { get; private set; }
    public string SessionInfo_SessionTime { get; private set; }
    public string SessionInfo_MiddlewareValue { get; private set; }

    public void OnGet()
    {
        // Requires: using Microsoft.AspNetCore.Http;
        if (string.IsNullOrEmpty(HttpContext.Session.GetString(SessionKeyName)))
        {
            HttpContext.Session.SetString(SessionKeyName, "The Doctor");
            HttpContext.Session.SetInt32(SessionKeyAge, 773);
        }
    }

    var name = HttpContext.Session.GetString(SessionKeyName);
    var age = HttpContext.Session.GetInt32(SessionKeyAge);
```

```

public class HomeController : Controller
{
    const string SessionKeyName = "_Name";
    const string SessionKeyYearsMember = "_YearsMember";
    const string SessionKeyDate = "_Date";

    public IActionResult Index()
    {
        // Requires using Microsoft.AspNetCore.Http;
        HttpContext.Session.SetString(SessionKeyName, "Rick");
        HttpContext.Session.SetInt32(SessionKeyYearsMember, 3);

        return RedirectToAction("SessionNameYears");
    }

    public IActionResult SessionNameYears()
    {
        var name = HttpContext.Session.GetString(SessionKeyName);
        var yearsMember = HttpContext.Session.GetInt32(SessionKeyYearsMember);

        return Content($"Name: \"{name}\", Membership years: \"{yearsMember}\"");
    }
}

```

Todos os dados de sessão devem ser serializados para habilitar um cenário de cache distribuído, mesmo ao usar o cache na memória. Cadeia de caracteres mínima e serializadores número são fornecidos (confira os métodos e os métodos de extensão de [ISession](#)). Tipos complexos devem ser serializados pelo usuário por meio de outro mecanismo, como JSON.

Adicione os seguintes métodos de extensão para definir e obter objetos serializáveis:

```

public static class SessionExtensions
{
    public static void Set<T>(this ISession session, string key, T value)
    {
        session.SetString(key, JsonConvert.SerializeObject(value));
    }

    public static T Get<T>(this ISession session, string key)
    {
        var value = session.GetString(key);

        return value == null ? default(T) :
            JsonConvert.DeserializeObject<T>(value);
    }
}

```

```

public static class SessionExtensions
{
    public static void Set<T>(this ISession session, string key, T value)
    {
        session.SetString(key, JsonConvert.SerializeObject(value));
    }

    public static T Get<T>(this ISession session, string key)
    {
        var value = session.GetString(key);
        return value == null ? default(T) :
            JsonConvert.DeserializeObject<T>(value);
    }
}

```

O exemplo a seguir mostra como definir e obter um objeto serializável com os métodos de extensão:

```
// Requires you add the Set and Get extension method mentioned in the topic.  
if (HttpContext.Session.Get<DateTime>(SessionKeyTime) == default(DateTime))  
{  
    HttpContext.Session.Set<DateTime>(SessionKeyTime, currentTime);  
}
```

```
public IActionResult SetDate()  
{  
    // Requires you add the Set extension method mentioned in the topic.  
    HttpContext.Session.Set<DateTime>(SessionKeyDate, DateTime.Now);  
  
    return RedirectToAction("GetDate");  
}  
  
public IActionResult GetDate()  
{  
    // Requires you add the Get extension method mentioned in the topic.  
    var date = HttpContext.Session.Get<DateTime>(SessionKeyDate);  
    var sessionTime = date.TimeOfDay.ToString();  
    var currentTime = DateTime.Now.TimeOfDay.ToString();  
  
    return Content($"Current time: {currentTime} - "  
        + $"session time: {sessionTime}");  
}
```

## TempData

O ASP.NET Core expõe a [propriedade TempData de um modelo de página do Razor Pages](#) ou [TempData de um controlador MVC](#). Essa propriedade armazena dados até eles serem lidos. Os métodos `Keep` e `Peek` podem ser usados para examinar os dados sem exclusão. TempData é particularmente útil para redirecionamento em casos em que são necessários dados para mais de uma única solicitação. TempData é implementado por provedores de TempData usando cookies ou estado de sessão.

### Provedores de TempData

No ASP.NET Core 2.0 ou posterior, o provedor TempData baseado em cookies é usado por padrão para armazenar TempData em cookies.

Os dados do cookie são criptografados usando [IDataProtector](#), codificado com [Base64UrlTextEncoder](#), em seguida, em partes. Como o cookie é dividido em partes, o limite de tamanho de cookie único encontrado no ASP.NET Core 1.x não se aplica. Os dados do cookie não são compactados porque a compactação de dados criptografados pode levar a problemas de segurança, como os ataques [CRIME](#) e [BREACH](#). Para obter mais informações sobre o provedor de TempData baseado em cookie, consulte [CookieTempDataProvider](#).

No ASP.NET Core 1.0 e 1.1, o provedor de TempData do estado de sessão é o provedor padrão.

### Escolha um provedor de TempData

Escolher um provedor de TempData envolve várias considerações, como:

1. O aplicativo já usa estado de sessão? Se for o caso, usar o provedor de TempData do estado de sessão não terá custos adicionais para o aplicativo (além do tamanho dos dados).
2. O aplicativo usa TempData apenas raramente para quantidades relativamente pequenas de dados (até 500 bytes)? Em caso afirmativo, o provedor de TempData do cookie adiciona um pequeno custo a cada solicitação que transportar TempData. Caso contrário, o provedor de TempData do estado de sessão pode ser útil para evitar fazer viagens de ida e volta para uma grande quantidade de dados a cada solicitação até que TempData seja consumido.
3. O aplicativo é executado em um farm de servidores em vários servidores? Se for o caso, não será necessária nenhuma configuração adicional para usar o provedor TempData do cookie fora da Proteção de Dados

(confira [Proteção de dados do ASP.NET Core e Provedores de armazenamento de chaves](#)).

#### NOTE

A maioria dos clientes da Web (como navegadores da Web) impõem limites quanto ao tamanho máximo de cada cookie, o número total de cookies ou ambos. Ao usar o provedor TempData do cookie, verifique se o aplicativo não ultrapassará esses limites. Considere o tamanho total dos dados. Conta para aumento no tamanho de cookie devido à criptografia e ao agrupamento.

## Configurar o provedor de TempData

O provedor de TempData baseado em cookie é habilitado por padrão.

Para habilitar o provedor TempData baseado em sessão, use o método de extensão

[AddSessionStateTempDataProvider](#):

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddMvc()
        .SetCompatibilityVersion(CompatibilityVersion.Version_2_1)
        .AddSessionStateTempDataProvider();

    services.AddSession();
}

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseCookiePolicy();
    app.UseSession();
    app.UseMvc();
}
```

O código da classe `Startup` a seguir configura o provedor de TempData baseado em sessão:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddSession();
}

public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
{
    app.UseSession();
    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

A ordem do middleware é importante. No exemplo anterior, uma exceção `InvalidOperationException` ocorre quando `UseSession` é invocado após `UseMvc`. Para obter mais informações, veja [Ordenação de Middleware](#).

#### IMPORTANT

Se o alvo for o .NET Framework e o provedor TempData baseado em sessão for usado, adicione o pacote `Microsoft.AspNetCore.Session` ao projeto.

## Cadeias de consulta

É possível passar uma quantidade limitada de dados de uma solicitação para outra adicionando-a à cadeia de caracteres de consulta da nova solicitação. Isso é útil para capturar o estado de uma maneira persistente que permita que links com estado inserido sejam compartilhados por email ou por redes sociais. Uma vez que cadeias de consulta de URL são públicas, nunca use cadeias de consulta para dados confidenciais.

Além da possibilidade de ocorrência de compartilhamento não intencional, incluir dados em cadeias de consulta pode criar oportunidades para ataques de [CSRF \(solicitação intersite forjada\)](#), que podem enganar os usuários para que eles visitem sites mal-intencionados enquanto estão autenticados. Invasores então podem roubar dados do usuário do aplicativo ou executar ações mal-intencionadas em nome do usuário. Qualquer estado de sessão ou aplicativo preservado deve se proteger contra ataques CSRF. Para obter mais informações, confira [Impedir ataques de XSRF/CSRF \(solicitação intersite forjada\)](#).

## Campos ocultos

Dados podem ser salvos em campos de formulário ocultos e postados novamente na solicitação seguinte. Isso é comum em formulários com várias páginas. Uma vez que o cliente potencialmente pode adulterar os dados, o aplicativo deve sempre revalidar os dados armazenados nos campos ocultos.

## HttpContext.Items

A coleção `HttpContext.Items` é usada para armazenar dados durante o processamento de uma única solicitação. O conteúdo da coleção é descartado após uma solicitação ser processada. A coleção `Items` costuma ser usada para permitir que componentes ou middleware se comuniquem quando operam em diferentes momentos durante uma solicitação e não têm nenhuma maneira direta de passar parâmetros.

No exemplo a seguir, `middleware` adiciona `isverified` à coleção `Items`.

```
app.Use(async (context, next) =>
{
    // perform some verification
    context.Items["isVerified"] = true;
    await next.Invoke();
});
```

Mais tarde no pipeline, outro middleware poderá acessar o valor de `isVerified`:

```
app.Run(async (context) =>
{
    await context.Response.WriteAsync($"Verified: {context.Items["isVerified"]}");
});
```

Para middleware usado apenas por um único aplicativo, chaves `string` são aceitáveis. Middleware compartilhado entre instâncias do aplicativo deve usar chaves de objeto únicas para evitar colisões de chaves. O exemplo a seguir mostra como usar uma chave de objeto exclusiva definida em uma classe de middleware:

```
public class HttpContextItemsMiddleware
{
    private readonly RequestDelegate _next;
    public static readonly object HttpContextItemsMiddlewareKey = new Object();

    public HttpContextItemsMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public async Task Invoke(HttpContext httpContext)
    {
        httpContext.Items[HttpContextItemsMiddlewareKey] = "K-9";

        await _next(httpContext);
    }
}

public static class HttpContextItemsMiddlewareExtensions
{
    public static IApplicationBuilder
        UseHttpContextItemsMiddleware(this IApplicationBuilder builder)
    {
        return builder.UseMiddleware<HttpContextItemsMiddleware>();
    }
}
```

```
// You may need to install the Microsoft.AspNetCore.Http.Abstractions package
public class HttpContextItemsMiddleware
{
    private readonly RequestDelegate _next;
    public static readonly object HttpContextItemsMiddlewareKey = new Object();

    public HttpContextItemsMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public async Task Invoke(HttpContext httpContext)
    {
        httpContext.Items[HttpContextItemsMiddlewareKey] = "K-9";

        await _next(httpContext);
    }
}

public static class HttpContextItemsMiddlewareExtensions
```

Outros códigos podem acessar o valor armazenado em `HttpContext.Items` usando a chave exposta pela classe do middleware:

```
HttpContext.Items
    .TryGetValue(HttpContextItemsMiddleware.HttpContextItemsMiddlewareKey,
        out var middlewareSetValue);
SessionInfo_MiddlewareValue =
    middlewareSetValue?.ToString() ?? "Middleware value not set!";
```

```
public IActionResult GetMiddlewareValue()
{
    var middlewareSetValue =
        HttpContext.Items[HttpContextItemsMiddleware.HttpContextItemsMiddlewareKey];

    return Content($"Value: {middlewareSetValue}");
}
```

Essa abordagem também tem a vantagem de eliminar o uso de cadeias de caracteres de chave no código.

## Cache

O cache é uma maneira eficiente de armazenar e recuperar dados. O aplicativo pode controlar o tempo de vida de itens em cache.

Dados armazenados em cache não são associados uma solicitação, usuário ou sessão específico. **Tenha cuidado para não armazenar em cache dados específicos do usuário que podem ser recuperados pelas solicitações de outros usuários.**

Para obter mais informações, consulte [Cache de resposta no ASP.NET Core](#).

## Injeção de dependência

Use a [Injeção de dependência](#) para disponibilizar dados para todos os usuários:

1. Defina um serviço que contenha os dados. Por exemplo, uma classe denominada `MyAppData` está definida:

```
public class MyAppData
{
    // Declare properties and methods
}
```

2. Adicione a classe de serviço a `Startup.ConfigureServices`:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<MyAppData>();
}
```

3. Consuma a classe de serviço de dados:

```
public class IndexModel : PageModel
{
    public IndexModel(MyAppData myService)
    {
        // Do something with the service
        // Examples: Read data, store in a field or property
    }
}
```

```
public class HomeController : Controller
{
    public HomeController(MyAppData myService)
    {
        // Do something with the service
        // Examples: Read data, store in a field or property
    }
}
```

## Erros comuns

- "Não é possível resolver o serviço para o tipo 'Microsoft.Extensions.Caching.Distributed.IDistributedCache' ao tentar ativar 'Microsoft.AspNetCore.Session.DistributedSessionStore'."

Geralmente, isso é causado quando não é configurada pelo menos uma implementação de `IDistributedCache`. Para obter mais informações, consulte [O cache no ASP.NET Core distribuído e Memória de cache no ASP.NET Core](#).

- Caso a sessão de middleware não consiga manter uma sessão (por exemplo, se o repositório de backup não estiver disponível), o middleware registrará a exceção e a solicitação continuará normalmente. Isso leva a um comportamento imprevisível.

Por exemplo, um usuário armazena um carrinho de compras na sessão. O usuário adiciona um item ao carrinho, mas a confirmação falha. O aplicativo não sabe sobre a falha, assim, relata ao usuário que o item foi adicionado ao seu carrinho, o que não é verdade.

A abordagem recomendada para verificar se há erros é chamar `await feature.Session.CommitAsync();` do código de aplicativo quando o aplicativo tiver terminado de gravar na sessão. `CommitAsync` gerará uma exceção se o repositório de backup não estiver disponível. Se `CommitAsync` falhar, o aplicativo poderá

processar a exceção. `LoadAsync` gera sob as mesmas condições em que o armazenamento de dados não está disponível.

## Recursos adicionais

[Hospedar o ASP.NET Core em um web farm](#)

# Auxiliares de Marca no ASP.NET Core

29/10/2018 • 25 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

## E que são Auxiliares de Marca?

Os Auxiliares de Marca permitem que o código do lado do servidor participe da criação e renderização de elementos HTML em arquivos do Razor. Por exemplo, o `ImageTagHelper` interno pode acrescentar um número de versão ao nome da imagem. Sempre que a imagem é alterada, o servidor gera uma nova versão exclusiva para a imagem, de modo que os clientes tenham a garantia de obter a imagem atual (em vez de uma imagem obsoleta armazenada em cache). Há muitos Auxiliares de Marca internos para tarefas comuns – como criação de formulários, links, carregamento de ativos e muito mais – e ainda outros disponíveis em repositórios GitHub públicos e como NuGet. Os Auxiliares de Marca são criados no C# e são direcionados a elementos HTML de acordo com o nome do elemento, o nome do atributo ou a marca pai. Por exemplo, o `LabelTagHelper` interno pode ser direcionado ao elemento `<label>` HTML quando os atributos `LabelTagHelper` são aplicados. Se você está familiarizado com [Auxiliares HTML](#), os Auxiliares de Marca reduzem as transições explícitas entre HTML e C# em exibições do Razor. Em muitos casos, os Auxiliares HTML fornecem uma abordagem alternativa a um Auxiliar de Marca específico, mas é importante reconhecer que os Auxiliares de Marca não substituem os Auxiliares HTML e que não há um Auxiliar de Marca para cada Auxiliar HTML. [Comparação entre Auxiliares de Marca e Auxiliares HTML](#) explica as diferenças mais detalhadamente.

## O que os Auxiliares de Marca fornecem

**Uma experiência de desenvolvimento amigável a HTML** Na maioria dos casos, a marcação do Razor com Auxiliares de Marca parece um HTML padrão. Designers de front-end familiarizados com HTML/CSS/JavaScript podem editar o Razor sem aprender a sintaxe Razor do C#.

**Um ambiente avançado do IntelliSense para criação do HTML e da marcação do Razor** Isso é um nítido contraste com Auxiliares HTML, a abordagem anterior para a criação do lado do servidor de marcação nas exibições do Razor. [Comparação entre Auxiliares de Marca e Auxiliares HTML](#) explica as diferenças mais detalhadamente. [Suporte do IntelliSense para Auxiliares de Marca](#) explica o ambiente do IntelliSense. Até mesmo desenvolvedores experientes com a sintaxe Razor do C# são mais produtivos usando Auxiliares de Marca do que escrevendo a marcação do Razor do C#.

**Uma maneira de fazer com que você fique mais produtivo e possa produzir um código mais robusto, confiável e possível de ser mantido usando as informações apenas disponíveis no servidor** Por exemplo, historicamente, o mantra da atualização de imagens era alterar o nome da imagem quando a imagem era alterada. As imagens devem ser armazenadas em cache de forma agressiva por motivos de desempenho e, a menos que você altere o nome de uma imagem, você corre o risco de os clientes obterem uma cópia obsoleta. Historicamente, depois que uma imagem era editada, o nome precisava ser alterado e cada referência à imagem no aplicativo Web precisava ser atualizada. Não apenas isso exige muito trabalho, mas também é propenso a erros (você pode perder uma referência, inserir a cadeia de caracteres incorreta

acidentalmente, etc.) O `ImageTagHelper` interno pode fazer isso para você automaticamente. O `ImageTagHelper` pode acrescentar um número de versão ao nome da imagem, de modo que sempre que a imagem é alterada, o servidor gera automaticamente uma nova versão exclusiva para a imagem. Os clientes têm a garantia de obter a imagem atual. Basicamente, essa economia na robustez e no trabalho é obtida gratuitamente com o `ImageTagHelper`.

A maioria dos auxiliares de marca internos é direcionada a elementos HTML padrão e fornece atributos do lado do servidor para o elemento. Por exemplo, o elemento `<input>` usado em várias exibições na pasta *Exibição/Conta* contém o atributo `asp-for`. Esse atributo extrai o nome da propriedade do modelo especificado no HTML renderizado. Considere uma exibição Razor com o seguinte modelo:

```
public class Movie
{
    public int ID { get; set; }
    public string Title { get; set; }
    public DateTime ReleaseDate { get; set; }
    public string Genre { get; set; }
    public decimal Price { get; set; }
}
```

A seguinte marcação do Razor:

```
<label asp-for="Movie.Title"></label>
```

Gera o seguinte HTML:

```
<label for="Movie_Title">Title</label>
```

O atributo `asp-for` é disponibilizado pela propriedade `For` no `LabelTagHelper`. Confira [Auxiliares de marca de autor](#) para obter mais informações.

## Gerenciando o escopo do Auxiliar de Marca

O escopo dos Auxiliares de Marca é controlado por uma combinação de `@addTagHelper`, `@removeTagHelper` e o caractere de recusa "!".

`@addTagHelper` **disponibiliza os Auxiliares de Marca**

Se você criar um novo aplicativo Web ASP.NET Core chamado *AuthoringTagHelpers*, o seguinte arquivo *Views/\_ViewImports.cshtml* será adicionado ao projeto:

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@addTagHelper *, AuthoringTagHelpers
```

A diretiva `@addTagHelper` disponibiliza os Auxiliares de Marca para a exibição. Nesse caso, o arquivo de exibição é *Pages/\_ViewImports.cshtml*, que é herdado por padrão por todos os arquivos na pasta *Páginas* e suas subpastas, disponibilizando os Auxiliares de Marca. O código acima usa a sintaxe de curinga ("\*") para especificar que todos os Auxiliares de Marca no assembly especificado (*Microsoft.AspNetCore.Mvc.TagHelpers*) estarão disponíveis para todos os arquivos de exibição no diretório *Views* ou subdiretório. O primeiro parâmetro após `@addTagHelper` especifica os Auxiliares de Marca a serem carregados (estamos usando "\*" para todos os Auxiliares de Marca) e o segundo parâmetro "*Microsoft.AspNetCore.Mvc.TagHelpers*" especifica o assembly que contém os Auxiliares de Marca.

*Microsoft.AspNetCore.Mvc.TagHelpers* é o assembly para os Auxiliares de Marca internos do ASP.NET Core.

Para expor todos os Auxiliares de Marca neste projeto (que cria um assembly chamado *AuthoringTagHelpers*), você usará o seguinte:

```
@using AuthoringTagHelpers  
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers  
@addTagHelper *, AuthoringTagHelpers
```

Se o projeto contém um `EmailTagHelper` com o namespace padrão (`AuthoringTagHelpers.TagHelpers.EmailTagHelper`), forneça o FQN ( nome totalmente qualificado) do Auxiliar de Marca:

```
@using AuthoringTagHelpers  
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers  
@addTagHelper AuthoringTagHelpers.TagHelpers.EmailTagHelper, AuthoringTagHelpers
```

Para adicionar um Auxiliar de Marca a uma exibição usando um FQN, primeiro adicione o FQN (`AuthoringTagHelpers.TagHelpers.EmailTagHelper`) e, em seguida, o nome do assembly (*AuthoringTagHelpers*). A maioria dos desenvolvedores prefere usar a sintaxe de curinga `"*"`. A sintaxe de curinga permite que você insira o caractere curinga `"*"` como o sufixo de um FQN. Por exemplo, uma das seguintes diretivas exibirá o `EmailTagHelper`:

```
@addTagHelper AuthoringTagHelpers.TagHelpers.E*, AuthoringTagHelpers  
@addTagHelper AuthoringTagHelpers.TagHelpers.Email*, AuthoringTagHelpers
```

Conforme mencionado anteriormente, a adição da diretiva `@addTagHelper` ao arquivo `Views/_ViewImports.cshtml` disponibiliza o Auxiliar de Marca para todos os arquivos de exibição no diretório `Views` e subdiretórios. Use a diretiva `@addTagHelper` nos arquivos de exibição específicos se você deseja aceitar a exposição do Auxiliar de Marca a apenas essas exibições.

#### `@removeTagHelper` remove os Auxiliares de Marca

O `@removeTagHelper` tem os mesmos dois parâmetros `@addTagHelper` e remove um Auxiliar de Marca adicionado anteriormente. Por exemplo, `@removeTagHelper` aplicado a uma exibição específica remove o Auxiliar de Marca especificado da exibição. O uso de `@removeTagHelper` em um arquivo `Views/Folder/_ViewImports.cshtml` remove o Auxiliar de Marca especificado de todas as exibições em `Folder`.

#### Controlando o escopo do Auxiliar de Marca com o arquivo `_ViewImports.cshtml`

Adicione um `_ViewImports.cshtml` a qualquer pasta de exibição e o mecanismo de exibição aplicará as diretivas desse arquivo e do arquivo `Views/_ViewImports.cshtml`. Se você adicionou um arquivo `Views/Home/_ViewImports.cshtml` vazio às exibições `Home`, não haverá nenhuma alteração porque o arquivo `_ViewImports.cshtml` é aditivo. As diretivas `@addTagHelper` que você adicionou ao arquivo `Views/Home/_ViewImports.cshtml` (que não estão no arquivo `Views/_ViewImports.cshtml` padrão) exporão os Auxiliares de Marca às exibições somente na pasta `Home`.

#### Recusando elementos individuais

Desabilite um Auxiliar de Marca no nível do elemento com o caractere de recusa do Auxiliar de Marca (`!"`). Por exemplo, a validação `Email` está desabilitada no `<span>` com o caractere de recusa do Auxiliar de Marca:

```
<!span asp-validation-for="Email" class="text-danger"></span>
```

É necessário aplicar o caractere de recusa do Auxiliar de Marca à marca de abertura e fechamento. (O editor do Visual Studio adiciona automaticamente o caractere de recusa à marca de fechamento quando um é adicionado à marca de abertura). Depois de adicionar o caractere de recusa, o elemento e os atributos do Auxiliar de Marca deixam de ser exibidos em uma fonte diferenciada.

### Usando `@tagHelperPrefix` para tornar explícito o uso do Auxiliar de Marca

A diretiva `@tagHelperPrefix` permite que você especifique uma cadeia de caracteres de prefixo de marca para habilitar o suporte do Auxiliar de Marca e tornar explícito o uso do Auxiliar de Marca. Por exemplo, você pode adicionar a seguinte marcação ao arquivo `Views/_ViewImports.cshtml`:

```
@tagHelperPrefix th:
```

Na imagem do código abaixo, o prefixo do Auxiliar de Marca é definido como `th:`; portanto, somente esses elementos que usam o prefixo `th:` dão suporte a Auxiliares de Marca (elementos habilitados para Auxiliar de Marca têm uma fonte diferenciada). Os elementos `<label>` e `<input>` têm o prefixo do Auxiliar de Marca e são habilitados para Auxiliar de Marca, ao contrário do elemento `<span>`.

```
<div class="form-group">
    <th:label asp-for="Password" class="col-md-2"></th:label>
    <div class="col-md-10">
        <th:input asp-for="Password" class="form-control" />
        <span asp-validation-for="Password" class="text-danger"></span>
    </div>
</div>
```

As mesmas regras de hierarquia que se aplicam a `@addTagHelper` também se aplicam a `@tagHelperPrefix`.

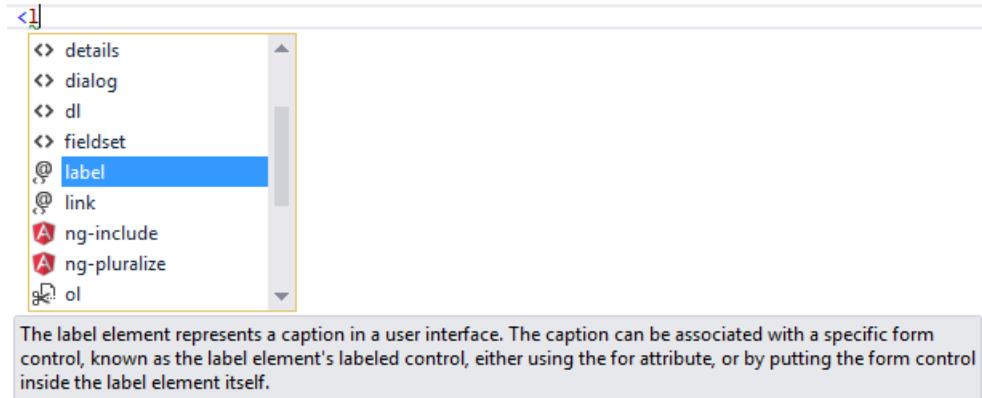
## Auxiliares de Marca com autofechamento

Muitos Auxiliares de Marca não podem ser usados como marcações com autofechamento. Alguns Auxiliares de Marca são projetados para serem marcações com autofechamento. Usar um Auxiliar de Marca que não foi projetado para ser de autofechamento suprime a saída renderizada. Um Auxiliar de Marca com autofechamento resulta em uma marca com autofechamento na saída renderizada. Para obter mais informações, confira [esta observação](#) em [Criando Auxiliares de Marca](#).

## Suporte do IntelliSense para Auxiliares de Marca

Quando você cria um novo aplicativo Web ASP.NET Core no Visual Studio, ele adiciona o pacote NuGet "Microsoft.AspNetCore.Razor.Tools". Esse é o pacote que adiciona ferramentas do Auxiliar de Marca.

Considere a escrita de um elemento `<label>` HTML. Assim que você insere `<l` no editor do Visual Studio, o IntelliSense exibe elementos correspondentes:



Não só você obtém a ajuda do HTML, mas também o ícone (o "@" symbol with "<>" abaixo dele).

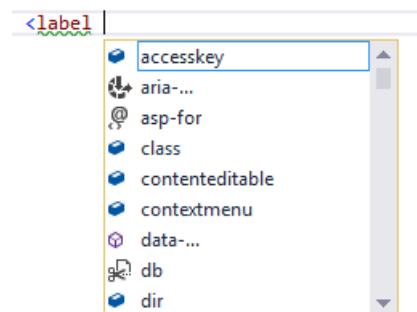


identifica o elemento como direcionado a Auxiliares de Marca. Elementos HTML puros (como o `fieldset`) exibem o ícone "<>".

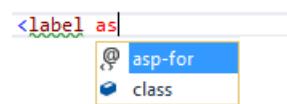
Uma marca `<label>` HTML pura exibe a marca HTML (com o tema de cores padrão do Visual Studio) em uma fonte marrom, os atributos em vermelho e os valores de atributo em azul.

```
<label class="col-md-2">Email</label>
```

Depois de inserir `<label>`, o IntelliSense lista os atributos HTML/CSS disponíveis e os atributos direcionados ao Auxiliar de Marca:



O preenchimento de declaração do IntelliSense permite que você pressione a tecla TAB para preencher a declaração com o valor selecionado:

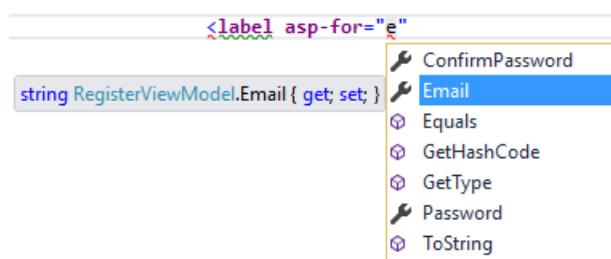


Assim que um atributo do Auxiliar de Marca é inserido, as fontes da marca e do atributo são alteradas. Usando o tema de cores padrão "Azul" ou "Claro" do Visual Studio, a fonte é roxo em negrito. Se estiver usando o tema "Escuro", a fonte será azul-petróleo em negrito. As imagens deste documento foram obtidas usando o tema padrão.

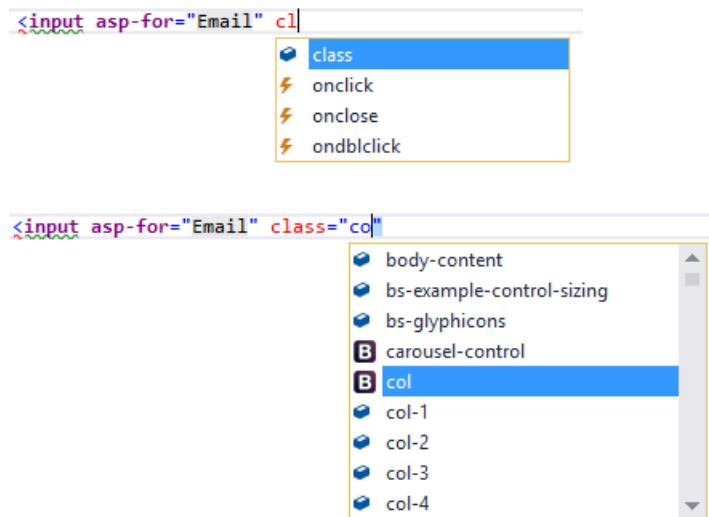
```
<label asp-
```

Insira o atalho *CompleteWord* do Visual Studio – Ctrl + barra de espaços é o **padrão** dentro das aspas duplas ("") e você está agora no C#, exatamente como estaria em uma classe do C#. O IntelliSense exibe todos os métodos e propriedades no modelo de página. Os métodos e as propriedades estão disponíveis porque o tipo de propriedade é `ModelExpression`. Na imagem

abaixo, estou editando a exibição `Register` e, portanto, o `RegisterViewModel` está disponível.



O IntelliSense lista as propriedades e os métodos disponíveis para o modelo na página. O ambiente avançado de IntelliSense ajuda você a selecionar a classe CSS:



## Comparação entre Auxiliares de Marca e Auxiliares HTML

Os Auxiliares de Marca são anexados a elementos HTML em exibições do Razor, enquanto os [Auxiliares HTML](#) são invocados como métodos intercalados com HTML nas exibições do Razor. Considere a seguinte marcação do Razor, que cria um rótulo HTML com a classe CSS "caption":

```
@Html.Label("FirstName", "First Name:", new {@class="caption"})
```

O símbolo de arroba (`@`) informa o Razor de que este é o início do código. Os dois próximos parâmetros ("FirstName" e "First Name:") são cadeias de caracteres; portanto, o [IntelliSense](#) não pode ajudar. O último argumento:

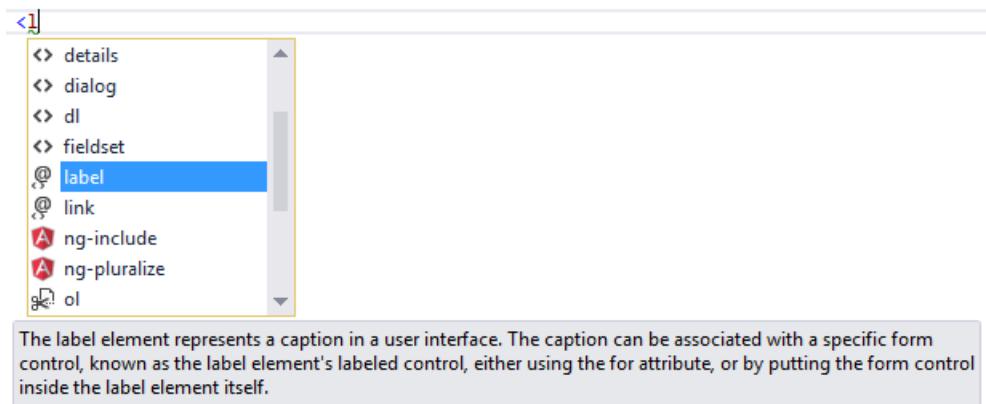
```
new {@class="caption"}
```

É um objeto anônimo usado para representar atributos. Como a **classe** é uma palavra-chave reservada no C#, use o símbolo `@` para forçar o C# a interpretar "`@class=`" como um símbolo (nome da propriedade). Para um designer de front-end (alguém familiarizado com HTML/CSS/JavaScript e outras tecnologias de cliente, mas não familiarizado com o C# e Razor), a maior parte da linha é estranha. Toda a linha precisa ser criada sem nenhuma ajuda do IntelliSense.

Usando o `LabelTagHelper`, a mesma marcação pode ser escrita como:

```
<label class="caption" asp-for="FirstName"></label>
```

Com a versão do Auxiliar de Marca, assim que você insere `<1` no editor do Visual Studio, o IntelliSense exibe elementos correspondentes:



O IntelliSense ajuda você a escrever a linha inteira. O `LabelTagHelper` também usa como padrão a definição do conteúdo do valor de atributo `asp-for` ("FirstName") como "First Name"; ele converte propriedades concatenadas em uma frase composta do nome da propriedade com um espaço em que ocorre cada nova letra maiúscula. Na seguinte marcação:

```
<label class="caption" asp-for="FirstName"></label>
```

gera:

```
<label class="caption" for="FirstName">First Name</label>
```

O conteúdo concatenado para maiúsculas e minúsculas não é usado se você adiciona o conteúdo ao `<label>`. Por exemplo:

```
<label class="caption" asp-for="FirstName">Name First</label>
```

gera:

```
<label class="caption" for="FirstName">Name First</label>
```

A imagem de código a seguir mostra a parte do Formulário da exibição do Razor `Views/Account/Register.cshtml` gerada com base no modelo herdado do ASP.NET 4.5 MVC incluído com o Visual Studio 2015.

```

@using (Html.BeginForm("Register", "Account", FormMethod.Post, new { @class = "form-horizo
{
    @Html.AntiForgeryToken()
    <h4>Create a new account.</h4>
    <hr />
    @Html.ValidationSummary("", new { @class = "text-danger" })
    <div class="form-group">
        @Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.TextBoxFor(m => m.Email, new { @class = "form-control" })
        </div>
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.PasswordFor(m => m.Password, new { @class = "form-control" })
        </div>
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.ConfirmPassword, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.PasswordFor(m => m.ConfirmPassword, new { @class = "form-control" })
        </div>
    </div>
    <div class="form-group">
        <div class="col-md-offset-2 col-md-10">
            <input type="submit" class="btn btn-default" value="Register" />
        </div>
    </div>
}

```

O editor do Visual Studio exibe o código C# com uma tela de fundo cinza. Por exemplo, o Auxiliar HTML `AntiForgeryToken`:

```

@Html.AntiForgeryToken()

```

é exibido com uma tela de fundo cinza. A maior parte da marcação na exibição Register é C#. Compare isso com a abordagem equivalente ao uso de Auxiliares de Marca:

```

<form asp-controller="Account" asp-action="Register" method="post" class="form-hori
    <h4>Create a new account.</h4>
    <hr />
    <div asp-validation-summary="ValidationSummary.All" class="text-danger"></div>
    <div class="form-group">
        <label asp-for="Email" class="col-md-2 control-label"></label>
        <div class="col-md-10">
            <input asp-for="Email" class="form-control" />
            <span asp-validation-for="Email" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group">
        <label asp-for="Password" class="col-md-2 control-label"></label>
        <div class="col-md-10">
            <input asp-for="Password" class="form-control" />
            <span asp-validation-for="Password" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group">
        <label asp-for="ConfirmPassword" class="col-md-2 control-label"></label>
        <div class="col-md-10">
            <input asp-for="ConfirmPassword" class="form-control" />
            <span asp-validation-for="ConfirmPassword" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group">
        <div class="col-md-offset-2 col-md-10">
            <button type="submit" class="btn btn-default">Register</button>
        </div>
    </div>
</form>

```

A marcação é muito mais limpa e fácil de ler, editar e manter que a abordagem dos Auxiliares HTML. O código C# é reduzido ao mínimo que o servidor precisa conhecer. O editor do Visual Studio exibe a marcação direcionada por um Auxiliar de Marca em uma fonte diferenciada.

Considere o grupo *Email*:

```
<div class="form-group">
    <label asp-for="Email" class="col-md-2 control-label"></label>
    <div class="col-md-10">
        <input asp-for="Email" class="form-control" />
        <span asp-validation-for="Email" class="text-danger"></span>
    </div>
</div>
```

Cada um dos atributos "asp-" tem um valor "Email", mas "Email" não é uma cadeia de caracteres. Nesse contexto, "Email" é a propriedade da expressão do modelo C# para o `RegisterViewModel`.

O editor do Visual Studio ajuda você a escrever **toda** a marcação na abordagem do Auxiliar de Marca de formulário de registro, enquanto o Visual Studio não fornece nenhuma ajuda para a maioria do código na abordagem de Auxiliares HTML. [Suporte do IntelliSense para Auxiliares de Marca](#) apresenta detalhes sobre como trabalhar com Auxiliares de Marca no editor do Visual Studio.

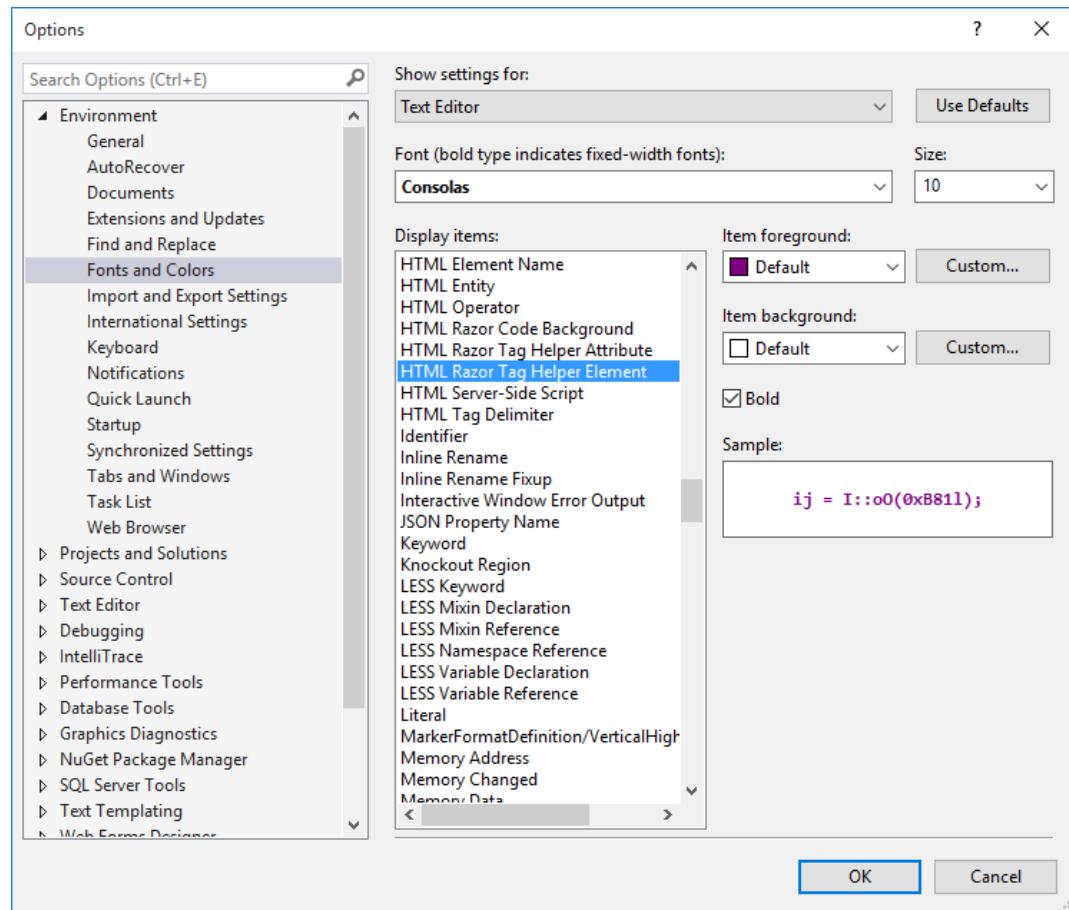
## Comparação entre Auxiliares de Marca e Controles de Servidor Web

- Os Auxiliares de Marca não têm o elemento ao qual estão associados; simplesmente participam da renderização do elemento e do conteúdo. Os [controles de Servidor Web](#) do ASP.NET são declarados e invocados em uma página.
- Os [controles de Servidor Web](#) têm um ciclo de vida não trivial que pode dificultar o desenvolvimento e a depuração.
- Os controles de Servidor Web permitem que você adicione a funcionalidade aos elementos DOM (Modelo de Objeto do Documento) do cliente usando um controle de cliente. Os Auxiliares de Marca não tem nenhum DOM.
- Os controles de Servidor Web incluem a detecção automática do navegador. Os Auxiliares de Marca não têm nenhum conhecimento sobre o navegador.
- Vários Auxiliares de Marca podem atuar no mesmo elemento (consulte [Evitando conflitos do Auxiliar de Marca](#)), embora normalmente não seja possível compor controles de Servidor Web.
- Os Auxiliares de Marca podem modificar a marca e o conteúdo de elementos HTML no escopo com o qual foram definidos, mas não modificam diretamente todo o resto em uma página. Os controles de Servidor Web têm um escopo menos específico e podem executar ações que afetam outras partes da página, permitindo efeitos colaterais não intencionais.
- Os controles de Servidor Web usam conversores de tipo para converter cadeias de caracteres em objetos. Com os Auxiliares de Marca, você trabalha nativamente no C# e, portanto, não precisa fazer a conversão de tipo.
- Os controles de Servidor Web usam [System.ComponentModel](#) para implementar o comportamento de componentes e controles em tempo de execução e em tempo de design. `System.ComponentModel` inclui as interfaces e as classes base para implementar atributos e conversores de tipo, associar a fontes de dados e licenciar componentes. Compare isso com os Auxiliares de Marca, que normalmente são derivados de

`TagHelper`, e a classe base `TagHelper` expõe apenas dois métodos, `Process` e `ProcessAsync`.

## Personalizando a fonte de elemento do Auxiliar de Marca

Personalize a fonte e a colorização em **Ferramentas > Opções > Ambiente > Fontes e Cores**:



## Recursos adicionais

- [Auxiliares de marca de autor](#)
- [Trabalhando com formulários](#)
- [TagHelperSamples no GitHub](#) contém amostras de Auxiliar de Marca para trabalhar com o Bootstrap.

# Auxiliares de marca de autor no ASP.NET Core

04/02/2019 • 27 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Introdução aos Auxiliares de Marca

Este tutorial fornece uma introdução à programação de auxiliares de marcação. [Introdução aos auxiliares de marcação](#) descreve os benefícios que os auxiliares de marcação fornecem.

Um auxiliar de marca é qualquer classe que implementa a interface `ITagHelper`. No entanto, quando cria um auxiliar de marca, você geralmente deriva de `TagHelper`, o que fornece acesso ao método `Process`.

1. Crie um novo projeto do ASP.NET Core chamado **AuthoringTagHelpers**. Você não precisará de autenticação para esse projeto.
2. Criar uma pasta para armazenar os auxiliares de marca chamados *TagHelpers*. A pasta *TagHelpers* pasta *não* é necessária, mas é uma convenção comum. Agora vamos começar a escrever alguns auxiliares de marca simples.

## Um auxiliar de marca mínimo

Nesta seção, você escreve um auxiliar de marca que atualiza uma marca de email. Por exemplo:

```
<email>Support</email>
```

O servidor usará nosso auxiliar de marca de email para converter essa marcação como a seguir:

```
<a href="mailto:Support@contoso.com">Support@contoso.com</a>
```

Ou seja, uma marca de âncora que torna isso um link de email. Talvez você deseje fazer isso se estiver escrevendo um mecanismo de blog e precisar que ele envie emails para o marketing, suporte e outros contatos, todos para o mesmo domínio.

1. Adicione a classe `EmailTagHelper` a seguir à pasta *TagHelpers*.

```
using Microsoft.AspNetCore.Razor.TagHelpers;
using System.Threading.Tasks;

namespace AuthoringTagHelpers.TagHelpers
{
    public class EmailTagHelper : TagHelper
    {
        public override void Process(TagHelperContext context, TagHelperOutput output)
        {
            output.TagName = "a";      // Replaces <email> with <a> tag
        }
    }
}
```

- Auxiliares de marcação usam uma convenção de nomenclatura que tem como alvo os elementos do nome da classe raiz (menos o `TagHelper` parte do nome de classe). Neste exemplo, o nome da raiz `EmailTagHelper` é `email` e, portanto, a marca `<email>` será direcionada. Essa convenção de nomenclatura deve funcionar para a maioria dos auxiliares de marcação, posteriormente, mostrarei como substituí-la.

- O `EmailTagHelper` classe deriva de `TagHelper`. O `TagHelper` classe fornece métodos e propriedades para gravar tag helpers.
- O método `Process` substituído controla o que o auxiliar de marca faz quando é executado. A classe `TagHelper` também fornece uma versão assíncrona (`ProcessAsync`) com os mesmos parâmetros.
- O parâmetro de contexto para `Process` (e `ProcessAsync`) contém informações associadas à execução da marca HTML atual.
- O parâmetro de saída para `Process` (e `ProcessAsync`) contém um elemento HTML com estado que representa a fonte original usada para gerar uma marca HTML e o conteúdo.
- Nossa nome de classe tem um sufixo `TagHelper`, que *não* é necessário, mas que é considerado uma convenção de melhor prática. Você pode declarar a classe como:

```
public class Email : TagHelper
```

2. Para disponibilizar a classe `EmailTagHelper` para todas as nossas exibições do Razor, adicione a diretiva `@addTagHelper` ao arquivo `Views/_ViewImports.cshtml`: [!code-html]

O código acima usa a sintaxe de curinga para especificar que todos os auxiliares de marca em nosso assembly estarão disponíveis. A primeira cadeia de caracteres após `@addTagHelper` especifica o auxiliar de marca a ser carregado (use "\*" para todos os auxiliares de marca) e a segunda cadeia de caracteres "AuthoringTagHelpers" especifica o assembly no qual o auxiliar de marca se encontra. Além disso, observe que a segunda linha insere os auxiliares de marca do ASP.NET Core MVC usando a sintaxe de curinga (esses auxiliares são abordados em [Introdução ao auxiliares de marcação](#)). É a diretiva `@addTagHelper` que disponibiliza o auxiliar de marca para a exibição do Razor. Como alternativa, você pode fornecer o FQN (nome totalmente qualificado) de um auxiliar de marca, conforme mostrado abaixo:

```
@using AuthoringTagHelpers  
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers  
@addTagHelper AuthoringTagHelpers.TagHelpers.EmailTagHelper, AuthoringTagHelpers
```

- Para adicionar um auxiliar de marca para uma exibição usando um FQN, primeiro adicione o FQN (`AuthoringTagHelpers.TagHelpers.EmailTagHelper`) e, em seguida, o **nome do assembly** (`AuthoringTagHelpers`, não necessariamente o `namespace`). A maioria dos desenvolvedores vão preferir usar a sintaxe de curinga. [Introdução ao tag helpers](#) apresenta detalhes sobre a sintaxe de adição, remoção, hierarquia e curinga do tag helper.

1. Atualize a marcação no arquivo `Views/Home/Contact.cshtml` com essas alterações:

```

@{
    ViewData["Title"] = "Contact";
}
<h2>@ViewData["Title"].</h2>
<h3>@ViewData["Message"]</h3>

<address>
    One Microsoft Way<br />
    Redmond, WA 98052<br />
    <abbr title="Phone">P:</abbr>
    425.555.0100
</address>

<address>
    <strong>Support:</strong><email>Support</email><br />
    <strong>Marketing:</strong><email>Marketing</email>
</address>

```

- Execute o aplicativo e use seu navegador favorito para exibir o código-fonte HTML para verificar se as marcas de email são substituídas pela marcação de âncora (por exemplo, `<a>Support</a>`). *Support* e *Marketing* são renderizados como links, mas não têm um atributo `href` para torná-los funcionais. Corrigiremos isso na próxima seção.

## SetAttribute e SetContent

Nesta seção, atualizaremos o `EmailTagHelper` para que ele crie uma marca de âncora válida para email. Vamos atualizá-lo para obter informações de uma exibição do Razor (na forma de um atributo `mail-to`) e usar isso na geração da âncora.

Atualize a classe `EmailTagHelper` com o seguinte:

```

public class EmailTagHelper : TagHelper
{
    private const string EmailDomain = "contoso.com";

    // Can be passed via <email mail-to="..." />.
    // PascalCase gets translated into kebab-case.
    public string MailTo { get; set; }

    public override void Process(TagHelperContext context, TagHelperOutput output)
    {
        output.TagName = "a";      // Replaces <email> with <a> tag

        var address = MailTo + "@" + EmailDomain;
        output.Attributes.SetAttribute("href", "mailto:" + address);
        output.Content.SetContent(address);
    }
}

```

- Nomes de classe e de propriedade na formatação Pascal Case para auxiliares de marcações são convertidos em **Kebab Case**. Portanto, para usar o atributo `MailTo`, você usará o equivalente de `<email mail-to="value"/>`.
- A última linha define o conteúdo concluído para nosso tag helper minimamente funcional.
- A linha realçada mostra a sintaxe para adicionar atributos:

```

public override void Process(TagHelperContext context, TagHelperOutput output)
{
    output.TagName = "a";      // Replaces <email> with <a> tag

    var address = MailTo + "@" + EmailDomain;
    output.Attributes.SetAttribute("href", "mailto:" + address);
    output.Content.SetContent(address);
}

```

Essa abordagem funciona para o atributo "href" como no momento, ele não existe na coleção de atributos. Você também pode usar o `output.Attributes.Add` para adicionar um atributo do tag helper ao final da coleção de atributos de marca.

- Atualize a marcação no arquivo `Views/Home/Contact.cshtml` com essas alterações: [!code-html]
- Execute o aplicativo e verifique se ele gera os links corretos.

#### NOTE

Se você pretende escrever o autofechamento da marca de email (`<email mail-to="Rick" />`), a saída final também é o autofechamento. Para permitir a capacidade de gravar a marca com uma marca de início (`<email mail-to="Rick">`), é necessário decorar a classe com o seguinte:

```

[HtmlTargetElement("email", TagStructure = TagStructure.WithoutEndTag)]
public class EmailVoidTagHelper : TagHelper
{
    private const string EmailDomain = "contoso.com";
    // Code removed for brevity

```

Com um auxiliar de marca da marca de email com autofechamento, a saída será

`<a href="mailto:Rick@contoso.com" />`. As marcas de âncora com autofechamento são um HTML inválido.

Portanto, não é recomendável criá-las, mas talvez criar um auxiliar de marca com autofechamento. Auxiliares de marca definem o tipo da propriedade `TagMode` após a leitura de uma marca.

#### ProcessAsync

Nesta seção, escreveremos um auxiliar de email assíncrono.

- Substitua a classe `EmailTagHelper` pelo seguinte código:

```

public class EmailTagHelper : TagHelper
{
    private const string EmailDomain = "contoso.com";
    public override async Task ProcessAsync(TagHelperContext context, TagHelperOutput output)
    {
        output.TagName = "a";                                // Replaces <email> with <a> tag
        var content = await output.GetChildContentAsync();
        var target = content.GetContent() + "@" + EmailDomain;
        output.Attributes.SetAttribute("href", "mailto:" + target);
        output.Content.SetContent(target);
    }
}

```

#### Observações:

- Essa versão usa o método `ProcessAsync` assíncrono. O `GetChildContentAsync` assíncrono retorna uma `Task` que contém o `TagHelperContent`.
- Use o parâmetro `output` para obter o conteúdo do elemento HTML.

2. Faça a alteração a seguir no arquivo *Views/Home/Contact.cshtml* para que o auxiliar de marca possa obter o email de destino.

```
@{
    ViewData["Title"] = "Contact";
}
<h2>@ViewData["Title"].</h2>
<h3>@ViewData["Message"]</h3>

<address>
    One Microsoft Way<br />
    Redmond, WA 98052<br />
    <abbr title="Phone">P:</abbr>
    425.555.0100
</address>

<address>
    <strong>Support:</strong><email>Support</email><br />
    <strong>Marketing:</strong><email>Marketing</email>
</address>
```

3. Execute o aplicativo e verifique se ele gera links de email válidos.

### **RemoveAll, PreContent.SetHtmlContent e PostContent.SetHtmlContent**

1. Adicione a classe `BoldTagHelper` a seguir à pasta *TagHelpers*.

```
using Microsoft.AspNetCore.Razor.TagHelpers;

namespace AuthoringTagHelpers.TagHelpers
{
    [HtmlTargetElement(Attributes = "bold")]
    public class BoldTagHelper : TagHelper
    {
        public override void Process(TagHelperContext context, TagHelperOutput output)
        {
            output.Attributes.RemoveAll("bold");
            output.PreContent.SetHtmlContent("<strong>");
            output.PostContent.SetHtmlContent("</strong>");
        }
    }
}
```

- O atributo `[HtmlTargetElement]` passa um parâmetro de atributo que especifica que qualquer elemento HTML que contenha um atributo HTML nomeado "bold" terá uma correspondência e que o método de substituição `Process` na classe será executado. Em nossa amostra, o método `Process` remove o atributo "bold" e envolve a marcação contida com `<strong></strong>`.
- Como você não deseja substituir o conteúdo de marca existente, você precisa escrever a marca `<strong>` de abertura com o método `PreContent.SetHtmlContent` e a marca `</strong>` de fechamento com o método `PostContent.SetHtmlContent`.

2. Modifique a exibição *About.cshtml* para que ela contenha um valor de atributo `bold`. O código completo é mostrado abaixo.

```

@{
    ViewData["Title"] = "About";
}
<h2>@ViewData["Title"].</h2>
<h3>@ViewData["Message"]</h3>

<p bold>Use this area to provide additional information.</p>

<bold> Is this bold?</bold>

```

3. Execute o aplicativo. Use seu navegador favorito para inspecionar a origem e verificar a marcação.

O `[HtmlTargetElement]` atributo acima se destina somente a marcação HTML que fornece um nome de atributo de "bold". O `<bold>` elemento não foi modificado pelo tag helper.

4. Comente a linha de atributo `[HtmlTargetElement]` e ela usará como padrão as marcações `<bold>` de direcionamento, ou seja, a marcação HTML do formato `<bold>`. Lembre-se de que a convenção de nomenclatura padrão fará a correspondência do nome da classe `BoldTagHelper` com as marcações `<bold>`.

5. Execute o aplicativo e verifique se a marca `<bold>` é processada pelo auxiliar de marca.

A decoração de uma classe com vários atributos `[HtmlTargetElement]` resulta em um OR lógico dos destinos. Por exemplo, o uso do código a seguir, uma marca de negrito ou um atributo de negrito terá uma correspondência.

```

[HtmlTargetElement("bold")]
[HtmlTargetElement(Attributes = "bold")]
public class BoldTagHelper : TagHelper
{
    public override void Process(TagHelperContext context, TagHelperOutput output)
    {
        output.Attributes.RemoveAll("bold");
        output.PreContent.SetHtmlContent("<strong>");
        output.PostContent.SetHtmlContent("</strong>");
    }
}

```

Quando vários atributos são adicionados à mesma instrução, o tempo de execução trata-os como um AND lógico. Por exemplo, no código abaixo, um elemento HTML precisa ser nomeado "bold" com um atributo nomeado "bold" (`<bold bold />`) para que haja a correspondência.

```
[HtmlTargetElement("bold", Attributes = "bold")]
```

Também use o `[HtmlTargetElement]` para alterar o nome do elemento de destino. Por exemplo, se você deseja que o `BoldTagHelper` seja direcionado a marcações `<MyBold>`, use o seguinte atributo:

```
[HtmlTargetElement("MyBold")]
```

## Passe um model para um tag helper

1. Adicionar uma pasta `models`.
2. Adicione a seguinte classe `WebsiteContext` à pasta `Models`:

```

using System;

namespace AuthoringTagHelpers.Models
{
    public class WebsiteContext
    {
        public Version Version { get; set; }
        public int CopyrightYear { get; set; }
        public bool Approved { get; set; }
        public int TagsToShow { get; set; }
    }
}

```

3. Adicione a classe `WebsiteInformationTagHelper` a seguir à pasta `TagHelpers`.

```

using System;
using AuthoringTagHelpers.Models;
using Microsoft.AspNetCore.Razor.TagHelpers;

namespace AuthoringTagHelpers.TagHelpers
{
    public class WebsiteInformationTagHelper : TagHelper
    {
        public WebsiteContext Info { get; set; }

        public override void Process(TagHelperContext context, TagHelperOutput output)
        {
            output.TagName = "section";
            output.Content.SetHtmlContent(
$@"<ul><li><strong>Version:</strong> {Info.Version}</li>
<li><strong>Copyright Year:</strong> {Info.CopyrightYear}</li>
<li><strong>Approved:</strong> {Info.Approved}</li>
<li><strong>Number of tags to show:</strong> {Info.TagsToShow}</li></ul>");
            output.TagMode = TagMode.StartTagAndEndTag;
        }
    }
}

```

- Conforme mencionado anteriormente, os auxiliares de marcações convertem nomes de classes e de propriedades dos auxiliares de marcações do C# na formatação Pascal Case em **Kebab Case**. Portanto, para usar o `WebsiteInformationTagHelper` no Razor, você escreverá `<website-information />`.
- Você não identifica de forma explícita o elemento de destino com o atributo `[HtmlTargetElement]`. Portanto, o padrão de `website-information` será o destino. Se você aplicou o seguinte atributo (observe que não está em kebab case, mas corresponde ao nome da classe):

```
[HtmlTargetElement("WebsiteInformation")]
```

A marcação `<website-information />` em Kebab Case não terá uma correspondência. Caso deseje usar o atributo `[HtmlTargetElement]`, use o kebab case, conforme mostrado abaixo:

```
[HtmlTargetElement("Website-Information")]
```

- Os elementos com autofechamento não têm nenhum conteúdo. Para este exemplo, a marcação do Razor usará uma marca com autofechamento, mas o auxiliar de marca criará um elemento `section` (que não tem autofechamento) e você escreve o conteúdo dentro do elemento `section`). Portanto, você precisa definir `TagMode` como `StartTagAndEndTag` para escrever a saída. Como alternativa,

você pode comentar a linha definindo `TagMode` e escrever a marcação com uma marca de fechamento. (A marcação de exemplo é fornecida mais adiante neste tutorial.)

- O `$` (cifrão) na seguinte linha usa uma [cadeia de caracteres interpolada](#):

```
$@"<ul><li><strong>Version:</strong> {Info.Version}</li>
```

4. Adicione a marcação a seguir à exibição *About.cshtml*. A marcação realçada exibe as informações do site.

```
@using AuthoringTagHelpers.Models  
{@  
    ViewData["Title"] = "About";  
    WebsiteContext webContext = new WebsiteContext {  
        Version = new Version(1, 3),  
        CopyrightYear = 1638,  
        Approved = true,  
        TagsToShow = 131 };  
}  
<h2>@ViewData["Title"].</h2>  
<h3>@ViewData["Message"]</h3>  
  
<p bold>Use this area to provide additional information.</p>  
  
<bold> Is this bold?</bold>  
  
<h3> web site info </h3>  
<website-information info="webContext" />
```

#### NOTE

Na marcação do Razor mostrada abaixo:

```
<website-information info="webContext" />
```

O Razor reconhece que o atributo `info` é uma classe, e não uma cadeia de caracteres, bem como que você deseja escrever o código C#. Qualquer atributo do auxiliar de marca que não seja uma cadeia de caracteres deve ser escrito sem o caractere `@`.

5. Execute o aplicativo e navegue para a exibição About sobre as informações do site.

#### NOTE

Você pode usar a seguinte marcação com uma marca de fechamento e remova a linha com `TagMode.StartTagAndEndTag` no tag helper:

```
<website-information info="new WebsiteContext {  
    Version = new Version(1, 3),  
    CopyrightYear = 1638,  
    Approved = true,  
    TagsToShow = 131 }" >  
</website-information>
```

## Tag helper condicional

O auxiliar de marca de condição renderiza a saída quando recebe um valor `true`.

1. Adicione a classe `ConditionTagHelper` a seguir à pasta `TagHelpers`.

```
using Microsoft.AspNetCore.Razor.TagHelpers;

namespace AuthoringTagHelpers.TagHelpers
{
    [HtmlTargetElement(Attributes = nameof(Condition))]
    public class ConditionTagHelper : TagHelper
    {
        public bool Condition { get; set; }

        public override void Process(TagHelperContext context, TagHelperOutput output)
        {
            if (!Condition)
            {
                output.SuppressOutput();
            }
        }
    }
}
```

2. Substitua o conteúdo do arquivo `Views/Home/Index.cshtml` pela seguinte marcação:

```
@using AuthoringTagHelpers.Models
@model WebsiteContext

 @{
     ViewData["Title"] = "Home Page";
 }

 <div>
     <h3>Information about our website (outdated):</h3>
     <Website-InforMation info="Model" />
     <div condition="Model.Approved">
         <p>
             This website has <strong surround="em">@Model.Approved</strong> been approved yet.
             Visit www.contoso.com for more information.
         </p>
     </div>
 </div>
```

3. Substitua o método `Index` no controlador `Home` pelo seguinte código:

```
public IActionResult Index(bool approved = false)
{
    return View(new WebsiteContext
    {
        Approved = approved,
        CopyrightYear = 2015,
        Version = new Version(1, 3, 3, 7),
        TagsToShow = 20
    });
}
```

4. Execute o aplicativo e navegue para a home page. A marcação no `div` condicional não será renderizada.

Acrescente a cadeia de caracteres de consulta `?approved=true` à URL (por exemplo,

`http://localhost:1235/Home/Index?approved=true`). `approved` é definido como verdadeiro e a marcação condicional será exibida.

## NOTE

Use o `nameof` operador para especificar o atributo de destino em vez de especificar uma cadeia de caracteres como você fez com o tag helper em negrito:

```
[HtmlTargetElement(Attributes = nameof(Condition))]
// [HtmlTargetElement(Attributes = "condition")]
public class ConditionTagHelper : TagHelper
{
    public bool Condition { get; set; }

    public override void Process(TagHelperContext context, TagHelperOutput output)
    {
        if (!Condition)
        {
            output.SuppressOutput();
        }
    }
}
```

O operador `nameof` protegerá o código, caso ele seja refatorado (recomendamos alterar o nome para `RedCondition`).

## Evitar conflitos de tag helper

Nesta seção, você escreve um par de tag helpers de vinculação automática. O primeiro substituirá a marcação que contém uma URL iniciada por HTTP para um HTML âncora marca que contém a mesma URL (e, portanto, resultando em um link de URL). O segundo fará o mesmo para uma URL começando com WWW.

Como esses dois auxiliares estão intimamente relacionados e você poderá refatorá-los no futuro, vamos mantê-los no mesmo arquivo.

1. Adicione a classe `AutoLinkerHttpTagHelper` a seguir à pasta `TagHelpers`.

```
[HtmlTargetElement("p")]
public class AutoLinkerHttpTagHelper : TagHelper
{
    public override async Task ProcessAsync(TagHelperContext context, TagHelperOutput output)
    {
        var childContent = await output.GetChildContentAsync();
        // Find URLs in the content and replace them with their anchor tag equivalent.
        output.Content.SetHtmlContent(Regex.Replace(
            childContent.GetContent(),
            @"\b(?:https?:\/\/)(\S+)\b",
            "<a target=\"_blank\" href=\"$0\">$0</a>")); // http link version}
    }
}
```

## NOTE

A classe `AutoLinkerHttpTagHelper` é direcionada a elementos `p` e usa o `Regex` para criar a âncora.

2. Adicione a seguinte marcação ao final do arquivo `Views/Home/Contact.cshtml`:

```

@{
    ViewData["Title"] = "Contact";
}
<h2>@ViewData["Title"].</h2>
<h3>@ViewData["Message"]</h3>

<address>
    One Microsoft Way<br />
    Redmond, WA 98052<br />
    <abbr title="Phone">P:</abbr>
    425.555.0100
</address>

<address>
    <strong>Support:</strong><email>Support</email><br />
    <strong>Marketing:</strong><email>Marketing</email>
</address>

<p>Visit us at http://docs.asp.net or at www.microsoft.com</p>

```

3. Execute o aplicativo e verifique se o tag helper renderiza a âncora corretamente.
4. Atualize a classe `AutoLinker` para incluir o `AutoLinkerWwwTagHelper` que converterá o texto www em uma marca de âncora que também contém o texto www original. O código atualizado é realçado abaixo:

```

[HtmlTargetElement("p")]
public class AutoLinkerHttpTagHelper : TagHelper
{
    public override async Task ProcessAsync(TagHelperContext context, TagHelperOutput output)
    {
        var childContent = await output.GetChildContentAsync();
        // Find URLs in the content and replace them with their anchor tag equivalent.
        output.Content.SetHtmlContent(Regex.Replace(
            childContent.GetContent(),
            @"\b(?:https?:\/\/)(\S+)\b",
            "<a target=\"_blank\" href=\"$0\">$0</a>")); // http link version}
    }
}

[HtmlTargetElement("p")]
public class AutoLinkerWwwTagHelper : TagHelper
{
    public override async Task ProcessAsync(TagHelperContext context, TagHelperOutput output)
    {
        var childContent = await output.GetChildContentAsync();
        // Find URLs in the content and replace them with their anchor tag equivalent.
        output.Content.SetHtmlContent(Regex.Replace(
            childContent.GetContent(),
            @"\b(www\.)\S+\b",
            "<a target=\"_blank\" href=\"http://$0\">$0</a>")); // www version
    }
}

```

5. Execute o aplicativo. Observe que o texto www é renderizado como um link, ao contrário do texto HTTP. Se você colocar um ponto de interrupção em ambas as classes, poderá ver que a classe do auxiliar de marca HTTP é executada primeiro. O problema é que a saída do auxiliar de marca é armazenada em cache e quando o auxiliar de marca WWW é executado, ele substitui a saída armazenada em cache do auxiliar de marca HTTP. Mais adiante no tutorial, veremos como controlar a ordem na qual os auxiliares de marca são executados. Corrigiremos o código com o seguinte:

```

public class AutoLinkerHttpTagHelper : TagHelper
{
    public override async Task ProcessAsync(TagHelperContext context, TagHelperOutput output)
    {
        var childContent = output.Content.IsModified ? output.Content.GetContent() :
            (await output.GetChildContentAsync()).GetContent();

        // Find URLs in the content and replace them with their anchor tag equivalent.
        output.Content.SetHtmlContent(Regex.Replace(
            childContent,
            @"\b(?:https?:\/\/)(\S+)\b",
            "<a target=\"_blank\" href=\"$0\">$0</a>"); // http link version}
    }
}

[HtmlTargetElement("p")]
public class AutoLinkerWwwTagHelper : TagHelper
{
    public override async Task ProcessAsync(TagHelperContext context, TagHelperOutput output)
    {
        var childContent = output.Content.IsModified ? output.Content.GetContent() :
            (await output.GetChildContentAsync()).GetContent();

        // Find URLs in the content and replace them with their anchor tag equivalent.
        output.Content.SetHtmlContent(Regex.Replace(
            childContent,
            @"\b(www\.).(\S+)\b",
            "<a target=\"_blank\" href=\"http://$0\">$0</a>"); // www version
    }
}

```

#### NOTE

Na primeira edição os tag helpers de vinculação automática, você obteve o conteúdo do destino com o código a seguir:

```
var childContent = await output.GetChildContentAsync();
```

Ou seja, você chama `GetChildContentAsync` usando a `TagHelperOutput` passada para o método `ProcessAsync`. Conforme mencionado anteriormente, como a saída é armazenada em cache, o último auxiliar de marca a ser executado vence. Você corrigiu o problema com o seguinte código:

```
var childContent = output.Content.IsModified ? output.Content.GetContent() :
    (await output.GetChildContentAsync()).GetContent();
```

O código acima verifica se o conteúdo foi modificado e, em caso afirmativo, ele obtém o conteúdo do buffer de saída.

6. Execute o aplicativo e verifique se os dois links funcionam conforme esperado. Embora possa parecer que nosso auxiliar de marca de vinculador automático está correto e completo, ele tem um problema sutil. Se o auxiliar de marca WWW for executado primeiro, os links www não estarão corretos. Atualize o código adicionando a sobrecarga `Order` para controlar a ordem em que a marca é executada. A propriedade `Order` determina a ordem de execução em relação aos outros auxiliares de marca direcionados ao mesmo elemento. O valor de ordem padrão é zero e as instâncias com valores mais baixos são executadas primeiro.

```
public class AutoLinkerHttpTagHelper : TagHelper
{
    // This filter must run before the AutoLinkerWwwTagHelper as it searches and replaces http and
    // the AutoLinkerWwwTagHelper adds http to the markup.
    public override int Order
    {
        get { return int.MinValue; }
    }
}
```

O código acima garante que o auxiliar de marca HTTP seja executado antes do auxiliar de marca WWW.

Altere `Order` para `MaxValue` e verifique se a marcação gerada para a marcação WWW está incorreta.

## Inspecionar e recuperar o conteúdo filho

Os tag helpers fornecem várias propriedades para recuperar o conteúdo.

- O resultado de `GetChildContentAsync` pode ser acrescentado ao `output.Content`.
- Inspecione o resultado de `GetChildContentAsync` com `GetContent`.
- Se você modificar `output.Content`, o corpo da TagHelper não será executado ou renderizado, a menos que você chame `GetChildContentAsync` como em nossa amostra de vinculador automático:

```
public class AutoLinkerHttpTagHelper : TagHelper
{
    public override async Task ProcessAsync(TagHelperContext context, TagHelperOutput output)
    {
        var childContent = output.Content.IsModified ? output.Content.GetContent() :
            (await output.GetChildContentAsync()).GetContent();

        // Find URLs in the content and replace them with their anchor tag equivalent.
        output.Content.SetHtmlContent(Regex.Replace(
            childContent,
            @"\b(?:https?:\/\/)(\S+)\b",
            "<a target=_blank" href=\"$0\">$0</a>"); // http link version)
    }
}
```

- Várias chamadas a `GetChildContentAsync` retornam o mesmo valor e não executam o corpo `TagHelper` novamente, a menos que você passe um parâmetro falso indicando para não usar o resultado armazenado em cache.

# Auxiliares de marca em formulários no ASP.NET Core

14/01/2019 • 28 minutes to read • [Edit Online](#)

Por [Rick Anderson](#), [Dave Paquette](#) e [Jerrie Pelser](#)

Este documento demonstra como é o trabalho com Formulários e os elementos HTML usados comumente em um Formulário. O elemento HTML [Formulário](#) fornece o mecanismo primário que os aplicativos Web usam para postar dados para o servidor. A maior parte deste documento descreve os [Auxiliares de marca](#) e como eles podem ajudar você a criar formulários HTML robustos de forma produtiva. É recomendável que você leia [Introdução ao auxiliares de marca](#) antes de ler este documento.

Em muitos casos, os Auxiliares HTML fornecem uma abordagem alternativa a um Auxiliar de Marca específico, mas é importante reconhecer que os Auxiliares de Marca não substituem os Auxiliares HTML e que não há um Auxiliar de Marca para cada Auxiliar HTML. Quando existe um Auxiliar HTML alternativo, ele é mencionado.

## O Auxiliar de marca de formulário

O Auxiliar de marca de [formulário](#):

- Gera o valor do atributo HTML `<FORM>` `action` para uma ação do controlador MVC ou uma rota nomeada
- Gera um [Token de verificação de solicitação](#) oculto para evitar a falsificação de solicitações entre sites (quando usado com o atributo `[ValidateAntiForgeryToken]` no método de ação HTTP Post)
- Fornece o atributo `asp-route-<Parameter Name>`, em que `<Parameter Name>` é adicionado aos valores de rota. Os parâmetros `routeValues` para `Html.BeginForm` e `Html.BeginRouteForm` fornecem funcionalidade semelhante.
- Tem uma alternativa de Auxiliar HTML `Html.BeginForm` e `Html.BeginRouteForm`

Amostra:

```
<form asp-controller="Demo" asp-action="Register" method="post">
    <!-- Input and Submit elements -->
</form>
```

O Auxiliar de marca de formulário acima gera o HTML a seguir:

```
<form method="post" action="/Demo/Register">
    <!-- Input and Submit elements -->
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

O tempo de execução do MVC gera o valor do atributo `action` dos atributos `asp-controller` e `asp-action` do Auxiliar de marca de formulário. O Auxiliar de marca de formulário também gera um [Token de verificação de solicitação](#) oculto para evitar a falsificação de solicitações entre sites (quando usado com o atributo `[ValidateAntiForgeryToken]` no método de ação HTTP Post). É difícil proteger um Formulário HTML puro contra falsificação de solicitações entre sites e o Auxiliar de marca de formulário fornece este serviço para você.

### Usando uma rota nomeada

O atributo do Auxiliar de Marca `asp-route` também pode gerar a marcação para o atributo HTML `action`. Um

aplicativo com uma [rota](#) chamada `register` poderia usar a seguinte marcação para a página de registro:

```
<form asp-route="register" method="post">
    <!-- Input and Submit elements -->
</form>
```

Muitas das exibições na pasta *Modos de Exibição/Conta* (gerada quando você cria um novo aplicativo Web com *Contas de usuário individuais*) contêm o atributo [asp-route-returnurl](#):

```
<form asp-controller="Account" asp-action="Login"
      asp-route-returnurl="@ViewData["ReturnUrl"]"
      method="post" class="form-horizontal" role="form">
```

#### NOTE

Com os modelos internos, `returnUrl` só é preenchido automaticamente quando você tenta acessar um recurso autorizado, mas não está autenticado ou autorizado. Quando você tenta fazer um acesso não autorizado, o middleware de segurança o redireciona para a página de logon com o `returnUrl` definido.

## O auxiliar de marca de entrada

O Auxiliar de marca de entrada associa um elemento HTML `<input>` a uma expressão de modelo em sua exibição do Razor.

Sintaxe:

```
<input asp-for="<Expression Name>" />
```

O auxiliar de marca de entrada:

- Gera os atributos HTML `id` e `name` para o nome da expressão especificada no atributo `asp-for`.  
`asp-for="Property1.Property2"` equivale a `m => m.Property1.Property2`. O nome da expressão é o que é usado para o valor do atributo `asp-for`. Consulte a seção [Nomes de expressão](#) para obter informações adicionais.
- Define o valor do atributo HTML `type` com base nos atributos de tipo de modelo e [anotação de dados](#) aplicados à propriedade de modelo
- O valor do atributo HTML `type` não será substituído quando um for especificado
- Gera atributos de validação [HTML5](#) de atributos de [anotação de dados](#) aplicados a propriedades de modelo
- Tem uma sobreposição de recursos de Auxiliar HTML com `Html.TextBoxFor` e `Html.EditorFor`. Consulte a seção **Alternativas de Auxiliar HTML ao Auxiliar de marca de entrada** para obter detalhes.
- Fornece tipagem forte. Se o nome da propriedade for alterado e você não atualizar o Auxiliar de marca, você verá um erro semelhante ao seguinte:

An error occurred during the compilation of a resource required to process this request. Please review the following specific error details and modify your source code appropriately.

Type expected

'RegisterViewModel' does not contain a definition for 'Email' and no extension method 'Email' accepting a first argument of type 'RegisterViewModel' could be found (are you missing a using directive or an assembly reference?)

O Auxiliar de marca `Input` define o atributo HTML `type` com base no tipo .NET. A tabela a seguir lista alguns tipos .NET comuns e o tipo HTML gerado (não estão listados todos os tipos .NET).

TIPO .NET	TIPO DE ENTRADA
Bool	<code>type="checkbox"</code>
Cadeia de Caracteres	<code>type="text"</code>
DateTime	<code>type="datetime-local"</code>
Byte	<code>type="number"</code>
int	<code>type="number"</code>
Single e Double	<code>type="number"</code>

A tabela a seguir mostra alguns atributos de [anotações de dados](#) comuns que o auxiliar de marca de entrada mapeará para tipos de entrada específicos (não são listados todos os atributos de validação):

ATRIBUTO	TIPO DE ENTRADA
<code>[EmailAddress]</code>	<code>type="email"</code>
<code>[Url]</code>	<code>type="url"</code>
<code>[HiddenInput]</code>	<code>type="hidden"</code>
<code>[Phone]</code>	<code>type="tel"</code>
<code>[DataType(DataType.Password)]</code>	<code>type="password"</code>
<code>[DataType(DataType.Date)]</code>	<code>type="date"</code>
<code>[DataType(DataType.Time)]</code>	<code>type="time"</code>

Amostra:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}

```

```

@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterInput" method="post">
    Email: <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    <button type="submit">Register</button>
</form>

```

O código acima gera o seguinte HTML:

```

<form method="post" action="/Demo/RegisterInput">
    Email:
    <input type="email" data-val="true"
           data-val-email="The Email Address field is not a valid email address."
           data-val-required="The Email Address field is required."
           id="Email" name="Email" value="" /> <br>
    Password:
    <input type="password" data-val="true"
           data-val-required="The Password field is required."
           id="Password" name="Password" /><br>
    <button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

As anotações de dados aplicadas às propriedades `Email` e `Password` geram metadados no modelo. O Auxiliar de marca de entrada consome os metadados do modelo e produz atributos HTML5 `data-val-*` (consulte [Validação de modelo](#)). Esses atributos descrevem os validadores a serem anexados aos campos de entrada. Isso fornece validação de [jQuery](#) e HTML5 discreto. Os atributos discretos têm o formato `data-val-rule="Error Message"`, em que a regra é o nome da regra de validação (como `data-val-required`, `data-val-email`, `data-val-maxlength` etc.). Se uma mensagem de erro for fornecida no atributo, ela será exibida como o valor para o atributo `data-val-rule`. Também há atributos do formulário `data-val-ruleName-argumentName="argumentValue"` que fornecem detalhes adicionais sobre a regra, por exemplo, `data-val-maxlength-max="1024"`.

## Alternativas de Auxiliar HTML ao Auxiliar de marca de entrada

`Html.TextBox`, `Html.TextBoxFor`, `Html.Editor` e `Html.EditorFor` têm recursos que se sobrepõem aos do Auxiliar de marca de entrada. O Auxiliar de marca de entrada define automaticamente o atributo `type`; `Html.TextBox` e `Html.TextBoxFor` não o fazem. `Html.Editor` e `Html.EditorFor` manipulam coleções, objetos complexos e modelos; o Auxiliar de marca de entrada não o faz. O Auxiliar de marca de entrada, `Html.EditorFor` e `Html.TextBoxFor` são fortemente tipados (eles usam expressões lambda); `Html.TextBox` e `Html.Editor` não usam (eles usam nomes de expressão).

## HtmlAttributes

`@Html.Editor()` e `@Html.EditorFor()` usam uma entrada `ViewDataDictionary` especial chamada `htmlAttributes` ao executar seus modelos padrão. Esse comportamento pode ser aumentado usando parâmetros `additional ViewData`. A chave "htmlAttributes" diferencia maiúsculas de minúsculas. A chave "htmlAttributes" é tratada de forma semelhante ao objeto `htmlAttributes` passado para auxiliares de entrada como `@Html.TextBox()`.

```
@Html.EditorFor(model => model.YourProperty,  
new { htmlAttributes = new { @class="myCssClass", style="Width:100px" } })
```

## Nomes de expressão

O valor do atributo `asp-for` é um `ModelExpression` e o lado direito de uma expressão lambda. Portanto, `asp-for="Property1"` se torna `m => m.Property1` no código gerado e é por isso você não precisa colocar o prefixo `Model`. Você pode usar o caractere "@" para iniciar uma expressão embutida e mover para antes de `m`:

```
@{  
    var joe = "Joe";  
}  
<input asp-for="@joe" />
```

Gera o seguinte:

```
<input type="text" id="joe" name="joe" value="Joe" />
```

Com propriedades de coleção, `asp-for="CollectionProperty[23].Member"` gera o mesmo nome que `asp-for="CollectionProperty[i].Member"` quando `i` tem o valor `23`.

Quando o ASP.NET Core MVC calcula o valor de `ModelExpression`, ele inspeciona várias fontes, inclusive o `ModelState`. Considere o `<input type="text" asp-for="@Name" />`. O atributo `value` calculado é o primeiro valor não nulo:

- Da entrada de `ModelState` com a chave "Name".
- Do resultado da expressão `Model.Name`.

## Navegando para propriedades filho

Você também pode navegar para propriedades filho usando o caminho da propriedade do modelo de exibição. Considere uma classe de modelo mais complexa que contém uma propriedade `Address` filho.

```
public class AddressViewModel  
{  
    public string AddressLine1 { get; set; }  
}
```

```
public class RegisterAddressViewModel  
{  
    public string Email { get; set; }  
  
    [DataType(DataType.Password)]  
    public string Password { get; set; }  
  
    public AddressViewModel Address { get; set; }  
}
```

Na exibição, associamos a `Address.AddressLine1`:

```
@model RegisterAddressViewModel

<form asp-controller="Demo" asp-action="RegisterAddress" method="post">
    Email: <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    Address: <input asp-for="Address.AddressLine1" /><br />
    <button type="submit">Register</button>
</form>
```

O HTML a seguir é gerado para `Address.AddressLine1`:

```
<input type="text" id="Address_AddressLine1" name="Address.AddressLine1" value="" />
```

## Nomes de expressão e coleções

Exemplo, um modelo que contém uma matriz de `Colors`:

```
public class Person
{
    public List<string> Colors { get; set; }

    public int Age { get; set; }
}
```

O método de ação:

```
public IActionResult Edit(int id, int colorIndex)
{
    ViewData["Index"] = colorIndex;
    return View(GetPerson(id));
}
```

O Razor a seguir mostra como você acessa um elemento `Color` específico:

```
@model Person
 @{
     var index = (int)ViewData["index"];
 }

<form asp-controller="ToDo" asp-action="Edit" method="post">
    @Html.EditorFor(m => m.Colors[index])
    <label asp-for="Age"></label>
    <input asp-for="Age" /><br />
    <button type="submit">Post</button>
</form>
```

O modelo `Views/Shared/EditorTemplates/String.cshtml`:

```
@model string

<label asp-for="@Model"></label>
<input asp-for="@Model" /> <br />
```

Exemplo usando `List<T>`:

```

public class ToDoItem
{
    public string Name { get; set; }

    public bool IsDone { get; set; }
}

```

O Razor a seguir mostra como iterar em uma coleção:

```

@model List<ToDoItem>

<form asp-controller="ToDo" asp-action="Edit" method="post">
    <table>
        <tr> <th>Name</th> <th>Is Done</th> </tr>

        @for (int i = 0; i < Model.Count; i++)
        {
            <tr>
                @Html.EditorFor(model => model[i])
            </tr>
        }

    </table>
    <button type="submit">Save</button>
</form>

```

O modelo *Views/Shared/EditorTemplates/ToDoItem.cshtml*:

```

@model ToDoItem

<td>
    <label asp-for="@Model.Name"></label>
    @Html.DisplayFor(model => model.Name)
</td>
<td>
    <input asp-for="@Model.IsDone" />
</td>

/*
This template replaces the following Razor which evaluates the indexer three times.
<td>
    <label asp-for="@Model[i].Name"></label>
    @Html.DisplayFor(model => model[i].Name)
</td>
<td>
    <input asp-for="@Model[i].IsDone" />
</td>
*/

```

`foreach` deve ser usado, se possível, quando o valor está prestes a ser usado em um contexto equivalente `asp-for` ou `Html.DisplayFor`. Em geral, `for` é melhor do que `foreach` (se o cenário permitir) porque não é necessário alocar um enumerador; no entanto, avaliar um indexador em uma expressão LINQ pode ser caro, o que deve ser minimizado.

#### NOTE

O código de exemplo comentado acima mostra como você substituiria a expressão lambda pelo operador `@` para acessar cada `ToDoItem` na lista.

## Auxiliar de marca de área de texto

O auxiliar de marca `Textarea Tag Helper` é semelhante ao Auxiliar de marca de entrada.

- Gera os atributos `id` e `name`, bem como os atributos de validação de dados do modelo para um elemento `<textarea>`.
- Fornece tipagem forte.
- Alternativa de Auxiliar HTML: `Html.TextAreaFor`

Amostra:

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class DescriptionViewModel
    {
        [MinLength(5)]
        [MaxLength(1024)]
        public string Description { get; set; }
    }
}
```

```
@model DescriptionViewModel

<form asp-controller="Demo" asp-action="RegisterTextArea" method="post">
    <textarea asp-for="Description"></textarea>
    <button type="submit">Test</button>
</form>
```

O HTML a seguir é gerado:

```
<form method="post" action="/Demo/RegisterTextArea">
    <textarea data-val="true"
              data-val-maxlength="The field Description must be a string or array type with a maximum length of
              &#x27;1024&#x27;;."
              data-val-maxlength-max="1024"
              data-val-minlength="The field Description must be a string or array type with a minimum length of
              &#x27;5&#x27;;."
              data-val-minlength-min="5"
              id="Description" name="Description">
    </textarea>
    <button type="submit">Test</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## O auxiliar de marca de rótulo

- Gera a legenda do rótulo e o atributo `for` em um elemento para um nome de expressão
- Alternativa de Auxiliar HTML: `Html.LabelFor`

O **Label Tag Helper** fornece os seguintes benefícios em comparação com um elemento de rótulo HTML puro:

- Você obtém automaticamente o valor do rótulo descritivo do atributo `Display`. O nome de exibição desejado pode mudar com o tempo e a combinação do atributo `Display` e do Auxiliar de Marca de Rótulo aplicará `Display` em qualquer lugar em que for usado.
- Menos marcação no código-fonte
- Tipagem forte com a propriedade de modelo.

Amostra:

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class SimpleViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }
    }
}
```

```
@model SimpleViewModel

<form asp-controller="Demo" asp-action="RegisterLabel" method="post">
    <label asp-for="Email"></label>
    <input asp-for="Email" /> <br />
</form>
```

O HTML a seguir é gerado para o elemento `<label>`:

```
<label for="Email">Email Address</label>
```

O Auxiliar de marca de rótulo gerou o valor do atributo `for` de "Email", que é a ID associada ao elemento `<input>`. Auxiliares de marca geram elementos `id` e `for` consistentes para que eles possam ser associados corretamente. A legenda neste exemplo é proveniente do atributo `Display`. Se o modelo não contivesse um atributo `Display`, a legenda seria o nome da propriedade da expressão.

## Os auxiliares de marca de validação

Há dois auxiliares de marca de validação. O **Validation Message Tag Helper** (que exibe uma mensagem de validação para uma única propriedade em seu modelo) e o **Validation Summary Tag Helper** (que exibe um resumo dos erros de validação). O **Input Tag Helper** adiciona atributos de validação do lado do cliente HTML5 para elementos de entrada baseados em atributos de anotação de dados em suas classes de modelo. A validação também é executada no servidor. O Auxiliar de marca de validação exibe essas mensagens de erro quando ocorre um erro de validação.

### O Auxiliar de marca de mensagem de validação

- Adiciona o atributo `data-valmsg-for="property"` [HTML5](#) ao elemento `span`, que anexa as mensagens de erro de validação no campo de entrada da propriedade do modelo especificado. Quando ocorre um erro de validação do lado do cliente, [jQuery](#) exibe a mensagem de erro no elemento `<span>`.

- A validação também é feita no servidor. Os clientes poderão ter o JavaScript desabilitado e parte da validação só pode ser feita no lado do servidor.

- Alternativa de Auxiliar HTML: `@Html.ValidationMessageFor`

O `Validation Message Tag Helper` é usado com o atributo `asp-validation-for` em um elemento HTML `span`.

```
<span asp-validation-for="Email"></span>
```

O Auxiliar de marca de mensagem de validação gerará o HTML a seguir:

```
<span class="field-validation-valid"
      data-valmsg-for="Email"
      data-valmsg-replace="true"></span>
```

Geralmente, você usa o `Validation Message Tag Helper` após um Auxiliar de marca `Input` para a mesma propriedade. Fazer isso exibe as mensagens de erro de validação próximo à entrada que causou o erro.

#### NOTE

É necessário ter uma exibição com as referências de script `jQuery` e JavaScript corretas em vigor para a validação do lado do cliente. Consulte [Validação de Modelo](#) para obter mais informações.

Quando ocorre um erro de validação do lado do servidor (por exemplo, quando você tem validação do lado do servidor personalizada ou a validação do lado do cliente está desabilitada), o MVC coloca essa mensagem de erro como o corpo do elemento `<span>`.

```
<span class="field-validation-error" data-valmsg-for="Email"
      data-valmsg-replace="true">
    The Email Address field is required.
</span>
```

#### Auxiliar de marca de resumo de validação

- Tem como alvo elementos `<div>` com o atributo `asp-validation-summary`
- Alternativa de Auxiliar HTML: `@Html.ValidationSummary`

O `Validation Summary Tag Helper` é usado para exibir um resumo das mensagens de validação. O valor do atributo `asp-validation-summary` pode ser qualquer um dos seguintes:

ASP-VALIDATION-SUMMARY	MENSAGENS DE VALIDAÇÃO EXIBIDAS
<code>ValidationSummary.All</code>	Nível da propriedade e do modelo
<code>ValidationSummary.ModelOnly</code>	Modelo
<code>ValidationSummary.None</code>	Nenhum

#### Amostra

No exemplo a seguir, o modelo de dados é decorado com atributos `DataAnnotation`, o que gera mensagens de erro de validação no elemento `<input>`. Quando ocorre um erro de validação, o Auxiliar de marca de validação exibe a mensagem de erro:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}

```

```

@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterValidation" method="post">
    <div asp-validation-summary="ModelOnly"></div>
    Email: <input asp-for="Email" /> <br />
    <span asp-validation-for="Email"></span><br />
    Password: <input asp-for="Password" /><br />
    <span asp-validation-for="Password"></span><br />
    <button type="submit">Register</button>
</form>

```

O código HTML gerado (quando o modelo é válido):

```

<form action="/DemoReg/Register" method="post">
    <div class="validation-summary-valid" data-valmsg-summary="true">
        <ul><li style="display:none"></li></ul></div>
    Email: <input name="Email" id="Email" type="email" value="" data-val-required="The Email field is required." data-val-email="The Email field is not a valid email address." data-val="true"> <br />
    <span class="field-validation-valid" data-valmsg-replace="true" data-valmsg-for="Email"></span><br />
    Password: <input name="Password" id="Password" type="password" data-val-required="The Password field is required." data-val="true"><br />
    <span class="field-validation-valid" data-valmsg-replace="true" data-valmsg-for="Password"></span><br />
    <button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## Auxiliar de Marca de Seleção

- Gera `select` e os elementos `option` associados para as propriedades do modelo.
- Tem uma alternativa de Auxiliar HTML `Html.DropDownListFor` e `Html.ListBoxFor`

O `Select Tag Helper` `asp-for` especifica o nome da propriedade do modelo para o elemento `select` e `asp-items` especifica os elementos `option`. Por exemplo:

```

<select asp-for="Country" asp-items="Model.Countries"></select>

```

Amostra:

```

using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace FormsTagHelper.ViewModels
{
    public class CountryViewModel
    {
        public string Country { get; set; }

        public List<SelectListItem> Countries { get; } = new List<SelectListItem>
        {
            new SelectListItem { Value = "MX", Text = "Mexico" },
            new SelectListItem { Value = "CA", Text = "Canada" },
            new SelectListItem { Value = "US", Text = "USA" },
        };
    }
}

```

O método `Index` inicializa o `CountryViewModel`, define o país selecionado e o transmite para a exibição `Index`.

```

public IActionResult Index()
{
    var model = new CountryViewModel();
    model.Country = "CA";
    return View(model);
}

```

O método `Index` HTTP POST exibe a seleção:

```

[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Index(CountryViewModel model)
{
    if (ModelState.IsValid)
    {
        var msg = model.Country + " selected";
        return RedirectToAction("IndexSuccess", new { message = msg });
    }

    // If we got this far, something failed; redisplay form.
    return View(model);
}

```

A exibição `Index`:

```

@model CountryViewModel

<form asp-controller="Home" asp-action="Index" method="post">
    <select asp-for="Country" asp-items="Model.Countries"></select>
    <br /><button type="submit">Register</button>
</form>

```

Que gera o seguinte HTML (com "CA" selecionado):

```

<form method="post" action="/">
    <select id="Country" name="Country">
        <option value="MX">Mexico</option>
        <option selected="selected" value="CA">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## NOTE

Não é recomendável usar `ViewBag` ou  `ViewData` com o Auxiliar de Marca de Seleção. Um modelo de exibição é mais robusto para fornecer metadados MVC e, geralmente, menos problemático.

O valor do atributo `asp-for` é um caso especial e não requer um prefixo `Model`, os outros atributos do Auxiliar de marca requerem (como `asp-items`)

```
<select asp-for="Country" asp-items="Model.Countries"></select>
```

## Associação de enumeração

Geralmente, é conveniente usar `<select>` com uma propriedade `enum` e gerar os elementos `SelectListItem` dos valores `enum`.

Amostra:

```

public class CountryEnumViewModel
{
    public CountryEnum EnumCountry { get; set; }
}

```

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public enum CountryEnum
    {
        [Display(Name = "United Mexican States")]
        Mexico,
        [Display(Name = "United States of America")]
        USA,
        Canada,
        France,
        Germany,
        Spain
    }
}

```

O método `GetEnumSelectList` gera um objeto `selectList` para uma enumeração.

```

@model CountryEnumViewModel

<form asp-controller="Home" asp-action="IndexEnum" method="post">
    <select asp-for="EnumCountry"
        asp-items="Html.GetEnumSelectList<CountryEnum>()">
    </select>
    <br /><button type="submit">Register</button>
</form>

```

É possível decorar sua lista de enumeradores com o atributo `Display` para obter uma interface do usuário mais rica:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public enum CountryEnum
    {
        [Display(Name = "United Mexican States")]
        Mexico,
        [Display(Name = "United States of America")]
        USA,
        Canada,
        France,
        Germany,
        Spain
    }
}

```

O HTML a seguir é gerado:

```

<form method="post" action="/Home/IndexEnum">
    <select data-val="true" data-val-required="The EnumCountry field is required."
        id="EnumCountry" name="EnumCountry">
        <option value="0">United Mexican States</option>
        <option value="1">United States of America</option>
        <option value="2">Canada</option>
        <option value="3">France</option>
        <option value="4">Germany</option>
        <option selected="selected" value="5">Spain</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## Grupo de opções

O elemento HTML `<optgroup>` é gerado quando o modelo de exibição contém um ou mais objetos `SelectListGroup`.

O `CountryViewModelGroup` agrupa os elementos `SelectListItem` nos grupos "América do Norte" e "Europa":

```

public class CountryViewModelGroup
{
    public CountryViewModelGroup()
    {
        var NorthAmericaGroup = new SelectListGroup { Name = "North America" };
        var EuropeGroup = new SelectListGroup { Name = "Europe" };

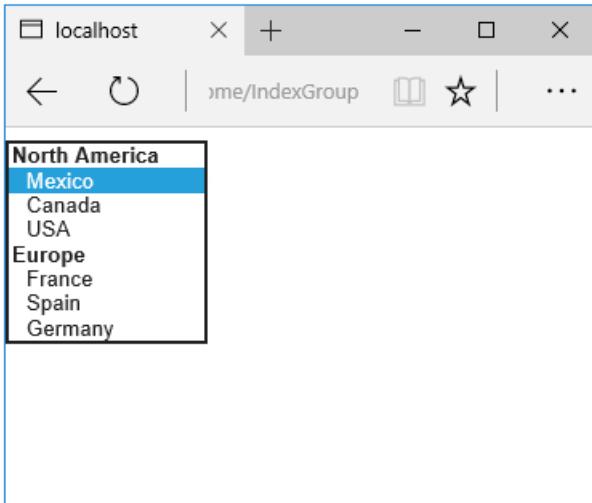
        Countries = new List<SelectListItem>
        {
            new SelectListItem
            {
                Value = "MEX",
                Text = "Mexico",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "CAN",
                Text = "Canada",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "US",
                Text = "USA",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "FR",
                Text = "France",
                Group = EuropeGroup
            },
            new SelectListItem
            {
                Value = "ES",
                Text = "Spain",
                Group = EuropeGroup
            },
            new SelectListItem
            {
                Value = "DE",
                Text = "Germany",
                Group = EuropeGroup
            }
        };
    }

    public string Country { get; set; }

    public List<SelectListItem> Countries { get; }
}

```

Os dois grupos são mostrados abaixo:



O HTML gerado:

```
<form method="post" action="/Home/IndexGroup">
    <select id="Country" name="Country">
        <optgroup label="North America">
            <option value="MEX">Mexico</option>
            <option value="CAN">Canada</option>
            <option value="US">USA</option>
        </optgroup>
        <optgroup label="Europe">
            <option value="FR">France</option>
            <option value="ES">Spain</option>
            <option value="DE">Germany</option>
        </optgroup>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="removed for brevity" />
</form>
```

## Seleção múltipla

O Auxiliar de Marca de Seleção gerará automaticamente o atributo `multiple = "multiple"` se a propriedade especificada no atributo `asp-for` for um `IEnumerable`. Por exemplo, considerando o seguinte modelo:

```
using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace FormsTagHelper.ViewModels
{
    public class CountryViewModel : IEnumerable
    {
        public IEnumerable<string> CountryCodes { get; set; }

        public List<SelectListItem> Countries { get; } = new List<SelectListItem>
        {
            new SelectListItem { Value = "MX", Text = "Mexico" },
            new SelectListItem { Value = "CA", Text = "Canada" },
            new SelectListItem { Value = "US", Text = "USA" },
            new SelectListItem { Value = "FR", Text = "France" },
            new SelectListItem { Value = "ES", Text = "Spain" },
            new SelectListItem { Value = "DE", Text = "Germany" }
        };
    }
}
```

Com a seguinte exibição:

```
@model CountryViewModelIEnumarable

<form asp-controller="Home" asp-action="IndexMultiSelect" method="post">
    <select asp-for="CountryCodes" asp-items="Model.Countries"></select>
    <br /><button type="submit">Register</button>
</form>
```

Gera o seguinte HTML:

```
<form method="post" action="/Home/IndexMultiSelect">
    <select id="CountryCodes"
        multiple="multiple"
        name="CountryCodes"><option value="MX">Mexico</option>
    <option value="CA">Canada</option>
    <option value="US">USA</option>
    <option value="FR">France</option>
    <option value="ES">Spain</option>
    <option value="DE">Germany</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## Nenhuma seleção

Se acabar usando a opção "não especificado" em várias páginas, você poderá criar um modelo para eliminar o HTML de repetição:

```
@model CountryViewModel

<form asp-controller="Home" asp-action="IndexEmpty" method="post">
    @Html.EditorForModel()
    <br /><button type="submit">Register</button>
</form>
```

O modelo *Views/Shared/EditorTemplates/CountryViewModel.cshtml*:

```
@model CountryViewModel

<select asp-for="Country" asp-items="Model.Countries">
    <option value="">--none--</option>
</select>
```

O acréscimo de elementos HTML `<option>` não está limitado ao caso de *Nenhuma seleção*. Por exemplo, o seguinte método de ação e exibição gerarão HTML semelhante ao código acima:

```
public IActionResult IndexOption(int id)
{
    var model = new CountryViewModel();
    model.Country = "CA";
    return View(model);
}
```

```
@model CountryViewModel

<form asp-controller="Home" asp-action="IndexEmpty" method="post">
    <select asp-for="Country">
        <option value="">&lt;none&gt;</option>
        <option value="MX">Mexico</option>
        <option value="CA">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
</form>
```

O elemento `<option>` correto será selecionado (contém o atributo `selected="selected"`) dependendo do valor atual de `Country`.

```
<form method="post" action="/Home/IndexEmpty">
    <select id="Country" name="Country">
        <option value="">&lt;none&gt;</option>
        <option value="MX">Mexico</option>
        <option value="CA" selected="selected">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## Recursos adicionais

- [Auxiliares de Marca no ASP.NET Core](#)
- [Elemento de formulário HTML](#)
- [Token de verificação de solicitação](#)
- [Model binding no ASP.NET Core](#)
- [Validação de modelo no ASP.NET Core MVC](#)
- [Interface IAttributeAdapter](#)
- [Snippets de código para este documento](#)

# Componentes do Auxiliar de Marca no ASP.NET Core

30/10/2018 • 10 minutes to read • [Edit Online](#)

Por [Scott Addie](#) e [Fiyaz Bin Hasan](#)

Um Componente do Auxiliar de Marca é um Auxiliar de Marca que permite que você modifique ou adicione condicionalmente elementos HTML de código do lado do servidor. Esse recurso está disponível no ASP.NET Core 2.0 ou posterior.

O ASP.NET Core inclui dois Componentes de Auxiliar de Marca internos: `head` e `body`. Eles estão localizados no namespace [Microsoft.AspNetCore.Mvc.Razor.TagHelpers](#) e podem ser usados no MVC e no Razor Pages.

Componentes do Auxiliar de Marca não requerem registro com o aplicativo em `_ViewImports.cshtml`.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Casos de uso

Dois casos de uso comum dos Componentes do Auxiliar de Marca incluem:

1. [Injetar um `<link>` no `<head>`](#).
2. [Injetar um `<script>` no `<body>`](#).

As seções a seguir descrevem esses casos de uso.

### Injetar no elemento HTML `head`

Dentro do elemento HTML `<head>`, os arquivos CSS são geralmente importados com o elemento HTML `<link>`.

O código a seguir injeta um elemento `<link>` no elemento `<head>` usando o Componente do Auxiliar de Marca `head`:

```
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Razor.TagHelpers;

namespace RazorPagesSample.TagHelpers
{
    public class AddressStyleTagHelperComponent : TagHelperComponent
    {
        private readonly string _style =
            @"<link rel=""stylesheet"" href="""/css/address.css"" />";

        public override int Order => 1;

        public override Task ProcessAsync(TagHelperContext context,
                                         TagHelperOutput output)
        {
            if (string.Equals(context.TagName, "head",
                              StringComparison.OrdinalIgnoreCase))
            {
                output.PostContent.AppendHtml(_style);
            }

            return Task.CompletedTask;
        }
    }
}
```

No código anterior:

- `AddressStyleTagHelperComponent` implementa [TagHelperComponent](#). A abstração:
  - Permite a inicialização da classe com um `TagHelperContext`.
  - Habilita o uso de Componentes do Auxiliar de Marca para adicionar ou modificar elementos HTML.
- A propriedade `Order` define a ordem na qual os Componentes são renderizados. `Order` é necessário quando há vários usos de Componentes do Auxiliar de Marca em um aplicativo.
- `ProcessAsync` compara o valor da propriedade `TagName` do contexto de execução com `head`. Se a comparação for avaliada como verdadeira, o conteúdo do campo `_style` será injetado no elemento HTML `<head>`.

### Injetar no elemento HTML `body`

O Componente do Auxiliar de Marca `body` pode injetar um elemento `<script>` no elemento `<body>`. O código a seguir demonstra essa técnica:

```
using System;
using System.IO;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Razor.TagHelpers;

namespace RazorPagesSample.TagHelpers
{
    public class AddressScriptTagHelperComponent : TagHelperComponent
    {
        public override int Order => 2;

        public override async Task ProcessAsync(TagHelperContext context,
                                                TagHelperOutput output)
        {
            if (string.Equals(context.TagName, "body",
                              StringComparison.OrdinalIgnoreCase))
            {
                var script = await File.ReadAllTextAsync(
                    "TagHelpers/Templates/AddressToolTipScript.html");
                output.PostContent.AppendHtml(script);
            }
        }
    }
}
```

Um arquivo HTML separado é usado para armazenar o elemento `<script>`. O arquivo HTML torna o código mais limpo e mais sustentável. O código anterior lê o conteúdo de `TagHelpers/Templates/AddressToolTipScript.html` e acrescenta-o com a saída do Auxiliar de Marca. O arquivo `AddressToolTipScript.html` inclui a seguinte marcação:

```
<script>
$("address[printable]").hover(function() {
    $(this).attr({
        "data-toggle": "tooltip",
        "data-placement": "right",
        "title": "Home of Microsoft!"
    });
});
</script>
```

O código anterior associa um [widget de Dica de ferramenta de inicialização](#) a qualquer elemento `<address>` que inclua um atributo `printable`. O efeito é visível quando um ponteiro do mouse focaliza o elemento.

## Registrar um componente

Um Componente do Auxiliar de Marca precisa ser adicionado à coleção de Componentes do Auxiliar de Marca do

aplicativo. Há três maneiras de adicioná-lo à coleção:

1. [Registro por contêiner de serviços](#)
2. [Registro por arquivo Razor](#)
3. [Registro por Modelo de página ou controlador](#)

### Registro por contêiner de serviços

Se a classe do Componente do Auxiliar de Marca não for gerenciada com [ITagHelperComponentManager](#), ela precisará ser registrada com o sistema de [DI \(injeção de dependência\)](#). O código [Startup.ConfigureServices](#) a seguir registra as classes [AddressStyleTagHelperComponent](#) e [AddressScriptTagHelperComponent](#) com um [tempo de vida transitório](#):

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddMvc()
        .SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    services.AddTransient<ITagHelperComponent,
        AddressScriptTagHelperComponent>();
    services.AddTransient<ITagHelperComponent,
        AddressStyleTagHelperComponent>();
}
```

### Registro por arquivo Razor

Se o Componente do Auxiliar de Marca não estiver registrado com DI, ele poderá ser registrado por uma página do Razor Pages ou uma exibição do MVC. Essa técnica é usada para controlar a marcação injetada e o pedido de execução do componente de um arquivo Razor.

[ITagHelperComponentManager](#) é usado para adicionar Componentes do Auxiliar de Marca ou removê-los do aplicativo. O código a seguir demonstra essa técnica com [AddressTagHelperComponent](#):

```
@using RazorPagesSample.TagHelpers;
@using Microsoft.AspNetCore.Mvc.Razor.TagHelpers;
@inject ITagHelperComponentManager manager;

 @{
    string markup;

    if (Model.IsWeekend)
    {
        markup = "<em class='text-warning'>Office closed today!</em>";
    }
    else
    {
        markup = "<em class='text-info'>Office open today!</em>";
    }

    manager.Components.Add(new AddressTagHelperComponent(markup, 1));
}
```

No código anterior:

- A diretiva [@inject](#) fornece uma instância de [ITagHelperComponentManager](#). A instância é atribuída a uma variável chamada [manager](#) para acesso downstream no arquivo Razor.

- Uma instância do `AddressTagHelperComponent` é adicionada à coleção de Componentes do Auxiliar de Marca do aplicativo.

`AddressTagHelperComponent` é modificado para acomodar um construtor que aceita os parâmetros `markup` e `order`:

```
private readonly string _markup;

public override int Order { get; }

public AddressTagHelperComponent(string markup = "", int order = 1)
{
    _markup = markup;
    Order = order;
}
```

O parâmetro `markup` fornecido é usado em `ProcessAsync` da seguinte maneira:

```
public override async Task ProcessAsync(TagHelperContext context,
                                         TagHelperOutput output)
{
    if (string.Equals(context.TagName, "address",
                      StringComparison.OrdinalIgnoreCase) &&
        output.Attributes.ContainsName("printable"))
    {
        TagHelperContent childContent = await output.GetChildContentAsync();
        string content = childContent.GetContent();
        output.Content.SetHtmlContent(
            $"<div>{content}<br>{_markup}</div>{_printableButton}");
    }
}
```

## Registro por Modelo de página ou controlador

Se o Componente do Auxiliar de Marca não estiver registrado com DI, ele poderá ser registrado por um modelo de página do Razor Pages ou um controlador do MVC. Essa técnica é útil para separar a lógica C# de arquivos Razor.

A injeção do construtor é usada para acessar uma instância de `ITagHelperComponentManager`. Um Componente do Auxiliar de Marca é adicionado à coleção de Componentes do Auxiliar de Marca da instância. O modelo de página do Razor Pages a seguir demonstra essa técnica com `AddressTagHelperComponent`:

```

using System;
using Microsoft.AspNetCore.Mvc.Razor.TagHelpers;
using Microsoft.AspNetCore.Mvc.RazorPages;
using RazorPagesSample.TagHelpers;

public class IndexModel : PageModel
{
    private readonly ITagHelperComponentManager _tagHelperComponentManager;

    public bool IsWeekend
    {
        get
        {
            var dayOfWeek = DateTime.Now.DayOfWeek;

            return dayOfWeek == DayOfWeek.Saturday ||
                   dayOfWeek == DayOfWeek.Sunday;
        }
    }

    public IndexModel(ITagHelperComponentManager tagHelperComponentManager)
    {
        _tagHelperComponentManager = tagHelperComponentManager;
    }

    public void OnGet()
    {
        string markup;

        if (IsWeekend)
        {
            markup = "<em class='text-warning'>Office closed today!</em>";
        }
        else
        {
            markup = "<em class='text-info'>Office open today!</em>";
        }

        _tagHelperComponentManager.Components.Add(
            new AddressTagHelperComponent(markup, 1));
    }
}

```

No código anterior:

- A injeção do construtor é usada para acessar uma instância de `ITagHelperComponentManager`.
- Uma instância do `AddressTagHelperComponent` é adicionada à coleção de Componentes do Auxiliar de Marca do aplicativo.

## Criar um componente

Para criar um Componente do Auxiliar de Marca personalizado:

- Crie uma classe pública derivada de `TagHelperComponentTagHelper`.
- Aplique um atributo `[HtmlTargetElement]` à classe. Especifique o nome do elemento HTML de destino.
- *Opcional:* aplique um atributo `[EditorBrowsable(EditorBrowsableState.Never)]` à classe para suprimir a exibição do tipo no IntelliSense.

O código a seguir cria um Componente do Auxiliar de Marca personalizado que tem como destino o elemento HTML `<address>`:

```

using System.ComponentModel;
using Microsoft.AspNetCore.Mvc.Razor.TagHelpers;
using Microsoft.AspNetCore.Razor.TagHelpers;
using Microsoft.Extensions.Logging;

namespace RazorPagesSample.TagHelpers
{
    [HtmlTargetElement("address")]
    [EditorBrowsable(EditorBrowsableState.Never)]
    public class AddressTagHelperComponentTagHelper : TagHelperComponentTagHelper
    {
        public AddressTagHelperComponentTagHelper(
            ITagHelperComponentManager componentManager,
            ILoggerFactory loggerFactory) : base(componentManager, loggerFactory)
        {
        }
    }
}

```

Usar o Componente do Auxiliar de Marca `<address>` personalizado para inserir uma marcação HTML da seguinte maneira:

```

public class AddressTagHelperComponent : TagHelperComponent
{
    private readonly string _printableButton =
        "<button type='button' class='btn btn-info' onclick=\"window.open(\""
        "https://binged.it/2AXRRYw\")\">" +
        "<span class='glyphicon glyphicon-road' aria-hidden='true'></span>" +
        "</button>";

    public override int Order => 3;

    public override async Task ProcessAsync(TagHelperContext context,
                                            TagHelperOutput output)
    {
        if (string.Equals(context.TagName, "address",
                          StringComparison.OrdinalIgnoreCase) &&
            output.Attributes.ContainsName("printable"))
        {
            var content = await output.GetChildContentAsync();
            output.Content.SetHtmlContent(
                $"<div>{content.GetContent()}</div>{_printableButton}");
        }
    }
}

```

O método `ProcessAsync` anterior injeta o HTML fornecido para `SetHtmlContent` no elemento `<address>` correspondente. A injeção ocorre quando:

- O valor da propriedade `TagName` do contexto de execução é igual a `<address>`.
- O elemento `<address>` correspondente tem um atributo `printable`.

Por exemplo, a instrução `if` é avaliada como verdadeira ao processar o seguinte elemento `<address>`:

```

<address printable>
    One Microsoft Way<br />
    Redmond, WA 98052-6399<br />
    <abbr title="Phone">P:</abbr>
    425.555.0100
</address>

```

## Recursos adicionais

- [Injeção de dependência no ASP.NET Core](#)
- [Injeção de dependência em exibições no ASP.NET Core](#)
- [Auxiliares de marcação internos do ASP.NET Core](#)

# Auxiliar de Marca de Âncora no ASP.NET Core

10/01/2019 • 12 minutes to read • [Edit Online](#)

De [Peter Kellner](#) e [Scott Addie](#)

O [Auxiliar de Marca de Âncora](#) aprimora a marca de âncora HTML padrão (`<a ... ></a>`) adicionando novos atributos. Por convenção, os nomes de atributos são prefixados com `asp-`. O valor do atributo `href` do elemento de âncora renderizado é determinado pelos valores dos atributos `asp-`.

Para obter uma visão geral dos Auxiliares de Marca, confira [Auxiliares de Marca no ASP.NET Core](#).

[Exibir ou baixar código de exemplo \(como baixar\)](#)

*SpeakerController* é usado em exemplos ao longo de todo este documento:

```
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Linq;

public class SpeakerController : Controller
{
    private List<Speaker> Speakers =
        new List<Speaker>
    {
        new Speaker {SpeakerId = 10},
        new Speaker {SpeakerId = 11},
        new Speaker {SpeakerId = 12}
    };

    [Route("Speaker/{id:int}")]
    public IActionResult Detail(int id) =>
        View(Speakers.FirstOrDefault(a => a.SpeakerId == id));

    [Route("/Speaker/Evaluations",
        Name = "speakerevals")]
    public IActionResult Evaluations() => View();

    [Route("/Speaker/EvaluationsCurrent",
        Name = "speakerevalscurrent")]
    public IActionResult Evaluations(
        int speakerId,
        bool currentYear) => View();

    public IActionResult Index() => View(Speakers);
}

public class Speaker
{
    public int SpeakerId { get; set; }
}
```

Segue um inventário dos atributos `asp-`.

## asp-controller

O atributo `asp-controller` designa o controlador usado para gerar a URL. A marcação a seguir lista todos os palestrantes:

```
<a asp-controller="Speaker"  
    asp-action="Index">All Speakers</a>
```

O HTML gerado:

```
<a href="/Speaker">All Speakers</a>
```

Se o atributo `asp-controller` for especificado e `asp-action` não for, o valor `asp-action` padrão é a ação do controlador associada ao modo de exibição em execução no momento. Se `asp-action` for omitido da marcação anterior e o Auxiliar de Marca de Âncora for usado no modo de exibição `Índice (/home)` do `HomeController`, o HTML gerado será:

```
<a href="/Home">All Speakers</a>
```

## asp-action

O valor do atributo `asp-action` representa o nome da ação do controlador incluído no atributo `href` gerado. A seguinte marcação define o valor do atributo `href` gerado para a página de avaliações do palestrante:

```
<a asp-controller="Speaker"  
    asp-action="Evaluations">Speaker Evaluations</a>
```

O HTML gerado:

```
<a href="/Speaker/Evaluations">Speaker Evaluations</a>
```

Se nenhum atributo `asp-controller` for especificado, o controlador padrão que está chamando a exibição que executa a exibição atual é usado.

Se o valor do atributo `asp-action` for `Index`, nenhuma ação será anexada à URL, causando a invocação da ação `Index` padrão. A ação especificada (ou padrão), deve existir no controlador referenciado em `asp-controller`.

## asp-route-{value}

O atributo `asp-route-{value}` permite um prefixo de roteamento de caractere curinga. Qualquer valor que esteja ocupando o espaço reservado `{value}` é interpretado como um possível parâmetro de roteamento. Se uma rota padrão não for encontrada, esse prefixo de rota será anexado ao atributo `href` gerado como um valor e parâmetro de solicitação. Caso contrário, ele será substituído no modelo de rota.

Considere a seguinte ação do controlador:

```
public IActionResult AnchorTagHelper(int id)  
{  
    var speaker = new Speaker  
    {  
        SpeakerId = id  
    };  
  
    return View(speaker);  
}
```

Com um modelo de rota padrão definido em `Startup.Configure`:

```

app.UseMvc(routes =>
{
    // need route and attribute on controller: [Area("Blogs")]
    routes.MapRoute(name: "mvcAreaRoute",
                    template: "{area:exists}/{controller=Home}/{action=Index}");

    // default route for non-areas
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});

```

O modo de exibição MVC usa o modelo fornecido pela ação, da seguinte forma:

```

@model Speaker
<!DOCTYPE html>
<html>
<body>
    <a asp-controller="Speaker"
       asp-action="Detail"
       asp-route-id="@Model.SpeakerId">SpeakerId: @Model.SpeakerId</a>
</body>
</html>

```

O espaço reservado `{id?}` da rota padrão foi correspondido. O HTML gerado:

```
<a href="/Speaker/Detail/12">SpeakerId: 12</a>
```

Suponha que o prefixo de roteamento não faça parte do modelo de roteamento de correspondência, como com o seguinte modo de exibição de MVC:

```

@model Speaker
<!DOCTYPE html>
<html>
<body>
    <a asp-controller="Speaker"
       asp-action="Detail"
       asp-route-speakerid="@Model.SpeakerId">SpeakerId: @Model.SpeakerId</a>
</body>
</html>

```

O seguinte HTML é gerado porque o `speakerid` não foi encontrado na rota correspondente:

```
<a href="/Speaker/Detail?speakerid=12">SpeakerId: 12</a>
```

Se `asp-controller` ou `asp-action` não forem especificados, o mesmo processamento padrão será seguido, como no atributo `asp-route`.

## asp-route

O atributo `asp-route` é usado para criar um link de URL diretamente para uma rota nomeada. Usando [atributos de roteamento](#), uma rota pode ser nomeada como mostrado em `SpeakerController` e usada em sua ação `Evaluations`:

```
[Route("/Speaker/Evaluations",
      Name = "speakerevals")]
public IActionResult Evaluations() => View();
```

Na seguinte marcação, o atributo `asp-route` faz referência à rota nomeada:

```
<a asp-route="speakerevals">Speaker Evaluations</a>
```

O Auxiliar de Marca de Âncora gera uma rota diretamente para essa ação de controlador usando a URL `/Speaker/Evaluations`. O HTML gerado:

```
<a href="/Speaker/Evaluations">Speaker Evaluations</a>
```

Se `asp-controller` ou `asp-action` for especificado além de `asp-route`, a rota gerada poderá não ser a esperada. Para evitar um conflito de rota, `asp-route` não deve ser usado com os atributos `asp-controller` ou `asp-action`.

## asp-all-route-data

O atributo `asp-all-route-data` oferece suporte à criação de um dicionário ou par chave-valor. A chave é o nome do parâmetro e o valor é o valor do parâmetro.

No exemplo a seguir, um dicionário é inicializado e passado para um modo de exibição Razor. Como alternativa, os dados podem ser passado com seu modelo.

```
@{
    var parms = new Dictionary<string, string>
    {
        { "speakerId", "11" },
        { "currentYear", "true" }
    };
}

<a asp-route="speakerevalscurrent"
    asp-all-route-data="parms">Speaker Evaluations</a>
```

O código anterior gera o seguinte HTML:

```
<a href="/Speaker/EvaluationsCurrent?speakerId=11&currentYear=true">Speaker Evaluations</a>
```

O dicionário `asp-all-route-data` é simplificado para produzir um querystring que atenda aos requisitos da ação `Evaluations` sobrecarregada:

```
[Route("/Speaker/EvaluationsCurrent",
      Name = "speakerevalscurrent")]
public IActionResult Evaluations(
    int speakerId,
    bool currentYear) => View();
```

Se todas as chaves no dicionário corresponderem aos parâmetros, esses valores serão substituídos na rota conforme apropriado. Os outros valores não correspondentes são gerados como parâmetros de solicitação.

## asp-fragment

O atributo `asp-fragment` define um fragmento de URL para anexar à URL. O Auxiliar de Marca de Âncora adiciona o caractere de hash (#). Considere a seguinte marcação:

```
<a asp-controller="Speaker"  
    asp-action="Evaluations"  
    asp-fragment="SpeakerEvaluations">Speaker Evaluations</a>
```

O HTML gerado:

```
<a href="/Speaker/Evaluations#SpeakerEvaluations">Speaker Evaluations</a>
```

As marcas de hash são úteis ao criar aplicativos do lado do cliente. Elas podem ser usadas para marcar e pesquisar com facilidade em JavaScript, por exemplo.

## asp-area

O atributo `asp-area` define o nome de área usado para definir a rota apropriada. O exemplo a seguir ilustra como o atributo `asp-area` causa um remapeamento das rotas.

### Uso no Razor Pages

As áreas do Razor Pages têm suporte no ASP.NET Core 2.1 ou posterior.

Considere a seguinte hierarquia de diretórios:

- {Nome do projeto}
  - **wwwroot**
  - **Áreas**
    - **Sessões**
      - **Páginas**
        - *\_ViewStart.cshtml*
        - *Index.cshtml*
        - *Index.cshtml.cs*
    - **Páginas**

A marcação para fazer referência ao Razor Page de *Índice* da área *Sessões* é:

```
<a asp-area="Sessions"  
    asp-page="/Index">View Sessions</a>
```

O HTML gerado:

```
<a href="/Sessions">View Sessions</a>
```

## TIP

para dar suporte a áreas em um aplicativo do Razor Pages, siga um destes procedimentos em

`Startup.ConfigureServices` :

- Defina a [versão de compatibilidade](#) para 2.1 ou posterior.
- Defina a propriedade `RazorPagesOptions.AllowAreas` como `true` :

```
services.AddMvc()
    .AddRazorPagesOptions(options => options.AllowAreas = true);
```

## Uso no MVC

Considere a seguinte hierarquia de diretórios:

- {Nome do projeto}
  - **wwwroot**
  - **Áreas**
    - **Blogs**
      - **Controladores**
        - *HomeController.cs*
      - **Exibições**
        - **Início**
          - *AboutBlog.cshtml*
          - *Index.cshtml*
          - *\_ViewStart.cshtml*
      - **Controladores**

Definir `asp-area` como "Blogs" faz com que o diretório *Áreas/Blogs* seja prefixado nas rotas dos controladores e exibições associados a essa marca de âncora. A marcação para fazer referência à exibição *AboutBlog* é:

```
<a asp-area="Blogs"
    asp-controller="Home"
    asp-action="AboutBlog">About Blog</a>
```

O HTML gerado:

```
<a href="/Blogs/Home/AboutBlog">About Blog</a>
```

## TIP

Para dar suporte às áreas em um aplicativo MVC, o modelo de rota deverá incluir uma referência à área, se ela existir. Esse modelo é representado pelo segundo parâmetro da chamada do método `routes.MapRoute` em `Startup.Configure`:

```
app.UseMvc(routes =>
{
    // need route and attribute on controller: [Area("Blogs")]
    routes.MapRoute(name: "mvcAreaRoute",
                    template: "{area:exists}/{controller=Home}/{action=Index}");

    // default route for non-areas
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});
```

## asp-protocol

O atributo `asp-protocol` é destinado a especificar um protocolo (como `https`) em sua URL. Por exemplo:

```
<a asp-protocol="https"
    asp-controller="Home"
    asp-action="About">About</a>
```

O HTML gerado:

```
<a href="https://localhost/Home/About">About</a>
```

O nome do host no exemplo é localhost. O Auxiliar de Marca de Âncora usa o domínio público do site ao gerar a URL.

## asp-host

O atributo `asp-host` é para especificar um nome do host na sua URL. Por exemplo:

```
<a asp-protocol="https"
    asp-host="microsoft.com"
    asp-controller="Home"
    asp-action="About">About</a>
```

O HTML gerado:

```
<a href="https://microsoft.com/Home/About">About</a>
```

## asp-page

O atributo `asp-page` é usado com as Páginas Razor. Use-o para definir um valor de atributo `href` da marca de âncora para uma página específica. Prefixar o nome de página com uma barra ("/") cria a URL.

O exemplo a seguir aponta para o participante da Página Razor:

```
<a asp-page="/Attendee">All Attendees</a>
```

O HTML gerado:

```
<a href="/Attendee">All Attendees</a>
```

O atributo `asp-page` é mutuamente exclusivo com os atributos `asp-route`, `asp-controller` e `asp-action`. No entanto, o `asp-page` pode ser usado com o `asp-route-{value}` para controlar o roteamento, como mostra a marcação a seguir:

```
<a asp-page="/Attendee"  
    asp-route-attendeeid="10">View Attendee</a>
```

O HTML gerado:

```
<a href="/Attendee?attendeeid=10">View Attendee</a>
```

## asp-page-handler

O atributo `asp-page-handler` é usado com as Páginas Razor. Ele se destina à vinculação a manipuladores de página específicos.

Considere o seguinte manipulador de página:

```
public void OnGetProfile(int attendeeId)  
{  
    ViewData["AttendeeId"] = attendeeId;  
  
    // code omitted for brevity  
}
```

A marcação associada do modelo de página vincula ao manipulador de página `OnGetProfile`. Observe que o prefixo `On<Verb>` do nome do método do manipulador de página é omitido no valor do atributo `asp-page-handler`. Quando o método é assíncrono, o sufixo `Async` também é omitido.

```
<a asp-page="/Attendee"  
    asp-page-handler="Profile"  
    asp-route-attendeeid="12">Attendee Profile</a>
```

O HTML gerado:

```
<a href="/Attendee?attendeeid=12&handler=Profile">Attendee Profile</a>
```

## Recursos adicionais

- [Áreas no ASP.NET Core](#)
- [Introdução a Páginas do Razor no ASP.NET Core](#)
- [Versão de compatibilidade do ASP.NET Core MVC](#)

# Auxiliar de Marca de Cache no ASP.NET Core MVC

10/11/2018 • 8 minutes to read • [Edit Online](#)

Por [Peter Kellner](#) e [Luke Latham](#)

O Auxiliar de Marca de Cache possibilita melhorar o desempenho de seu aplicativo ASP.NET Core armazenando seu conteúdo em cache no provedor de cache interno do ASP.NET Core.

Para obter uma visão geral dos Auxiliares de Marca, confira [Auxiliares de Marca no ASP.NET Core](#).

A seguinte marcação Razor armazena em cache a data atual:

```
<cache>@DateTime.Now</cache>
```

A primeira solicitação para a página que contém o Auxiliar de Marca exibe a data atual. Solicitações adicionais mostram o valor armazenado em cache até que o cache expire (padrão de 20 minutos) ou até que a data em cache seja removida do cache.

## Atributos do Auxiliar de Marca de Cache

### habilitado

TIPO DE ATRIBUTO	EXEMPLOS	PADRÃO
Boolean	<code>true</code> , <code>false</code>	<code>true</code>

`enabled` determina se o conteúdo envolto pelo Auxiliar de Marca de Cache é armazenado em cache. O padrão é `true`. Se definido como `false`, a saída renderizada **não** é armazenada em cache.

Exemplo:

```
<cache enabled="true">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

### expires-on

TIPO DE ATRIBUTO	EXEMPLO
<code>DateTimeOffset</code>	<code>@new DateTime(2025,1,29,17,02,0)</code>

`expires-on` define uma data do término absoluta para o item em cache.

O exemplo a seguir armazena em cache o conteúdo do Auxiliar de Marca de Cache até às 17:02 de 29 de janeiro de 2025:

```
<cache expires-on="@new DateTime(2025,1,29,17,02,0)">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

### expires-after

TIPO DE ATRIBUTO	EXEMPLO	PADRÃO
TimeSpan	@TimeSpan.FromSeconds(120)	20 minutos

`expires-after` define o tempo decorrido desde a primeira solicitação para armazenar o conteúdo em cache.

Exemplo:

```
<cache expires-after="@TimeSpan.FromSeconds(120)">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

O Mecanismo de Exibição do Razor define o valor `expires-after` padrão como vinte minutos.

### expires-sliding

TIPO DE ATRIBUTO	EXEMPLO
TimeSpan	@TimeSpan.FromSeconds(60)

Define a hora em que uma entrada de cache deverá ser removida se o seu valor não tiver sido acessado.

Exemplo:

```
<cache expires-sliding="@TimeSpan.FromSeconds(60)">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

### vary-by-header

TIPO DE ATRIBUTO	EXEMPLOS
Cadeia de Caracteres	User-Agent , User-Agent,content-encoding

`vary-by-header` aceita uma lista delimitada por vírgulas de valores de cabeçalho que disparam uma atualização do cache quando eles mudam.

O exemplo a seguir monitora o valor do cabeçalho `User-Agent`. O exemplo armazena em cache o conteúdo para cada `User-Agent` diferente apresentado ao servidor Web:

```
<cache vary-by-header="User-Agent">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

### vary-by-query

TIPO DE ATRIBUTO	EXEMPLOS
Cadeia de Caracteres	Make , Make,Model

`vary-by-query` aceita uma lista delimitada por vírgula de [Keys](#) em uma cadeia de consulta ([Query](#)) que dispara uma atualização do cache quando o valor de qualquer chave listada é alterado.

O exemplo a seguir monitora os valores de `Make` e `Model`. O exemplo armazena em cache o conteúdo para todos os diferentes `Make` e `Model` apresentados ao servidor Web:

```
<cache vary-by-query="Make,Model">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

## vary-by-route

TIPO DE ATRIBUTO	EXEMPLOS
Cadeia de Caracteres	Make , Make,Model

`vary-by-route` aceita uma lista delimitada por vírgulas de nomes de parâmetros de rota que disparam uma atualização do cache quando o valor de parâmetro de dados de rota muda.

Exemplo:

*Startup.cs*:

```
routes.MapRoute(
    name: "default",
    template: "{controller=Home}/{action=Index}/{Make?}/{Model?}");
```

*Index.cshtml*:

```
<cache vary-by-route="Make,Model">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

## vary-by-cookie

TIPO DE ATRIBUTO	EXEMPLOS
Cadeia de Caracteres	.AspNetCore.Identity.Application , .AspNetCore.Identity.Application,HairColor

`vary-by-cookie` aceita uma lista delimitada por vírgulas de nomes de cookie que disparam uma atualização do cache quando os valores de cookie mudam.

O exemplo a seguir monitora o cookie associado à identidade do ASP.NET Core. Quando um usuário é autenticado, uma alteração ao cookie de Identidade dispara uma atualização do cache:

```
<cache vary-by-cookie=".AspNetCore.Identity.Application">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

## vary-by-user

TIPO DE ATRIBUTO	EXEMPLOS	PADRÃO
Boolean	true , false	true

`vary-by-user` especifica se o cache é redefinido ou não quando o usuário conectado (ou a Entidade de Contexto) muda. O usuário atual também é conhecido como a Entidade do contexto de solicitação e pode ser exibido em um modo de exibição do Razor referenciando `@User.Identity.Name`.

O exemplo a seguir monitora o usuário conectado atual para disparar uma atualização de cache:

```
<cache vary-by-user="true">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

Usar esse atributo mantém o conteúdo no cache durante um ciclo de entrada e saída. Quando o valor é definido como `true`, um ciclo de autenticação invalida o cache para o usuário autenticado. O cache é invalidado porque um novo valor de cookie exclusivo é gerado quando um usuário é autenticado. O cache é mantido para o estado anônimo quando nenhum cookie está presente ou quando o cookie expirou. Se o usuário **não** estiver autenticado, o cache será mantido.

### vary-by

TIPO DE ATRIBUTO	EXEMPLO
Cadeia de Caracteres	<code>@Model</code>

`vary-by` permite a personalização de quais dados são armazenados em cache. Quando o objeto referenciado pelo valor de cadeia de caracteres do atributo é alterado, o conteúdo do Auxiliar de Marca de Cache é atualizado. Frequentemente, uma concatenação de cadeia de caracteres de valores do modelo é atribuída a este atributo. Na verdade, isso resulta em um cenário em que uma atualização de qualquer um dos valores concatenados invalida o cache.

O exemplo a seguir supõe que o método do controlador que renderiza a exibição somas o valor inteiro dos dois parâmetros de rota, `myParam1` e `myParam2`, e os retorna a soma como a propriedade de modelo única. Quando essa soma é alterada, o conteúdo do Auxiliar de Marca de Cache é renderizado e armazenado em cache novamente.

Ação:

```
public IActionResult Index(string myParam1, string myParam2, string myParam3)
{
    int num1;
    int num2;
    int.TryParse(myParam1, out num1);
    int.TryParse(myParam2, out num2);
    return View(viewName, num1 + num2);
}
```

*Index.cshtml:*

```
<cache vary-by="@Model">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

### priority

TIPO DE ATRIBUTO	EXEMPLOS	PADRÃO
<code>CacheItemPriority</code>	<code>High</code> , <code>Low</code> , <code>NeverRemove</code> , <code>Normal</code>	<code>Normal</code>

`priority` fornece diretrizes de remoção do cache para o provedor de cache interno. O servidor Web remove entradas de cache `Low` primeiro quando está sob demanda de memória.

Exemplo:

```
<cache priority="High">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

O atributo `priority` não assegura um nível específico de retenção de cache. `CacheItemPriority` é apenas uma sugestão. Configurar esse atributo como `NeverRemove` não assegura que os itens armazenados em cache sempre sejam retidos. Veja os tópicos na seção [Recursos Adicionais](#) para obter mais informações.

O Auxiliar de Marca de Cache é dependente do [serviço de cache de memória](#). O Auxiliar de Marca de Cache adicionará o serviço se ele não tiver sido adicionado.

## Recursos adicionais

- [Memória de cache no ASP.NET Core](#)
- [Introdução à identidade do ASP.NET Core](#)

# Auxiliar de Marca de Cache Distribuído no ASP.NET Core

29/10/2018 • 3 minutes to read • [Edit Online](#)

Por [Peter Kellner](#) e [Luke Latham](#)

O Auxiliar de Marca de Cache Distribuído fornece a capacidade de melhorar consideravelmente o desempenho do aplicativo ASP.NET Core armazenando seu conteúdo em cache em uma fonte de cache distribuído.

Para obter uma visão geral dos Auxiliares de Marca, confira [Auxiliares de Marca no ASP.NET Core](#).

O Auxiliar de Marca de Cache Distribuído herda da mesma classe base do Auxiliar de Marca de Cache. Todos os atributos de [Auxiliar de Marca de Cache](#) estão disponíveis ao Auxiliar de Marca Distribuído.

O Auxiliar de Marca de Cache distribuído usa [injeção de construtor](#). A interface [IDistributedCache](#) é passada para o construtor do Auxiliar de Marca de Cache Distribuído. Se nenhuma implementação concreta de [IDistributedCache](#) for criada em [Startup.ConfigureServices](#) ([Startup.cs](#)), o Auxiliar de Marca de Cache Distribuído usará o mesmo provedor na memória para armazenar dados em cache como o [Auxiliar de Marca de Cache](#).

## Atributos do auxiliar de marca de cache distribuído

### Atributos compartilhados com o Auxiliar de Marca de Cache

- `enabled`
- `expires-on`
- `expires-after`
- `expires-sliding`
- `vary-by-header`
- `vary-by-query`
- `vary-by-route`
- `vary-by-cookie`
- `vary-by-user`
- `vary-by priority`

O Auxiliar de Marca de Cache Distribuído herda da mesma classe que o Auxiliar de Marca de Cache. Para obter descrições desses atributos, confira [Auxiliar de Marca de Cache](#).

#### **name**

TIPO DE ATRIBUTO	EXEMPLO
Cadeia de Caracteres	<code>my-distributed-cache-unique-key-101</code>

`name` é obrigatório. O atributo `name` é usado como uma chave para cada instância de cache armazenada. Ao contrário do Auxiliar de Marca de Cache que atribui uma chave de cache a cada instância com base no nome da página do Razor e na localização na página do Razor, o Auxiliar de Marca de Cache Distribuído somente localiza suas chaves no atributo `name`.

Exemplo:

```
<distributed-cache name="my-distributed-cache-unique-key-101">
    Time Inside Cache Tag Helper: @DateTime.Now
</distributed-cache>
```

## Implementações de `IDistributedCache` do Auxiliar de Marca de Cache Distribuído

Há duas implementações de `IDistributedCache` internas do ASP.NET Core. Uma é baseada no SQL Server e a outra, no Redis. Os detalhes dessas implementações podem ser encontrados em [O cache no ASP.NET Core distribuído](#). Ambas as implementações envolvem configurar de uma instância do `IDistributedCache` em `Startup`.

Não há nenhum atributo de marca especificamente associado ao uso de uma implementação específica de `IDistributedCache`.

## Recursos adicionais

- [Auxiliar de Marca de Cache no ASP.NET Core MVC](#)
- [Injeção de dependência no ASP.NET Core](#)
- [O cache no ASP.NET Core distribuído](#)
- [Memória de cache no ASP.NET Core](#)
- [Introdução à identidade do ASP.NET Core](#)

# Auxiliar de Marca de Ambiente no ASP.NET Core

29/10/2018 • 2 minutes to read • [Edit Online](#)

Por [Peter Kellner](#), [Hisham Bin Ateya](#) e [Luke Latham](#)

O Auxiliar de Marca de Ambiente renderiza condicionalmente seu conteúdo contido com base no [ambiente de hospedagem](#) atual. Atributo único do Auxiliar de Marca de Ambiente, `names`, é uma lista separada por vírgulas de nomes de ambiente. Se nenhum dos nomes de ambiente fornecido corresponder ao ambiente atual, o conteúdo contido será renderizado.

Para obter uma visão geral dos Auxiliares de Marca, confira [Auxiliares de Marca no ASP.NET Core](#).

## Atributos do Auxiliar de Marca de Ambiente

### **nomes**

`names` aceita um único nome de ambiente de hospedagem ou uma lista separada por vírgula de nomes de ambiente de hospedagem que disparam a renderização do conteúdo contido.

Valores de ambiente são comparados com o valor retornado por [IHostingEnvironment.EnvironmentName](#). A comparação ignora o uso de maiúsculas.

O exemplo a seguir usa um Auxiliar de Marca de Ambiente. O conteúdo será renderizado se o ambiente de hospedagem for De Preparo ou de Produção:

```
<environment names="Staging,Production">
  <strong>HostingEnvironment.EnvironmentName is Staging or Production</strong>
</environment>
```

## incluir e excluir atributos

Os atributos `include` & `exclude` controlam a renderização do conteúdo contido com base nos nomes de ambiente de hospedagem incluídos ou excluídos.

### **include**

A propriedade `include` exibe um comportamento semelhante para o atributo `names`. Um ambiente listado no valor do atributo `include` deve corresponder ao ambiente de hospedagem do aplicativo ([IHostingEnvironment.EnvironmentName](#)) para renderizar o conteúdo da marcação `<environment>`.

```
<environment include="Staging,Production">
  <strong>HostingEnvironment.EnvironmentName is Staging or Production</strong>
</environment>
```

### **exclude**

Em contraste com o atributo `include`, o conteúdo da marcação `<environment>` é processado quando o ambiente de hospedagem não corresponde a um ambiente listado no valor do atributo `exclude`.

```
<environment exclude="Development">
  <strong>HostingEnvironment.EnvironmentName is not Development</strong>
</environment>
```

## Recursos adicionais

- [Usar vários ambientes no ASP.NET Core](#)

# Auxiliares de marca em formulários no ASP.NET Core

14/01/2019 • 28 minutes to read • [Edit Online](#)

Por [Rick Anderson](#), [Dave Paquette](#) e [Jerrie Pelser](#)

Este documento demonstra como é o trabalho com Formulários e os elementos HTML usados comumente em um Formulário. O elemento HTML [Formulário](#) fornece o mecanismo primário que os aplicativos Web usam para postar dados para o servidor. A maior parte deste documento descreve os [Auxiliares de marca](#) e como eles podem ajudar você a criar formulários HTML robustos de forma produtiva. É recomendável que você leia [Introdução ao auxiliares de marca](#) antes de ler este documento.

Em muitos casos, os Auxiliares HTML fornecem uma abordagem alternativa a um Auxiliar de Marca específico, mas é importante reconhecer que os Auxiliares de Marca não substituem os Auxiliares HTML e que não há um Auxiliar de Marca para cada Auxiliar HTML. Quando existe um Auxiliar HTML alternativo, ele é mencionado.

## O Auxiliar de marca de formulário

O Auxiliar de marca de [formulário](#):

- Gera o valor do atributo HTML `<FORM>` `action` para uma ação do controlador MVC ou uma rota nomeada
- Gera um [Token de verificação de solicitação](#) oculto para evitar a falsificação de solicitações entre sites (quando usado com o atributo `[ValidateAntiForgeryToken]` no método de ação HTTP Post)
- Fornece o atributo `asp-route-<Parameter Name>`, em que `<Parameter Name>` é adicionado aos valores de rota. Os parâmetros `routeValues` para `Html.BeginForm` e `Html.BeginRouteForm` fornecem funcionalidade semelhante.
- Tem uma alternativa de Auxiliar HTML `Html.BeginForm` e `Html.BeginRouteForm`

Amostra:

```
<form asp-controller="Demo" asp-action="Register" method="post">
    <!-- Input and Submit elements -->
</form>
```

O Auxiliar de marca de formulário acima gera o HTML a seguir:

```
<form method="post" action="/Demo/Register">
    <!-- Input and Submit elements -->
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

O tempo de execução do MVC gera o valor do atributo `action` dos atributos `asp-controller` e `asp-action` do Auxiliar de marca de formulário. O Auxiliar de marca de formulário também gera um [Token de verificação de solicitação](#) oculto para evitar a falsificação de solicitações entre sites (quando usado com o atributo `[ValidateAntiForgeryToken]` no método de ação HTTP Post). É difícil proteger um Formulário HTML puro contra falsificação de solicitações entre sites e o Auxiliar de marca de formulário fornece este serviço para você.

### Usando uma rota nomeada

O atributo do Auxiliar de Marca `asp-route` também pode gerar a marcação para o atributo HTML `action`. Um

aplicativo com uma [rota](#) chamada `register` poderia usar a seguinte marcação para a página de registro:

```
<form asp-route="register" method="post">
    <!-- Input and Submit elements -->
</form>
```

Muitas das exibições na pasta *Modos de Exibição/Conta* (gerada quando você cria um novo aplicativo Web com *Contas de usuário individuais*) contêm o atributo [asp-route-returnurl](#):

```
<form asp-controller="Account" asp-action="Login"
      asp-route-returnurl="@ViewData["ReturnUrl"]"
      method="post" class="form-horizontal" role="form">
```

#### NOTE

Com os modelos internos, `returnUrl` só é preenchido automaticamente quando você tenta acessar um recurso autorizado, mas não está autenticado ou autorizado. Quando você tenta fazer um acesso não autorizado, o middleware de segurança o redireciona para a página de logon com o `returnUrl` definido.

## O auxiliar de marca de entrada

O Auxiliar de marca de entrada associa um elemento HTML `<input>` a uma expressão de modelo em sua exibição do Razor.

Sintaxe:

```
<input asp-for="<Expression Name>" />
```

O auxiliar de marca de entrada:

- Gera os atributos HTML `id` e `name` para o nome da expressão especificada no atributo `asp-for`.  
`asp-for="Property1.Property2"` equivale a `m => m.Property1.Property2`. O nome da expressão é o que é usado para o valor do atributo `asp-for`. Consulte a seção [Nomes de expressão](#) para obter informações adicionais.
- Define o valor do atributo HTML `type` com base nos atributos de tipo de modelo e [anotação de dados](#) aplicados à propriedade de modelo
- O valor do atributo HTML `type` não será substituído quando um for especificado
- Gera atributos de validação [HTML5](#) de atributos de [anotação de dados](#) aplicados a propriedades de modelo
- Tem uma sobreposição de recursos de Auxiliar HTML com `Html.TextBoxFor` e `Html.EditorFor`. Consulte a seção **Alternativas de Auxiliar HTML ao Auxiliar de marca de entrada** para obter detalhes.
- Fornece tipagem forte. Se o nome da propriedade for alterado e você não atualizar o Auxiliar de marca, você verá um erro semelhante ao seguinte:

An error occurred during the compilation of a resource required to process this request. Please review the following specific error details and modify your source code appropriately.

Type expected

'RegisterViewModel' does not contain a definition for 'Email' and no extension method 'Email' accepting a first argument of type 'RegisterViewModel' could be found (are you missing a using directive or an assembly reference?)

O Auxiliar de marca `Input` define o atributo HTML `type` com base no tipo .NET. A tabela a seguir lista alguns tipos .NET comuns e o tipo HTML gerado (não estão listados todos os tipos .NET).

TIPO .NET	TIPO DE ENTRADA
Bool	<code>type="checkbox"</code>
Cadeia de Caracteres	<code>type="text"</code>
DateTime	<code>type="datetime-local"</code>
Byte	<code>type="number"</code>
int	<code>type="number"</code>
Single e Double	<code>type="number"</code>

A tabela a seguir mostra alguns atributos de [anotações de dados](#) comuns que o auxiliar de marca de entrada mapeará para tipos de entrada específicos (não são listados todos os atributos de validação):

ATRIBUTO	TIPO DE ENTRADA
<code>[EmailAddress]</code>	<code>type="email"</code>
<code>[Url]</code>	<code>type="url"</code>
<code>[HiddenInput]</code>	<code>type="hidden"</code>
<code>[Phone]</code>	<code>type="tel"</code>
<code>[DataType(DataType.Password)]</code>	<code>type="password"</code>
<code>[DataType(DataType.Date)]</code>	<code>type="date"</code>
<code>[DataType(DataType.Time)]</code>	<code>type="time"</code>

Amostra:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}

```

```

@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterInput" method="post">
    Email: <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    <button type="submit">Register</button>
</form>

```

O código acima gera o seguinte HTML:

```

<form method="post" action="/Demo/RegisterInput">
    Email:
    <input type="email" data-val="true"
           data-val-email="The Email Address field is not a valid email address."
           data-val-required="The Email Address field is required."
           id="Email" name="Email" value="" /> <br>
    Password:
    <input type="password" data-val="true"
           data-val-required="The Password field is required."
           id="Password" name="Password" /><br>
    <button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

As anotações de dados aplicadas às propriedades `Email` e `Password` geram metadados no modelo. O Auxiliar de marca de entrada consome os metadados do modelo e produz atributos HTML5 `data-val-*` (consulte [Validação de modelo](#)). Esses atributos descrevem os validadores a serem anexados aos campos de entrada. Isso fornece validação de [jQuery](#) e HTML5 discreto. Os atributos discretos têm o formato `data-val-rule="Error Message"`, em que a regra é o nome da regra de validação (como `data-val-required`, `data-val-email`, `data-val-maxlength` etc.). Se uma mensagem de erro for fornecida no atributo, ela será exibida como o valor para o atributo `data-val-rule`. Também há atributos do formulário `data-val-ruleName-argumentName="argumentValue"` que fornecem detalhes adicionais sobre a regra, por exemplo, `data-val-maxlength-max="1024"`.

## Alternativas de Auxiliar HTML ao Auxiliar de marca de entrada

`Html.TextBox`, `Html.TextBoxFor`, `Html.Editor` e `Html.EditorFor` têm recursos que se sobrepõem aos do Auxiliar de marca de entrada. O Auxiliar de marca de entrada define automaticamente o atributo `type`; `Html.TextBox` e `Html.TextBoxFor` não o fazem. `Html.Editor` e `Html.EditorFor` manipulam coleções, objetos complexos e modelos; o Auxiliar de marca de entrada não o faz. O Auxiliar de marca de entrada, `Html.EditorFor` e `Html.TextBoxFor` são fortemente tipados (eles usam expressões lambda); `Html.TextBox` e `Html.Editor` não usam (eles usam nomes de expressão).

## HtmlAttributes

`@Html.Editor()` e `@Html.EditorFor()` usam uma entrada `ViewDataDictionary` especial chamada `htmlAttributes` ao executar seus modelos padrão. Esse comportamento pode ser aumentado usando parâmetros `additional ViewData`. A chave "htmlAttributes" diferencia maiúsculas de minúsculas. A chave "htmlAttributes" é tratada de forma semelhante ao objeto `htmlAttributes` passado para auxiliares de entrada como `@Html.TextBox()`.

```
@Html.EditorFor(model => model.YourProperty,  
new { htmlAttributes = new { @class="myCssClass", style="Width:100px" } })
```

## Nomes de expressão

O valor do atributo `asp-for` é um `ModelExpression` e o lado direito de uma expressão lambda. Portanto, `asp-for="Property1"` se torna `m => m.Property1` no código gerado e é por isso você não precisa colocar o prefixo `Model`. Você pode usar o caractere "@" para iniciar uma expressão embutida e mover para antes de `m`:

```
@{  
    var joe = "Joe";  
}  
<input asp-for="@joe" />
```

Gera o seguinte:

```
<input type="text" id="joe" name="joe" value="Joe" />
```

Com propriedades de coleção, `asp-for="CollectionProperty[23].Member"` gera o mesmo nome que `asp-for="CollectionProperty[i].Member"` quando `i` tem o valor `23`.

Quando o ASP.NET Core MVC calcula o valor de `ModelExpression`, ele inspeciona várias fontes, inclusive o `ModelState`. Considere o `<input type="text" asp-for="@Name" />`. O atributo `value` calculado é o primeiro valor não nulo:

- Da entrada de `ModelState` com a chave "Name".
- Do resultado da expressão `Model.Name`.

## Navegando para propriedades filho

Você também pode navegar para propriedades filho usando o caminho da propriedade do modelo de exibição. Considere uma classe de modelo mais complexa que contém uma propriedade `Address` filho.

```
public class AddressViewModel  
{  
    public string AddressLine1 { get; set; }  
}
```

```
public class RegisterAddressViewModel  
{  
    public string Email { get; set; }  
  
    [DataType(DataType.Password)]  
    public string Password { get; set; }  
  
    public AddressViewModel Address { get; set; }  
}
```

Na exibição, associamos a `Address.AddressLine1`:

```
@model RegisterAddressViewModel

<form asp-controller="Demo" asp-action="RegisterAddress" method="post">
    Email: <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    Address: <input asp-for="Address.AddressLine1" /><br />
    <button type="submit">Register</button>
</form>
```

O HTML a seguir é gerado para `Address.AddressLine1`:

```
<input type="text" id="Address_AddressLine1" name="Address.AddressLine1" value="" />
```

## Nomes de expressão e coleções

Exemplo, um modelo que contém uma matriz de `Colors`:

```
public class Person
{
    public List<string> Colors { get; set; }

    public int Age { get; set; }
}
```

O método de ação:

```
public IActionResult Edit(int id, int colorIndex)
{
    ViewData["Index"] = colorIndex;
    return View(GetPerson(id));
}
```

O Razor a seguir mostra como você acessa um elemento `Color` específico:

```
@model Person
 @{
     var index = (int)ViewData["index"];
 }

<form asp-controller="ToDo" asp-action="Edit" method="post">
    @Html.EditorFor(m => m.Colors[index])
    <label asp-for="Age"></label>
    <input asp-for="Age" /><br />
    <button type="submit">Post</button>
</form>
```

O modelo `Views/Shared/EditorTemplates/String.cshtml`:

```
@model string

<label asp-for="@Model"></label>
<input asp-for="@Model" /> <br />
```

Exemplo usando `List<T>`:

```

public class ToDoItem
{
    public string Name { get; set; }

    public bool IsDone { get; set; }
}

```

O Razor a seguir mostra como iterar em uma coleção:

```

@model List<ToDoItem>

<form asp-controller="ToDo" asp-action="Edit" method="post">
    <table>
        <tr> <th>Name</th> <th>Is Done</th> </tr>

        @for (int i = 0; i < Model.Count; i++)
        {
            <tr>
                @Html.EditorFor(model => model[i])
            </tr>
        }

    </table>
    <button type="submit">Save</button>
</form>

```

O modelo *Views/Shared/EditorTemplates/ToDoItem.cshtml*:

```

@model ToDoItem

<td>
    <label asp-for="@Model.Name"></label>
    @Html.DisplayFor(model => model.Name)
</td>
<td>
    <input asp-for="@Model.IsDone" />
</td>

/*
This template replaces the following Razor which evaluates the indexer three times.
<td>
    <label asp-for="@Model[i].Name"></label>
    @Html.DisplayFor(model => model[i].Name)
</td>
<td>
    <input asp-for="@Model[i].IsDone" />
</td>
*/

```

`foreach` deve ser usado, se possível, quando o valor está prestes a ser usado em um contexto equivalente `asp-for` ou `Html.DisplayFor`. Em geral, `for` é melhor do que `foreach` (se o cenário permitir) porque não é necessário alocar um enumerador; no entanto, avaliar um indexador em uma expressão LINQ pode ser caro, o que deve ser minimizado.

#### NOTE

O código de exemplo comentado acima mostra como você substituiria a expressão lambda pelo operador `@` para acessar cada `ToDoItem` na lista.

## Auxiliar de marca de área de texto

O auxiliar de marca `Textarea Tag Helper` é semelhante ao Auxiliar de marca de entrada.

- Gera os atributos `id` e `name`, bem como os atributos de validação de dados do modelo para um elemento `<textarea>`.
- Fornece tipagem forte.
- Alternativa de Auxiliar HTML: `Html.TextAreaFor`

Amostra:

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class DescriptionViewModel
    {
        [MinLength(5)]
        [MaxLength(1024)]
        public string Description { get; set; }
    }
}
```

```
@model DescriptionViewModel

<form asp-controller="Demo" asp-action="RegisterTextArea" method="post">
    <textarea asp-for="Description"></textarea>
    <button type="submit">Test</button>
</form>
```

O HTML a seguir é gerado:

```
<form method="post" action="/Demo/RegisterTextArea">
    <textarea data-val="true"
        data-val-maxlength="The field Description must be a string or array type with a maximum length of
        1024."
        data-val-maxlength-max="1024"
        data-val-minlength="The field Description must be a string or array type with a minimum length of
        5."
        data-val-minlength-min="5"
        id="Description" name="Description">
    </textarea>
    <button type="submit">Test</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## O auxiliar de marca de rótulo

- Gera a legenda do rótulo e o atributo `for` em um elemento para um nome de expressão
- Alternativa de Auxiliar HTML: `Html.LabelFor`

O **Label Tag Helper** fornece os seguintes benefícios em comparação com um elemento de rótulo HTML puro:

- Você obtém automaticamente o valor do rótulo descritivo do atributo `Display`. O nome de exibição desejado pode mudar com o tempo e a combinação do atributo `Display` e do Auxiliar de Marca de Rótulo aplicará `Display` em qualquer lugar em que for usado.
- Menos marcação no código-fonte
- Tipagem forte com a propriedade de modelo.

Amostra:

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class SimpleViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }
    }
}
```

```
@model SimpleViewModel

<form asp-controller="Demo" asp-action="RegisterLabel" method="post">
    <label asp-for="Email"></label>
    <input asp-for="Email" /> <br />
</form>
```

O HTML a seguir é gerado para o elemento `<label>`:

```
<label for="Email">Email Address</label>
```

O Auxiliar de marca de rótulo gerou o valor do atributo `for` de "Email", que é a ID associada ao elemento `<input>`. Auxiliares de marca geram elementos `id` e `for` consistentes para que eles possam ser associados corretamente. A legenda neste exemplo é proveniente do atributo `Display`. Se o modelo não contivesse um atributo `Display`, a legenda seria o nome da propriedade da expressão.

## Os auxiliares de marca de validação

Há dois auxiliares de marca de validação. O **Validation Message Tag Helper** (que exibe uma mensagem de validação para uma única propriedade em seu modelo) e o **Validation Summary Tag Helper** (que exibe um resumo dos erros de validação). O **Input Tag Helper** adiciona atributos de validação do lado do cliente HTML5 para elementos de entrada baseados em atributos de anotação de dados em suas classes de modelo. A validação também é executada no servidor. O Auxiliar de marca de validação exibe essas mensagens de erro quando ocorre um erro de validação.

### O Auxiliar de marca de mensagem de validação

- Adiciona o atributo `data-valmsg-for="property"` [HTML5](#) ao elemento `span`, que anexa as mensagens de erro de validação no campo de entrada da propriedade do modelo especificado. Quando ocorre um erro de validação do lado do cliente, [jQuery](#) exibe a mensagem de erro no elemento `<span>`.

- A validação também é feita no servidor. Os clientes poderão ter o JavaScript desabilitado e parte da validação só pode ser feita no lado do servidor.

- Alternativa de Auxiliar HTML: `@Html.ValidationMessageFor`

O `Validation Message Tag Helper` é usado com o atributo `asp-validation-for` em um elemento HTML `span`.

```
<span asp-validation-for="Email"></span>
```

O Auxiliar de marca de mensagem de validação gerará o HTML a seguir:

```
<span class="field-validation-valid"
      data-valmsg-for="Email"
      data-valmsg-replace="true"></span>
```

Geralmente, você usa o `Validation Message Tag Helper` após um Auxiliar de marca `Input` para a mesma propriedade. Fazer isso exibe as mensagens de erro de validação próximo à entrada que causou o erro.

#### NOTE

É necessário ter uma exibição com as referências de script `jQuery` e JavaScript corretas em vigor para a validação do lado do cliente. Consulte [Validação de Modelo](#) para obter mais informações.

Quando ocorre um erro de validação do lado do servidor (por exemplo, quando você tem validação do lado do servidor personalizada ou a validação do lado do cliente está desabilitada), o MVC coloca essa mensagem de erro como o corpo do elemento `<span>`.

```
<span class="field-validation-error" data-valmsg-for="Email"
      data-valmsg-replace="true">
    The Email Address field is required.
</span>
```

#### Auxiliar de marca de resumo de validação

- Tem como alvo elementos `<div>` com o atributo `asp-validation-summary`
- Alternativa de Auxiliar HTML: `@Html.ValidationSummary`

O `Validation Summary Tag Helper` é usado para exibir um resumo das mensagens de validação. O valor do atributo `asp-validation-summary` pode ser qualquer um dos seguintes:

ASP-VALIDATION-SUMMARY	MENSAGENS DE VALIDAÇÃO EXIBIDAS
ValidationSummary.All	Nível da propriedade e do modelo
ValidationSummary.ModelOnly	Modelo
ValidationSummary.None	Nenhum

#### Amostra

No exemplo a seguir, o modelo de dados é decorado com atributos `DataAnnotation`, o que gera mensagens de erro de validação no elemento `<input>`. Quando ocorre um erro de validação, o Auxiliar de marca de validação exibe a mensagem de erro:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}

```

```

@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterValidation" method="post">
    <div asp-validation-summary="ModelOnly"></div>
    Email: <input asp-for="Email" /> <br />
    <span asp-validation-for="Email"></span><br />
    Password: <input asp-for="Password" /><br />
    <span asp-validation-for="Password"></span><br />
    <button type="submit">Register</button>
</form>

```

O código HTML gerado (quando o modelo é válido):

```

<form action="/DemoReg/Register" method="post">
    <div class="validation-summary-valid" data-valmsg-summary="true">
        <ul><li style="display:none"></li></ul></div>
    Email: <input name="Email" id="Email" type="email" value="" data-val-required="The Email field is required." data-val-email="The Email field is not a valid email address." data-val="true"> <br />
    <span class="field-validation-valid" data-valmsg-replace="true" data-valmsg-for="Email"></span><br />
    Password: <input name="Password" id="Password" type="password" data-val-required="The Password field is required." data-val="true"><br />
    <span class="field-validation-valid" data-valmsg-replace="true" data-valmsg-for="Password"></span><br />
    <button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## Auxiliar de Marca de Seleção

- Gera `select` e os elementos `option` associados para as propriedades do modelo.
- Tem uma alternativa de Auxiliar HTML `Html.DropDownListFor` e `Html.ListBoxFor`

O `Select Tag Helper` `asp-for` especifica o nome da propriedade do modelo para o elemento `select` e `asp-items` especifica os elementos `option`. Por exemplo:

```

<select asp-for="Country" asp-items="Model.Countries"></select>

```

Amostra:

```

using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace FormsTagHelper.ViewModels
{
    public class CountryViewModel
    {
        public string Country { get; set; }

        public List<SelectListItem> Countries { get; } = new List<SelectListItem>
        {
            new SelectListItem { Value = "MX", Text = "Mexico" },
            new SelectListItem { Value = "CA", Text = "Canada" },
            new SelectListItem { Value = "US", Text = "USA" },
        };
    }
}

```

O método `Index` inicializa o `CountryViewModel`, define o país selecionado e o transmite para a exibição `Index`.

```

public IActionResult Index()
{
    var model = new CountryViewModel();
    model.Country = "CA";
    return View(model);
}

```

O método `Index` HTTP POST exibe a seleção:

```

[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Index(CountryViewModel model)
{
    if (ModelState.IsValid)
    {
        var msg = model.Country + " selected";
        return RedirectToAction("IndexSuccess", new { message = msg });
    }

    // If we got this far, something failed; redisplay form.
    return View(model);
}

```

A exibição `Index`:

```

@model CountryViewModel

<form asp-controller="Home" asp-action="Index" method="post">
    <select asp-for="Country" asp-items="Model.Countries"></select>
    <br /><button type="submit">Register</button>
</form>

```

Que gera o seguinte HTML (com "CA" selecionado):

```

<form method="post" action="/">
    <select id="Country" name="Country">
        <option value="MX">Mexico</option>
        <option selected="selected" value="CA">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## NOTE

Não é recomendável usar `ViewBag` ou `ViewData` com o Auxiliar de Marca de Seleção. Um modelo de exibição é mais robusto para fornecer metadados MVC e, geralmente, menos problemático.

O valor do atributo `asp-for` é um caso especial e não requer um prefixo `Model`, os outros atributos do Auxiliar de marca requerem (como `asp-items`)

```
<select asp-for="Country" asp-items="Model.Countries"></select>
```

## Associação de enumeração

Geralmente, é conveniente usar `<select>` com uma propriedade `enum` e gerar os elementos `SelectListItem` dos valores `enum`.

Amostra:

```

public class CountryEnumViewModel
{
    public CountryEnum EnumCountry { get; set; }
}

```

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public enum CountryEnum
    {
        [Display(Name = "United Mexican States")]
        Mexico,
        [Display(Name = "United States of America")]
        USA,
        Canada,
        France,
        Germany,
        Spain
    }
}

```

O método `GetEnumSelectList` gera um objeto `selectList` para uma enumeração.

```

@model CountryEnumViewModel

<form asp-controller="Home" asp-action="IndexEnum" method="post">
    <select asp-for="EnumCountry"
        asp-items="Html.GetEnumSelectList<CountryEnum>()">
    </select>
    <br /><button type="submit">Register</button>
</form>

```

É possível decorar sua lista de enumeradores com o atributo `Display` para obter uma interface do usuário mais rica:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public enum CountryEnum
    {
        [Display(Name = "United Mexican States")]
        Mexico,
        [Display(Name = "United States of America")]
        USA,
        Canada,
        France,
        Germany,
        Spain
    }
}

```

O HTML a seguir é gerado:

```

<form method="post" action="/Home/IndexEnum">
    <select data-val="true" data-val-required="The EnumCountry field is required." id="EnumCountry" name="EnumCountry">
        <option value="0">United Mexican States</option>
        <option value="1">United States of America</option>
        <option value="2">Canada</option>
        <option value="3">France</option>
        <option value="4">Germany</option>
        <option selected="selected" value="5">Spain</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## Grupo de opções

O elemento HTML `<optgroup>` é gerado quando o modelo de exibição contém um ou mais objetos `SelectListGroup`.

O `CountryViewModelGroup` agrupa os elementos `SelectListItem` nos grupos "América do Norte" e "Europa":

```

public class CountryViewModelGroup
{
    public CountryViewModelGroup()
    {
        var NorthAmericaGroup = new SelectListGroup { Name = "North America" };
        var EuropeGroup = new SelectListGroup { Name = "Europe" };

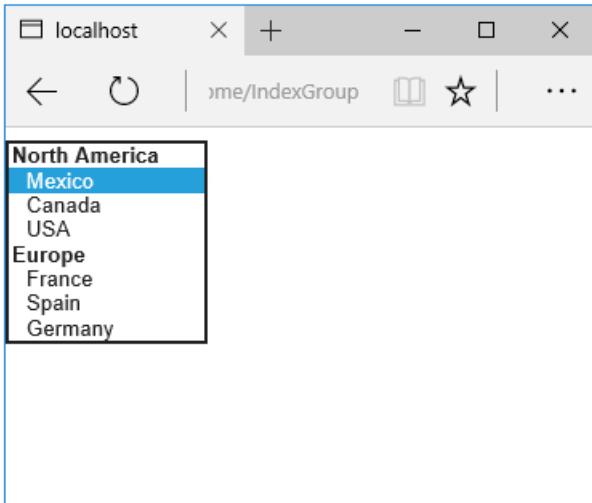
        Countries = new List<SelectListItem>
        {
            new SelectListItem
            {
                Value = "MEX",
                Text = "Mexico",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "CAN",
                Text = "Canada",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "US",
                Text = "USA",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "FR",
                Text = "France",
                Group = EuropeGroup
            },
            new SelectListItem
            {
                Value = "ES",
                Text = "Spain",
                Group = EuropeGroup
            },
            new SelectListItem
            {
                Value = "DE",
                Text = "Germany",
                Group = EuropeGroup
            }
        };
    }

    public string Country { get; set; }

    public List<SelectListItem> Countries { get; }
}

```

Os dois grupos são mostrados abaixo:



O HTML gerado:

```
<form method="post" action="/Home/IndexGroup">
    <select id="Country" name="Country">
        <optgroup label="North America">
            <option value="MEX">Mexico</option>
            <option value="CAN">Canada</option>
            <option value="US">USA</option>
        </optgroup>
        <optgroup label="Europe">
            <option value="FR">France</option>
            <option value="ES">Spain</option>
            <option value="DE">Germany</option>
        </optgroup>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="removed for brevity" />
</form>
```

## Seleção múltipla

O Auxiliar de Marca de Seleção gerará automaticamente o atributo `multiple = "multiple"` se a propriedade especificada no atributo `asp-for` for um `IEnumerable`. Por exemplo, considerando o seguinte modelo:

```
using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace FormsTagHelper.ViewModels
{
    public class CountryViewModel : IEnumerable
    {
        public IEnumerable<string> CountryCodes { get; set; }

        public List<SelectListItem> Countries { get; } = new List<SelectListItem>
        {
            new SelectListItem { Value = "MX", Text = "Mexico" },
            new SelectListItem { Value = "CA", Text = "Canada" },
            new SelectListItem { Value = "US", Text = "USA" },
            new SelectListItem { Value = "FR", Text = "France" },
            new SelectListItem { Value = "ES", Text = "Spain" },
            new SelectListItem { Value = "DE", Text = "Germany" }
        };
    }
}
```

Com a seguinte exibição:

```
@model CountryViewModelIEnumarable

<form asp-controller="Home" asp-action="IndexMultiSelect" method="post">
    <select asp-for="CountryCodes" asp-items="Model.Countries"></select>
    <br /><button type="submit">Register</button>
</form>
```

Gera o seguinte HTML:

```
<form method="post" action="/Home/IndexMultiSelect">
    <select id="CountryCodes"
        multiple="multiple"
        name="CountryCodes"><option value="MX">Mexico</option>
    <option value="CA">Canada</option>
    <option value="US">USA</option>
    <option value="FR">France</option>
    <option value="ES">Spain</option>
    <option value="DE">Germany</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## Nenhuma seleção

Se acabar usando a opção "não especificado" em várias páginas, você poderá criar um modelo para eliminar o HTML de repetição:

```
@model CountryViewModel

<form asp-controller="Home" asp-action="IndexEmpty" method="post">
    @Html.EditorForModel()
    <br /><button type="submit">Register</button>
</form>
```

O modelo *Views/Shared/EditorTemplates/CountryViewModel.cshtml*:

```
@model CountryViewModel

<select asp-for="Country" asp-items="Model.Countries">
    <option value="">--none--</option>
</select>
```

O acréscimo de elementos HTML `<option>` não está limitado ao caso de *Nenhuma seleção*. Por exemplo, o seguinte método de ação e exibição gerarão HTML semelhante ao código acima:

```
public IActionResult IndexOption(int id)
{
    var model = new CountryViewModel();
    model.Country = "CA";
    return View(model);
}
```

```
@model CountryViewModel

<form asp-controller="Home" asp-action="IndexEmpty" method="post">
    <select asp-for="Country">
        <option value="">&lt;none&gt;</option>
        <option value="MX">Mexico</option>
        <option value="CA">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
</form>
```

O elemento `<option>` correto será selecionado (contém o atributo `selected="selected"`) dependendo do valor atual de `Country`.

```
<form method="post" action="/Home/IndexEmpty">
    <select id="Country" name="Country">
        <option value="">&lt;none&gt;</option>
        <option value="MX">Mexico</option>
        <option value="CA" selected="selected">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## Recursos adicionais

- [Auxiliares de Marca no ASP.NET Core](#)
- [Elemento de formulário HTML](#)
- [Token de verificação de solicitação](#)
- [Model binding no ASP.NET Core](#)
- [Validação de modelo no ASP.NET Core MVC](#)
- [Interface IAttributeAdapter](#)
- [Snippets de código para este documento](#)

# Auxiliar de Marca de Imagem no ASP.NET Core

29/10/2018 • 3 minutes to read • [Edit Online](#)

Por [Peter Kellner](#)

O Auxiliar de Marca de Imagem aprimora a marca `<img>` para fornecer comportamento de extração de cache para arquivos de imagem estática.

Uma cadeia de caracteres de extração de cache é um valor exclusivo que representa o hash do arquivo de imagem estática acrescentado à URL do ativo. A cadeia de caracteres exclusiva solicita que os clientes (e alguns proxies) recarreguem a imagem do servidor Web do host, e não do cache do cliente.

Se a origem da imagem (`src`) for um arquivo estático no servidor Web de host:

- Uma cadeia de caracteres exclusiva de extração de cache será anexada como um parâmetro de consulta à origem da imagem.
- Se o arquivo no servidor Web host for alterado, será gerada uma URL de solicitação exclusiva que inclua o parâmetro de solicitação atualizado.

Para obter uma visão geral dos Auxiliares de Marca, confira [Auxiliares de Marca no ASP.NET Core](#).

## Atributos de Auxiliar de Marca de Imagem

### `src`

Para ativar o Auxiliar de Marca de Imagem, o atributo `src` é obrigatório no elemento `<img>`.

A origem da imagem (`src`) deve apontar para um arquivo estático físico no servidor. Se `src` for um URI remoto, o parâmetro de cadeia de caracteres de consulta de extração de cache não será gerado.

### `asp-append-version`

Quando `asp-append-version` for especificado com um valor `true` junto com um atributo `src`, o Auxiliar de Marca de Imagem será invocado.

O exemplo a seguir usa um Auxiliar de Marca de Imagem:

```
<img src "~/images/asplogo.png" asp-append-version="true" />
```

Se o arquivo estático existe no diretório `/wwwroot/images/`, o HTML gerado é semelhante ao seguinte (o hash será diferente):

```
<img src "/images/asplogo.png?v=K1_dqr9NVtnMdsM2MUg4qthUnWZm5T1fCEimBPWDNgM" />
```

O valor atribuído ao parâmetro `v` é o valor de hash do arquivo `asplogo.png` em disco. Se o servidor Web não conseguir obter acesso de leitura ao arquivo estático, nenhum parâmetro `v` será adicionado ao atributo `src` na marcação renderizada.

## Comportamento de armazenamento em cache de hash

O Auxiliar de Marca de Imagem usa o provedor de cache no servidor Web local para armazenar o hash `Sha512` calculado de um determinado arquivo. Se o arquivo for solicitado várias vezes, o hash não será recalculado. O

cache é invalidado por um observador de arquivo anexado ao arquivo quando o hash `sha512` do arquivo é calculado. Quando o arquivo muda no disco, um novo hash é calculado e armazenado em cache.

## Recursos adicionais

- [Memória de cache no ASP.NET Core](#)

# Auxiliares de marca em formulários no ASP.NET Core

14/01/2019 • 28 minutes to read • [Edit Online](#)

Por [Rick Anderson](#), [Dave Paquette](#) e [Jerrie Pelser](#)

Este documento demonstra como é o trabalho com Formulários e os elementos HTML usados comumente em um Formulário. O elemento HTML [Formulário](#) fornece o mecanismo primário que os aplicativos Web usam para postar dados para o servidor. A maior parte deste documento descreve os [Auxiliares de marca](#) e como eles podem ajudar você a criar formulários HTML robustos de forma produtiva. É recomendável que você leia [Introdução ao auxiliares de marca](#) antes de ler este documento.

Em muitos casos, os Auxiliares HTML fornecem uma abordagem alternativa a um Auxiliar de Marca específico, mas é importante reconhecer que os Auxiliares de Marca não substituem os Auxiliares HTML e que não há um Auxiliar de Marca para cada Auxiliar HTML. Quando existe um Auxiliar HTML alternativo, ele é mencionado.

## O Auxiliar de marca de formulário

O Auxiliar de marca de [formulário](#):

- Gera o valor do atributo HTML `<FORM>` `action` para uma ação do controlador MVC ou uma rota nomeada
- Gera um [Token de verificação de solicitação](#) oculto para evitar a falsificação de solicitações entre sites (quando usado com o atributo `[ValidateAntiForgeryToken]` no método de ação HTTP Post)
- Fornece o atributo `asp-route-<Parameter Name>`, em que `<Parameter Name>` é adicionado aos valores de rota. Os parâmetros `routeValues` para `Html.BeginForm` e `Html.BeginRouteForm` fornecem funcionalidade semelhante.
- Tem uma alternativa de Auxiliar HTML `Html.BeginForm` e `Html.BeginRouteForm`

Amostra:

```
<form asp-controller="Demo" asp-action="Register" method="post">
    <!-- Input and Submit elements -->
</form>
```

O Auxiliar de marca de formulário acima gera o HTML a seguir:

```
<form method="post" action="/Demo/Register">
    <!-- Input and Submit elements -->
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

O tempo de execução do MVC gera o valor do atributo `action` dos atributos `asp-controller` e `asp-action` do Auxiliar de marca de formulário. O Auxiliar de marca de formulário também gera um [Token de verificação de solicitação](#) oculto para evitar a falsificação de solicitações entre sites (quando usado com o atributo `[ValidateAntiForgeryToken]` no método de ação HTTP Post). É difícil proteger um Formulário HTML puro contra falsificação de solicitações entre sites e o Auxiliar de marca de formulário fornece este serviço para você.

### Usando uma rota nomeada

O atributo do Auxiliar de Marca `asp-route` também pode gerar a marcação para o atributo HTML `action`. Um

aplicativo com uma [rota](#) chamada `register` poderia usar a seguinte marcação para a página de registro:

```
<form asp-route="register" method="post">
    <!-- Input and Submit elements -->
</form>
```

Muitas das exibições na pasta *Modos de Exibição/Conta* (gerada quando você cria um novo aplicativo Web com *Contas de usuário individuais*) contêm o atributo [asp-route-returnurl](#):

```
<form asp-controller="Account" asp-action="Login"
      asp-route-returnurl="@ViewData["ReturnUrl"]"
      method="post" class="form-horizontal" role="form">
```

#### NOTE

Com os modelos internos, `returnUrl` só é preenchido automaticamente quando você tenta acessar um recurso autorizado, mas não está autenticado ou autorizado. Quando você tenta fazer um acesso não autorizado, o middleware de segurança o redireciona para a página de logon com o `returnUrl` definido.

## O auxiliar de marca de entrada

O Auxiliar de marca de entrada associa um elemento HTML `<input>` a uma expressão de modelo em sua exibição do Razor.

Sintaxe:

```
<input asp-for="<Expression Name>" />
```

O auxiliar de marca de entrada:

- Gera os atributos HTML `id` e `name` para o nome da expressão especificada no atributo `asp-for`.  
`asp-for="Property1.Property2"` equivale a `m => m.Property1.Property2`. O nome da expressão é o que é usado para o valor do atributo `asp-for`. Consulte a seção [Nomes de expressão](#) para obter informações adicionais.
- Define o valor do atributo HTML `type` com base nos atributos de tipo de modelo e [anotação de dados](#) aplicados à propriedade de modelo
- O valor do atributo HTML `type` não será substituído quando um for especificado
- Gera atributos de validação [HTML5](#) de atributos de [anotação de dados](#) aplicados a propriedades de modelo
- Tem uma sobreposição de recursos de Auxiliar HTML com `Html.TextBoxFor` e `Html.EditorFor`. Consulte a seção **Alternativas de Auxiliar HTML ao Auxiliar de marca de entrada** para obter detalhes.
- Fornece tipagem forte. Se o nome da propriedade for alterado e você não atualizar o Auxiliar de marca, você verá um erro semelhante ao seguinte:

An error occurred during the compilation of a resource required to process this request. Please review the following specific error details and modify your source code appropriately.

Type expected

'RegisterViewModel' does not contain a definition for 'Email' and no extension method 'Email' accepting a first argument of type 'RegisterViewModel' could be found (are you missing a using directive or an assembly reference?)

O Auxiliar de marca `Input` define o atributo HTML `type` com base no tipo .NET. A tabela a seguir lista alguns tipos .NET comuns e o tipo HTML gerado (não estão listados todos os tipos .NET).

TIPO .NET	TIPO DE ENTRADA
Bool	<code>type="checkbox"</code>
Cadeia de Caracteres	<code>type="text"</code>
DateTime	<code>type="datetime-local"</code>
Byte	<code>type="number"</code>
int	<code>type="number"</code>
Single e Double	<code>type="number"</code>

A tabela a seguir mostra alguns atributos de [anotações de dados](#) comuns que o auxiliar de marca de entrada mapeará para tipos de entrada específicos (não são listados todos os atributos de validação):

ATRIBUTO	TIPO DE ENTRADA
<code>[EmailAddress]</code>	<code>type="email"</code>
<code>[Url]</code>	<code>type="url"</code>
<code>[HiddenInput]</code>	<code>type="hidden"</code>
<code>[Phone]</code>	<code>type="tel"</code>
<code>[DataType(DataType.Password)]</code>	<code>type="password"</code>
<code>[DataType(DataType.Date)]</code>	<code>type="date"</code>
<code>[DataType(DataType.Time)]</code>	<code>type="time"</code>

Amostra:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}

```

```

@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterInput" method="post">
    Email: <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    <button type="submit">Register</button>
</form>

```

O código acima gera o seguinte HTML:

```

<form method="post" action="/Demo/RegisterInput">
    Email:
    <input type="email" data-val="true"
           data-val-email="The Email Address field is not a valid email address."
           data-val-required="The Email Address field is required."
           id="Email" name="Email" value="" /> <br>
    Password:
    <input type="password" data-val="true"
           data-val-required="The Password field is required."
           id="Password" name="Password" /><br>
    <button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

As anotações de dados aplicadas às propriedades `Email` e `Password` geram metadados no modelo. O Auxiliar de marca de entrada consome os metadados do modelo e produz atributos HTML5 `data-val-*` (consulte [Validação de modelo](#)). Esses atributos descrevem os validadores a serem anexados aos campos de entrada. Isso fornece validação de [jQuery](#) e HTML5 discreto. Os atributos discretos têm o formato `data-val-rule="Error Message"`, em que a regra é o nome da regra de validação (como `data-val-required`, `data-val-email`, `data-val-maxlength` etc.). Se uma mensagem de erro for fornecida no atributo, ela será exibida como o valor para o atributo `data-val-rule`. Também há atributos do formulário `data-val-ruleName-argumentName="argumentValue"` que fornecem detalhes adicionais sobre a regra, por exemplo, `data-val-maxlength-max="1024"`.

## Alternativas de Auxiliar HTML ao Auxiliar de marca de entrada

`Html.TextBox`, `Html.TextBoxFor`, `Html.Editor` e `Html.EditorFor` têm recursos que se sobrepõem aos do Auxiliar de marca de entrada. O Auxiliar de marca de entrada define automaticamente o atributo `type`; `Html.TextBox` e `Html.TextBoxFor` não o fazem. `Html.Editor` e `Html.EditorFor` manipulam coleções, objetos complexos e modelos; o Auxiliar de marca de entrada não o faz. O Auxiliar de marca de entrada, `Html.EditorFor` e `Html.TextBoxFor` são fortemente tipados (eles usam expressões lambda); `Html.TextBox` e `Html.Editor` não usam (eles usam nomes de expressão).

## HtmlAttributes

`@Html.Editor()` e `@Html.EditorFor()` usam uma entrada `ViewDataDictionary` especial chamada `htmlAttributes` ao executar seus modelos padrão. Esse comportamento pode ser aumentado usando parâmetros `additional ViewData`. A chave "htmlAttributes" diferencia maiúsculas de minúsculas. A chave "htmlAttributes" é tratada de forma semelhante ao objeto `htmlAttributes` passado para auxiliares de entrada como `@Html.TextBox()`.

```
@Html.EditorFor(model => model.YourProperty,  
new { htmlAttributes = new { @class="myCssClass", style="Width:100px" } })
```

## Nomes de expressão

O valor do atributo `asp-for` é um `ModelExpression` e o lado direito de uma expressão lambda. Portanto, `asp-for="Property1"` se torna `m => m.Property1` no código gerado e é por isso você não precisa colocar o prefixo `Model`. Você pode usar o caractere "@" para iniciar uma expressão embutida e mover para antes de `m`:

```
@{  
    var joe = "Joe";  
}  
<input asp-for="@joe" />
```

Gera o seguinte:

```
<input type="text" id="joe" name="joe" value="Joe" />
```

Com propriedades de coleção, `asp-for="CollectionProperty[23].Member"` gera o mesmo nome que `asp-for="CollectionProperty[i].Member"` quando `i` tem o valor `23`.

Quando o ASP.NET Core MVC calcula o valor de `ModelExpression`, ele inspeciona várias fontes, inclusive o `ModelState`. Considere o `<input type="text" asp-for="@Name" />`. O atributo `value` calculado é o primeiro valor não nulo:

- Da entrada de `ModelState` com a chave "Name".
- Do resultado da expressão `Model.Name`.

## Navegando para propriedades filho

Você também pode navegar para propriedades filho usando o caminho da propriedade do modelo de exibição. Considere uma classe de modelo mais complexa que contém uma propriedade `Address` filho.

```
public class AddressViewModel  
{  
    public string AddressLine1 { get; set; }  
}
```

```
public class RegisterAddressViewModel  
{  
    public string Email { get; set; }  
  
    [DataType(DataType.Password)]  
    public string Password { get; set; }  
  
    public AddressViewModel Address { get; set; }  
}
```

Na exibição, associamos a `Address.AddressLine1`:

```
@model RegisterAddressViewModel

<form asp-controller="Demo" asp-action="RegisterAddress" method="post">
    Email: <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    Address: <input asp-for="Address.AddressLine1" /><br />
    <button type="submit">Register</button>
</form>
```

O HTML a seguir é gerado para `Address.AddressLine1`:

```
<input type="text" id="Address_AddressLine1" name="Address.AddressLine1" value="" />
```

## Nomes de expressão e coleções

Exemplo, um modelo que contém uma matriz de `Colors`:

```
public class Person
{
    public List<string> Colors { get; set; }

    public int Age { get; set; }
}
```

O método de ação:

```
public IActionResult Edit(int id, int colorIndex)
{
    ViewData["Index"] = colorIndex;
    return View(GetPerson(id));
}
```

O Razor a seguir mostra como você acessa um elemento `Color` específico:

```
@model Person
 @{
     var index = (int)ViewData["index"];
 }

<form asp-controller="ToDo" asp-action="Edit" method="post">
    @Html.EditorFor(m => m.Colors[index])
    <label asp-for="Age"></label>
    <input asp-for="Age" /><br />
    <button type="submit">Post</button>
</form>
```

O modelo `Views/Shared/EditorTemplates/String.cshtml`:

```
@model string

<label asp-for="@Model"></label>
<input asp-for="@Model" /> <br />
```

Exemplo usando `List<T>`:

```

public class ToDoItem
{
    public string Name { get; set; }

    public bool IsDone { get; set; }
}

```

O Razor a seguir mostra como iterar em uma coleção:

```

@model List<ToDoItem>

<form asp-controller="ToDo" asp-action="Edit" method="post">
    <table>
        <tr> <th>Name</th> <th>Is Done</th> </tr>

        @for (int i = 0; i < Model.Count; i++)
        {
            <tr>
                @Html.EditorFor(model => model[i])
            </tr>
        }

    </table>
    <button type="submit">Save</button>
</form>

```

O modelo *Views/Shared/EditorTemplates/ToDoItem.cshtml*:

```

@model ToDoItem

<td>
    <label asp-for="@Model.Name"></label>
    @Html.DisplayFor(model => model.Name)
</td>
<td>
    <input asp-for="@Model.IsDone" />
</td>

/*
This template replaces the following Razor which evaluates the indexer three times.
<td>
    <label asp-for="@Model[i].Name"></label>
    @Html.DisplayFor(model => model[i].Name)
</td>
<td>
    <input asp-for="@Model[i].IsDone" />
</td>
*/

```

`foreach` deve ser usado, se possível, quando o valor está prestes a ser usado em um contexto equivalente `asp-for` ou `Html.DisplayFor`. Em geral, `for` é melhor do que `foreach` (se o cenário permitir) porque não é necessário alocar um enumerador; no entanto, avaliar um indexador em uma expressão LINQ pode ser caro, o que deve ser minimizado.

#### NOTE

O código de exemplo comentado acima mostra como você substituiria a expressão lambda pelo operador `@` para acessar cada `ToDoItem` na lista.

## Auxiliar de marca de área de texto

O auxiliar de marca `Textarea Tag Helper` é semelhante ao Auxiliar de marca de entrada.

- Gera os atributos `id` e `name`, bem como os atributos de validação de dados do modelo para um elemento `<textarea>`.
- Fornece tipagem forte.
- Alternativa de Auxiliar HTML: `Html.TextAreaFor`

Amostra:

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class DescriptionViewModel
    {
        [MinLength(5)]
        [MaxLength(1024)]
        public string Description { get; set; }
    }
}
```

```
@model DescriptionViewModel

<form asp-controller="Demo" asp-action="RegisterTextArea" method="post">
    <textarea asp-for="Description"></textarea>
    <button type="submit">Test</button>
</form>
```

O HTML a seguir é gerado:

```
<form method="post" action="/Demo/RegisterTextArea">
    <textarea data-val="true"
        data-val-maxlength="The field Description must be a string or array type with a maximum length of
        1024."
        data-val-maxlength-max="1024"
        data-val-minlength="The field Description must be a string or array type with a minimum length of
        5."
        data-val-minlength-min="5"
        id="Description" name="Description">
    </textarea>
    <button type="submit">Test</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## O auxiliar de marca de rótulo

- Gera a legenda do rótulo e o atributo `for` em um elemento para um nome de expressão
- Alternativa de Auxiliar HTML: `Html.LabelFor`

O **Label Tag Helper** fornece os seguintes benefícios em comparação com um elemento de rótulo HTML puro:

- Você obtém automaticamente o valor do rótulo descritivo do atributo `Display`. O nome de exibição desejado pode mudar com o tempo e a combinação do atributo `Display` e do Auxiliar de Marca de Rótulo aplicará `Display` em qualquer lugar em que for usado.
- Menos marcação no código-fonte
- Tipagem forte com a propriedade de modelo.

Amostra:

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class SimpleViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }
    }
}
```

```
@model SimpleViewModel

<form asp-controller="Demo" asp-action="RegisterLabel" method="post">
    <label asp-for="Email"></label>
    <input asp-for="Email" /> <br />
</form>
```

O HTML a seguir é gerado para o elemento `<label>`:

```
<label for="Email">Email Address</label>
```

O Auxiliar de marca de rótulo gerou o valor do atributo `for` de "Email", que é a ID associada ao elemento `<input>`. Auxiliares de marca geram elementos `id` e `for` consistentes para que eles possam ser associados corretamente. A legenda neste exemplo é proveniente do atributo `Display`. Se o modelo não contivesse um atributo `Display`, a legenda seria o nome da propriedade da expressão.

## Os auxiliares de marca de validação

Há dois auxiliares de marca de validação. O **Validation Message Tag Helper** (que exibe uma mensagem de validação para uma única propriedade em seu modelo) e o **Validation Summary Tag Helper** (que exibe um resumo dos erros de validação). O **Input Tag Helper** adiciona atributos de validação do lado do cliente HTML5 para elementos de entrada baseados em atributos de anotação de dados em suas classes de modelo. A validação também é executada no servidor. O Auxiliar de marca de validação exibe essas mensagens de erro quando ocorre um erro de validação.

### O Auxiliar de marca de mensagem de validação

- Adiciona o atributo `data-valmsg-for="property"` [HTML5](#) ao elemento `span`, que anexa as mensagens de erro de validação no campo de entrada da propriedade do modelo especificado. Quando ocorre um erro de validação do lado do cliente, [jQuery](#) exibe a mensagem de erro no elemento `<span>`.

- A validação também é feita no servidor. Os clientes poderão ter o JavaScript desabilitado e parte da validação só pode ser feita no lado do servidor.

- Alternativa de Auxiliar HTML: `@Html.ValidationMessageFor`

O `Validation Message Tag Helper` é usado com o atributo `asp-validation-for` em um elemento HTML `span`.

```
<span asp-validation-for="Email"></span>
```

O Auxiliar de marca de mensagem de validação gerará o HTML a seguir:

```
<span class="field-validation-valid"
      data-valmsg-for="Email"
      data-valmsg-replace="true"></span>
```

Geralmente, você usa o `Validation Message Tag Helper` após um Auxiliar de marca `Input` para a mesma propriedade. Fazer isso exibe as mensagens de erro de validação próximo à entrada que causou o erro.

#### NOTE

É necessário ter uma exibição com as referências de script `jQuery` e JavaScript corretas em vigor para a validação do lado do cliente. Consulte [Validação de Modelo](#) para obter mais informações.

Quando ocorre um erro de validação do lado do servidor (por exemplo, quando você tem validação do lado do servidor personalizada ou a validação do lado do cliente está desabilitada), o MVC coloca essa mensagem de erro como o corpo do elemento `<span>`.

```
<span class="field-validation-error" data-valmsg-for="Email"
      data-valmsg-replace="true">
    The Email Address field is required.
</span>
```

#### Auxiliar de marca de resumo de validação

- Tem como alvo elementos `<div>` com o atributo `asp-validation-summary`
- Alternativa de Auxiliar HTML: `@Html.ValidationSummary`

O `Validation Summary Tag Helper` é usado para exibir um resumo das mensagens de validação. O valor do atributo `asp-validation-summary` pode ser qualquer um dos seguintes:

ASP-VALIDATION-SUMMARY	MENSAGENS DE VALIDAÇÃO EXIBIDAS
ValidationSummary.All	Nível da propriedade e do modelo
ValidationSummary.ModelOnly	Modelo
ValidationSummary.None	Nenhum

#### Amostra

No exemplo a seguir, o modelo de dados é decorado com atributos `DataAnnotation`, o que gera mensagens de erro de validação no elemento `<input>`. Quando ocorre um erro de validação, o Auxiliar de marca de validação exibe a mensagem de erro:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}

```

```

@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterValidation" method="post">
    <div asp-validation-summary="ModelOnly"></div>
    Email: <input asp-for="Email" /> <br />
    <span asp-validation-for="Email"></span><br />
    Password: <input asp-for="Password" /><br />
    <span asp-validation-for="Password"></span><br />
    <button type="submit">Register</button>
</form>

```

O código HTML gerado (quando o modelo é válido):

```

<form action="/DemoReg/Register" method="post">
    <div class="validation-summary-valid" data-valmsg-summary="true">
        <ul><li style="display:none"></li></ul></div>
    Email: <input name="Email" id="Email" type="email" value="" data-val-required="The Email field is required." data-val-email="The Email field is not a valid email address." data-val="true"> <br />
    <span class="field-validation-valid" data-valmsg-replace="true" data-valmsg-for="Email"></span><br />
    Password: <input name="Password" id="Password" type="password" data-val-required="The Password field is required." data-val="true"><br />
    <span class="field-validation-valid" data-valmsg-replace="true" data-valmsg-for="Password"></span><br />
    <button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## Auxiliar de Marca de Seleção

- Gera `select` e os elementos `option` associados para as propriedades do modelo.
- Tem uma alternativa de Auxiliar HTML `Html.DropDownListFor` e `Html.ListBoxFor`

O `Select Tag Helper` `asp-for` especifica o nome da propriedade do modelo para o elemento `select` e `asp-items` especifica os elementos `option`. Por exemplo:

```
<select asp-for="Country" asp-items="Model.Countries"></select>
```

Amostra:

```

using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace FormsTagHelper.ViewModels
{
    public class CountryViewModel
    {
        public string Country { get; set; }

        public List<SelectListItem> Countries { get; } = new List<SelectListItem>
        {
            new SelectListItem { Value = "MX", Text = "Mexico" },
            new SelectListItem { Value = "CA", Text = "Canada" },
            new SelectListItem { Value = "US", Text = "USA" },
        };
    }
}

```

O método `Index` inicializa o `CountryViewModel`, define o país selecionado e o transmite para a exibição `Index`.

```

public IActionResult Index()
{
    var model = new CountryViewModel();
    model.Country = "CA";
    return View(model);
}

```

O método `Index` HTTP POST exibe a seleção:

```

[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Index(CountryViewModel model)
{
    if (ModelState.IsValid)
    {
        var msg = model.Country + " selected";
        return RedirectToAction("IndexSuccess", new { message = msg });
    }

    // If we got this far, something failed; redisplay form.
    return View(model);
}

```

A exibição `Index`:

```

@model CountryViewModel

<form asp-controller="Home" asp-action="Index" method="post">
    <select asp-for="Country" asp-items="Model.Countries"></select>
    <br /><button type="submit">Register</button>
</form>

```

Que gera o seguinte HTML (com "CA" selecionado):

```

<form method="post" action="/">
    <select id="Country" name="Country">
        <option value="MX">Mexico</option>
        <option selected="selected" value="CA">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## NOTE

Não é recomendável usar `ViewBag` ou  `ViewData` com o Auxiliar de Marca de Seleção. Um modelo de exibição é mais robusto para fornecer metadados MVC e, geralmente, menos problemático.

O valor do atributo `asp-for` é um caso especial e não requer um prefixo `Model`, os outros atributos do Auxiliar de marca requerem (como `asp-items`)

```
<select asp-for="Country" asp-items="Model.Countries"></select>
```

## Associação de enumeração

Geralmente, é conveniente usar `<select>` com uma propriedade `enum` e gerar os elementos `SelectListItem` dos valores `enum`.

Amostra:

```

public class CountryEnumViewModel
{
    public CountryEnum EnumCountry { get; set; }
}

```

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public enum CountryEnum
    {
        [Display(Name = "United Mexican States")]
        Mexico,
        [Display(Name = "United States of America")]
        USA,
        Canada,
        France,
        Germany,
        Spain
    }
}

```

O método `GetEnumSelectList` gera um objeto `selectList` para uma enumeração.

```

@model CountryEnumViewModel

<form asp-controller="Home" asp-action="IndexEnum" method="post">
    <select asp-for="EnumCountry"
            asp-items="Html.GetEnumSelectList<CountryEnum>()">
    </select>
    <br /><button type="submit">Register</button>
</form>

```

É possível decorar sua lista de enumeradores com o atributo `Display` para obter uma interface do usuário mais rica:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public enum CountryEnum
    {
        [Display(Name = "United Mexican States")]
        Mexico,
        [Display(Name = "United States of America")]
        USA,
        Canada,
        France,
        Germany,
        Spain
    }
}

```

O HTML a seguir é gerado:

```

<form method="post" action="/Home/IndexEnum">
    <select data-val="true" data-val-required="The EnumCountry field is required." id="EnumCountry" name="EnumCountry">
        <option value="0">United Mexican States</option>
        <option value="1">United States of America</option>
        <option value="2">Canada</option>
        <option value="3">France</option>
        <option value="4">Germany</option>
        <option selected="selected" value="5">Spain</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## Grupo de opções

O elemento HTML `<optgroup>` é gerado quando o modelo de exibição contém um ou mais objetos `SelectListGroup`.

O `CountryViewModelGroup` agrupa os elementos `SelectListItem` nos grupos "América do Norte" e "Europa":

```

public class CountryViewModelGroup
{
    public CountryViewModelGroup()
    {
        var NorthAmericaGroup = new SelectListGroup { Name = "North America" };
        var EuropeGroup = new SelectListGroup { Name = "Europe" };

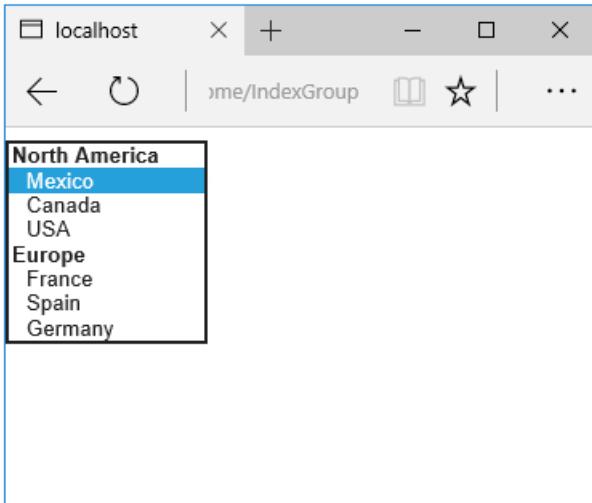
        Countries = new List<SelectListItem>
        {
            new SelectListItem
            {
                Value = "MEX",
                Text = "Mexico",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "CAN",
                Text = "Canada",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "US",
                Text = "USA",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "FR",
                Text = "France",
                Group = EuropeGroup
            },
            new SelectListItem
            {
                Value = "ES",
                Text = "Spain",
                Group = EuropeGroup
            },
            new SelectListItem
            {
                Value = "DE",
                Text = "Germany",
                Group = EuropeGroup
            }
        };
    }

    public string Country { get; set; }

    public List<SelectListItem> Countries { get; }
}

```

Os dois grupos são mostrados abaixo:



O HTML gerado:

```
<form method="post" action="/Home/IndexGroup">
    <select id="Country" name="Country">
        <optgroup label="North America">
            <option value="MEX">Mexico</option>
            <option value="CAN">Canada</option>
            <option value="US">USA</option>
        </optgroup>
        <optgroup label="Europe">
            <option value="FR">France</option>
            <option value="ES">Spain</option>
            <option value="DE">Germany</option>
        </optgroup>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="removed for brevity" />
</form>
```

## Seleção múltipla

O Auxiliar de Marca de Seleção gerará automaticamente o atributo `multiple = "multiple"` se a propriedade especificada no atributo `asp-for` for um `IEnumerable`. Por exemplo, considerando o seguinte modelo:

```
using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace FormsTagHelper.ViewModels
{
    public class CountryViewModel : IEnumerable
    {
        public IEnumerable<string> CountryCodes { get; set; }

        public List<SelectListItem> Countries { get; } = new List<SelectListItem>
        {
            new SelectListItem { Value = "MX", Text = "Mexico" },
            new SelectListItem { Value = "CA", Text = "Canada" },
            new SelectListItem { Value = "US", Text = "USA" },
            new SelectListItem { Value = "FR", Text = "France" },
            new SelectListItem { Value = "ES", Text = "Spain" },
            new SelectListItem { Value = "DE", Text = "Germany" }
        };
    }
}
```

Com a seguinte exibição:

```
@model CountryViewModelIEnumarable

<form asp-controller="Home" asp-action="IndexMultiSelect" method="post">
    <select asp-for="CountryCodes" asp-items="Model.Countries"></select>
    <br /><button type="submit">Register</button>
</form>
```

Gera o seguinte HTML:

```
<form method="post" action="/Home/IndexMultiSelect">
    <select id="CountryCodes"
        multiple="multiple"
        name="CountryCodes"><option value="MX">Mexico</option>
    <option value="CA">Canada</option>
    <option value="US">USA</option>
    <option value="FR">France</option>
    <option value="ES">Spain</option>
    <option value="DE">Germany</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## Nenhuma seleção

Se acabar usando a opção "não especificado" em várias páginas, você poderá criar um modelo para eliminar o HTML de repetição:

```
@model CountryViewModel

<form asp-controller="Home" asp-action="IndexEmpty" method="post">
    @Html.EditorForModel()
    <br /><button type="submit">Register</button>
</form>
```

O modelo *Views/Shared/EditorTemplates/CountryViewModel.cshtml*:

```
@model CountryViewModel

<select asp-for="Country" asp-items="Model.Countries">
    <option value="">--none--</option>
</select>
```

O acréscimo de elementos HTML `<option>` não está limitado ao caso de *Nenhuma seleção*. Por exemplo, o seguinte método de ação e exibição gerarão HTML semelhante ao código acima:

```
public IActionResult IndexOption(int id)
{
    var model = new CountryViewModel();
    model.Country = "CA";
    return View(model);
}
```

```
@model CountryViewModel

<form asp-controller="Home" asp-action="IndexEmpty" method="post">
    <select asp-for="Country">
        <option value="">&lt;none&gt;</option>
        <option value="MX">Mexico</option>
        <option value="CA">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
</form>
```

O elemento `<option>` correto será selecionado (contém o atributo `selected="selected"`) dependendo do valor atual de `Country`.

```
<form method="post" action="/Home/IndexEmpty">
    <select id="Country" name="Country">
        <option value="">&lt;none&gt;</option>
        <option value="MX">Mexico</option>
        <option value="CA" selected="selected">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## Recursos adicionais

- [Auxiliares de Marca no ASP.NET Core](#)
- [Elemento de formulário HTML](#)
- [Token de verificação de solicitação](#)
- [Model binding no ASP.NET Core](#)
- [Validação de modelo no ASP.NET Core MVC](#)
- [Interface IAttributeAdapter](#)
- [Snippets de código para este documento](#)

# Auxiliares de marca em formulários no ASP.NET Core

14/01/2019 • 28 minutes to read • [Edit Online](#)

Por [Rick Anderson](#), [Dave Paquette](#) e [Jerrie Pelser](#)

Este documento demonstra como é o trabalho com Formulários e os elementos HTML usados comumente em um Formulário. O elemento HTML [Formulário](#) fornece o mecanismo primário que os aplicativos Web usam para postar dados para o servidor. A maior parte deste documento descreve os [Auxiliares de marca](#) e como eles podem ajudar você a criar formulários HTML robustos de forma produtiva. É recomendável que você leia [Introdução ao auxiliares de marca](#) antes de ler este documento.

Em muitos casos, os Auxiliares HTML fornecem uma abordagem alternativa a um Auxiliar de Marca específico, mas é importante reconhecer que os Auxiliares de Marca não substituem os Auxiliares HTML e que não há um Auxiliar de Marca para cada Auxiliar HTML. Quando existe um Auxiliar HTML alternativo, ele é mencionado.

## O Auxiliar de marca de formulário

O Auxiliar de marca de [formulário](#):

- Gera o valor do atributo HTML `<FORM>` `action` para uma ação do controlador MVC ou uma rota nomeada
- Gera um [Token de verificação de solicitação](#) oculto para evitar a falsificação de solicitações entre sites (quando usado com o atributo `[ValidateAntiForgeryToken]` no método de ação HTTP Post)
- Fornece o atributo `asp-route-<Parameter Name>`, em que `<Parameter Name>` é adicionado aos valores de rota. Os parâmetros `routeValues` para `Html.BeginForm` e `Html.BeginRouteForm` fornecem funcionalidade semelhante.
- Tem uma alternativa de Auxiliar HTML `Html.BeginForm` e `Html.BeginRouteForm`

Amostra:

```
<form asp-controller="Demo" asp-action="Register" method="post">
    <!-- Input and Submit elements -->
</form>
```

O Auxiliar de marca de formulário acima gera o HTML a seguir:

```
<form method="post" action="/Demo/Register">
    <!-- Input and Submit elements -->
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

O tempo de execução do MVC gera o valor do atributo `action` dos atributos `asp-controller` e `asp-action` do Auxiliar de marca de formulário. O Auxiliar de marca de formulário também gera um [Token de verificação de solicitação](#) oculto para evitar a falsificação de solicitações entre sites (quando usado com o atributo `[ValidateAntiForgeryToken]` no método de ação HTTP Post). É difícil proteger um Formulário HTML puro contra falsificação de solicitações entre sites e o Auxiliar de marca de formulário fornece este serviço para você.

### Usando uma rota nomeada

O atributo do Auxiliar de Marca `asp-route` também pode gerar a marcação para o atributo HTML `action`. Um

aplicativo com uma [rota](#) chamada `register` poderia usar a seguinte marcação para a página de registro:

```
<form asp-route="register" method="post">
    <!-- Input and Submit elements -->
</form>
```

Muitas das exibições na pasta *Modos de Exibição/Conta* (gerada quando você cria um novo aplicativo Web com *Contas de usuário individuais*) contêm o atributo [asp-route-returnurl](#):

```
<form asp-controller="Account" asp-action="Login"
      asp-route-returnurl="@ViewData["ReturnUrl"]"
      method="post" class="form-horizontal" role="form">
```

#### NOTE

Com os modelos internos, `returnUrl` só é preenchido automaticamente quando você tenta acessar um recurso autorizado, mas não está autenticado ou autorizado. Quando você tenta fazer um acesso não autorizado, o middleware de segurança o redireciona para a página de logon com o `returnUrl` definido.

## O auxiliar de marca de entrada

O Auxiliar de marca de entrada associa um elemento HTML `<input>` a uma expressão de modelo em sua exibição do Razor.

Sintaxe:

```
<input asp-for="<Expression Name>" />
```

O auxiliar de marca de entrada:

- Gera os atributos HTML `id` e `name` para o nome da expressão especificada no atributo `asp-for`.  
`asp-for="Property1.Property2"` equivale a `m => m.Property1.Property2`. O nome da expressão é o que é usado para o valor do atributo `asp-for`. Consulte a seção [Nomes de expressão](#) para obter informações adicionais.
- Define o valor do atributo HTML `type` com base nos atributos de tipo de modelo e [anotação de dados](#) aplicados à propriedade de modelo
- O valor do atributo HTML `type` não será substituído quando um for especificado
- Gera atributos de validação [HTML5](#) de atributos de [anotação de dados](#) aplicados a propriedades de modelo
- Tem uma sobreposição de recursos de Auxiliar HTML com `Html.TextBoxFor` e `Html.EditorFor`. Consulte a seção **Alternativas de Auxiliar HTML ao Auxiliar de marca de entrada** para obter detalhes.
- Fornece tipagem forte. Se o nome da propriedade for alterado e você não atualizar o Auxiliar de marca, você verá um erro semelhante ao seguinte:

An error occurred during the compilation of a resource required to process this request. Please review the following specific error details and modify your source code appropriately.

Type expected

'RegisterViewModel' does not contain a definition for 'Email' and no extension method 'Email' accepting a first argument of type 'RegisterViewModel' could be found (are you missing a using directive or an assembly reference?)

O Auxiliar de marca `Input` define o atributo HTML `type` com base no tipo .NET. A tabela a seguir lista alguns tipos .NET comuns e o tipo HTML gerado (não estão listados todos os tipos .NET).

TIPO .NET	TIPO DE ENTRADA
Bool	<code>type="checkbox"</code>
Cadeia de Caracteres	<code>type="text"</code>
DateTime	<code>type="datetime-local"</code>
Byte	<code>type="number"</code>
int	<code>type="number"</code>
Single e Double	<code>type="number"</code>

A tabela a seguir mostra alguns atributos de [anotações de dados](#) comuns que o auxiliar de marca de entrada mapeará para tipos de entrada específicos (não são listados todos os atributos de validação):

ATRIBUTO	TIPO DE ENTRADA
<code>[EmailAddress]</code>	<code>type="email"</code>
<code>[Url]</code>	<code>type="url"</code>
<code>[HiddenInput]</code>	<code>type="hidden"</code>
<code>[Phone]</code>	<code>type="tel"</code>
<code>[DataType(DataType.Password)]</code>	<code>type="password"</code>
<code>[DataType(DataType.Date)]</code>	<code>type="date"</code>
<code>[DataType(DataType.Time)]</code>	<code>type="time"</code>

Amostra:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}

```

```

@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterInput" method="post">
    Email: <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    <button type="submit">Register</button>
</form>

```

O código acima gera o seguinte HTML:

```

<form method="post" action="/Demo/RegisterInput">
    Email:
    <input type="email" data-val="true"
           data-val-email="The Email Address field is not a valid email address."
           data-val-required="The Email Address field is required."
           id="Email" name="Email" value="" /> <br>
    Password:
    <input type="password" data-val="true"
           data-val-required="The Password field is required."
           id="Password" name="Password" /><br>
    <button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

As anotações de dados aplicadas às propriedades `Email` e `Password` geram metadados no modelo. O Auxiliar de marca de entrada consome os metadados do modelo e produz atributos HTML5 `data-val-*` (consulte [Validação de modelo](#)). Esses atributos descrevem os validadores a serem anexados aos campos de entrada. Isso fornece validação de [jQuery](#) e HTML5 discreto. Os atributos discretos têm o formato `data-val-rule="Error Message"`, em que a regra é o nome da regra de validação (como `data-val-required`, `data-val-email`, `data-val-maxlength` etc.). Se uma mensagem de erro for fornecida no atributo, ela será exibida como o valor para o atributo `data-val-rule`. Também há atributos do formulário `data-val-ruleName-argumentName="argumentValue"` que fornecem detalhes adicionais sobre a regra, por exemplo, `data-val-maxlength-max="1024"`.

## Alternativas de Auxiliar HTML ao Auxiliar de marca de entrada

`Html.TextBox`, `Html.TextBoxFor`, `Html.Editor` e `Html.EditorFor` têm recursos que se sobrepõem aos do Auxiliar de marca de entrada. O Auxiliar de marca de entrada define automaticamente o atributo `type`; `Html.TextBox` e `Html.TextBoxFor` não o fazem. `Html.Editor` e `Html.EditorFor` manipulam coleções, objetos complexos e modelos; o Auxiliar de marca de entrada não o faz. O Auxiliar de marca de entrada, `Html.EditorFor` e `Html.TextBoxFor` são fortemente tipados (eles usam expressões lambda); `Html.TextBox` e `Html.Editor` não usam (eles usam nomes de expressão).

## HtmlAttributes

`@Html.Editor()` e `@Html.EditorFor()` usam uma entrada `ViewDataDictionary` especial chamada `htmlAttributes` ao executar seus modelos padrão. Esse comportamento pode ser aumentado usando parâmetros `additional ViewData`. A chave "htmlAttributes" diferencia maiúsculas de minúsculas. A chave "htmlAttributes" é tratada de forma semelhante ao objeto `htmlAttributes` passado para auxiliares de entrada como `@Html.TextBox()`.

```
@Html.EditorFor(model => model.YourProperty,  
new { htmlAttributes = new { @class="myCssClass", style="Width:100px" } })
```

## Nomes de expressão

O valor do atributo `asp-for` é um `ModelExpression` e o lado direito de uma expressão lambda. Portanto, `asp-for="Property1"` se torna `m => m.Property1` no código gerado e é por isso você não precisa colocar o prefixo `Model`. Você pode usar o caractere "@" para iniciar uma expressão embutida e mover para antes de `m`:

```
@{  
    var joe = "Joe";  
}  
<input asp-for="@joe" />
```

Gera o seguinte:

```
<input type="text" id="joe" name="joe" value="Joe" />
```

Com propriedades de coleção, `asp-for="CollectionProperty[23].Member"` gera o mesmo nome que `asp-for="CollectionProperty[i].Member"` quando `i` tem o valor `23`.

Quando o ASP.NET Core MVC calcula o valor de `ModelExpression`, ele inspeciona várias fontes, inclusive o `ModelState`. Considere o `<input type="text" asp-for="@Name" />`. O atributo `value` calculado é o primeiro valor não nulo:

- Da entrada de `ModelState` com a chave "Name".
- Do resultado da expressão `Model.Name`.

## Navegando para propriedades filho

Você também pode navegar para propriedades filho usando o caminho da propriedade do modelo de exibição. Considere uma classe de modelo mais complexa que contém uma propriedade `Address` filho.

```
public class AddressViewModel  
{  
    public string AddressLine1 { get; set; }  
}
```

```
public class RegisterAddressViewModel  
{  
    public string Email { get; set; }  
  
    [DataType(DataType.Password)]  
    public string Password { get; set; }  
  
    public AddressViewModel Address { get; set; }  
}
```

Na exibição, associamos a `Address.AddressLine1`:

```
@model RegisterAddressViewModel

<form asp-controller="Demo" asp-action="RegisterAddress" method="post">
    Email: <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    Address: <input asp-for="Address.AddressLine1" /><br />
    <button type="submit">Register</button>
</form>
```

O HTML a seguir é gerado para `Address.AddressLine1`:

```
<input type="text" id="Address_AddressLine1" name="Address.AddressLine1" value="" />
```

## Nomes de expressão e coleções

Exemplo, um modelo que contém uma matriz de `Colors`:

```
public class Person
{
    public List<string> Colors { get; set; }

    public int Age { get; set; }
}
```

O método de ação:

```
public IActionResult Edit(int id, int colorIndex)
{
    ViewData["Index"] = colorIndex;
    return View(GetPerson(id));
}
```

O Razor a seguir mostra como você acessa um elemento `Color` específico:

```
@model Person
 @{
     var index = (int)ViewData["index"];
 }

<form asp-controller="ToDo" asp-action="Edit" method="post">
    @Html.EditorFor(m => m.Colors[index])
    <label asp-for="Age"></label>
    <input asp-for="Age" /><br />
    <button type="submit">Post</button>
</form>
```

O modelo `Views/Shared/EditorTemplates/String.cshtml`:

```
@model string

<label asp-for="@Model"></label>
<input asp-for="@Model" /> <br />
```

Exemplo usando `List<T>`:

```

public class ToDoItem
{
    public string Name { get; set; }

    public bool IsDone { get; set; }
}

```

O Razor a seguir mostra como iterar em uma coleção:

```

@model List<ToDoItem>

<form asp-controller="ToDo" asp-action="Edit" method="post">
    <table>
        <tr> <th>Name</th> <th>Is Done</th> </tr>

        @for (int i = 0; i < Model.Count; i++)
        {
            <tr>
                @Html.EditorFor(model => model[i])
            </tr>
        }

    </table>
    <button type="submit">Save</button>
</form>

```

O modelo *Views/Shared/EditorTemplates/ToDoItem.cshtml*:

```

@model ToDoItem

<td>
    <label asp-for="@Model.Name"></label>
    @Html.DisplayFor(model => model.Name)
</td>
<td>
    <input asp-for="@Model.IsDone" />
</td>

/*
This template replaces the following Razor which evaluates the indexer three times.
<td>
    <label asp-for="@Model[i].Name"></label>
    @Html.DisplayFor(model => model[i].Name)
</td>
<td>
    <input asp-for="@Model[i].IsDone" />
</td>
*/

```

`foreach` deve ser usado, se possível, quando o valor está prestes a ser usado em um contexto equivalente `asp-for` ou `Html.DisplayFor`. Em geral, `for` é melhor do que `foreach` (se o cenário permitir) porque não é necessário alocar um enumerador; no entanto, avaliar um indexador em uma expressão LINQ pode ser caro, o que deve ser minimizado.

#### NOTE

O código de exemplo comentado acima mostra como você substituiria a expressão lambda pelo operador `@` para acessar cada `ToDoItem` na lista.

## Auxiliar de marca de área de texto

O auxiliar de marca `Textarea Tag Helper` é semelhante ao Auxiliar de marca de entrada.

- Gera os atributos `id` e `name`, bem como os atributos de validação de dados do modelo para um elemento `<textarea>`.
- Fornece tipagem forte.
- Alternativa de Auxiliar HTML: `Html.TextAreaFor`

Amostra:

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class DescriptionViewModel
    {
        [MinLength(5)]
        [MaxLength(1024)]
        public string Description { get; set; }
    }
}
```

```
@model DescriptionViewModel

<form asp-controller="Demo" asp-action="RegisterTextArea" method="post">
    <textarea asp-for="Description"></textarea>
    <button type="submit">Test</button>
</form>
```

O HTML a seguir é gerado:

```
<form method="post" action="/Demo/RegisterTextArea">
    <textarea data-val="true"
        data-val-maxlength="The field Description must be a string or array type with a maximum length of
        1024."
        data-val-maxlength-max="1024"
        data-val-minlength="The field Description must be a string or array type with a minimum length of
        5."
        data-val-minlength-min="5"
        id="Description" name="Description">
    </textarea>
    <button type="submit">Test</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## O auxiliar de marca de rótulo

- Gera a legenda do rótulo e o atributo `for` em um elemento para um nome de expressão
- Alternativa de Auxiliar HTML: `Html.LabelFor`

O **Label Tag Helper** fornece os seguintes benefícios em comparação com um elemento de rótulo HTML puro:

- Você obtém automaticamente o valor do rótulo descritivo do atributo `Display`. O nome de exibição desejado pode mudar com o tempo e a combinação do atributo `Display` e do Auxiliar de Marca de Rótulo aplicará `Display` em qualquer lugar em que for usado.
- Menos marcação no código-fonte
- Tipagem forte com a propriedade de modelo.

Amostra:

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class SimpleViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }
    }
}
```

```
@model SimpleViewModel

<form asp-controller="Demo" asp-action="RegisterLabel" method="post">
    <label asp-for="Email"></label>
    <input asp-for="Email" /> <br />
</form>
```

O HTML a seguir é gerado para o elemento `<label>`:

```
<label for="Email">Email Address</label>
```

O Auxiliar de marca de rótulo gerou o valor do atributo `for` de "Email", que é a ID associada ao elemento `<input>`. Auxiliares de marca geram elementos `id` e `for` consistentes para que eles possam ser associados corretamente. A legenda neste exemplo é proveniente do atributo `Display`. Se o modelo não contivesse um atributo `Display`, a legenda seria o nome da propriedade da expressão.

## Os auxiliares de marca de validação

Há dois auxiliares de marca de validação. O **Validation Message Tag Helper** (que exibe uma mensagem de validação para uma única propriedade em seu modelo) e o **Validation Summary Tag Helper** (que exibe um resumo dos erros de validação). O **Input Tag Helper** adiciona atributos de validação do lado do cliente HTML5 para elementos de entrada baseados em atributos de anotação de dados em suas classes de modelo. A validação também é executada no servidor. O Auxiliar de marca de validação exibe essas mensagens de erro quando ocorre um erro de validação.

### O Auxiliar de marca de mensagem de validação

- Adiciona o atributo `data-valmsg-for="property"` [HTML5](#) ao elemento `span`, que anexa as mensagens de erro de validação no campo de entrada da propriedade do modelo especificado. Quando ocorre um erro de validação do lado do cliente, [jQuery](#) exibe a mensagem de erro no elemento `<span>`.

- A validação também é feita no servidor. Os clientes poderão ter o JavaScript desabilitado e parte da validação só pode ser feita no lado do servidor.

- Alternativa de Auxiliar HTML: `@Html.ValidationMessageFor`

O `Validation Message Tag Helper` é usado com o atributo `asp-validation-for` em um elemento HTML `span`.

```
<span asp-validation-for="Email"></span>
```

O Auxiliar de marca de mensagem de validação gerará o HTML a seguir:

```
<span class="field-validation-valid"
      data-valmsg-for="Email"
      data-valmsg-replace="true"></span>
```

Geralmente, você usa o `Validation Message Tag Helper` após um Auxiliar de marca `Input` para a mesma propriedade. Fazer isso exibe as mensagens de erro de validação próximo à entrada que causou o erro.

#### NOTE

É necessário ter uma exibição com as referências de script `jQuery` e JavaScript corretas em vigor para a validação do lado do cliente. Consulte [Validação de Modelo](#) para obter mais informações.

Quando ocorre um erro de validação do lado do servidor (por exemplo, quando você tem validação do lado do servidor personalizada ou a validação do lado do cliente está desabilitada), o MVC coloca essa mensagem de erro como o corpo do elemento `<span>`.

```
<span class="field-validation-error" data-valmsg-for="Email"
      data-valmsg-replace="true">
    The Email Address field is required.
</span>
```

#### Auxiliar de marca de resumo de validação

- Tem como alvo elementos `<div>` com o atributo `asp-validation-summary`
- Alternativa de Auxiliar HTML: `@Html.ValidationSummary`

O `Validation Summary Tag Helper` é usado para exibir um resumo das mensagens de validação. O valor do atributo `asp-validation-summary` pode ser qualquer um dos seguintes:

ASP-VALIDATION-SUMMARY	MENSAGENS DE VALIDAÇÃO EXIBIDAS
ValidationSummary.All	Nível da propriedade e do modelo
ValidationSummary.ModelOnly	Modelo
ValidationSummary.None	Nenhum

#### Amostra

No exemplo a seguir, o modelo de dados é decorado com atributos `DataAnnotation`, o que gera mensagens de erro de validação no elemento `<input>`. Quando ocorre um erro de validação, o Auxiliar de marca de validação exibe a mensagem de erro:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}

```

```

@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterValidation" method="post">
    <div asp-validation-summary="ModelOnly"></div>
    Email: <input asp-for="Email" /> <br />
    <span asp-validation-for="Email"></span><br />
    Password: <input asp-for="Password" /><br />
    <span asp-validation-for="Password"></span><br />
    <button type="submit">Register</button>
</form>

```

O código HTML gerado (quando o modelo é válido):

```

<form action="/DemoReg/Register" method="post">
    <div class="validation-summary-valid" data-valmsg-summary="true">
        <ul><li style="display:none"></li></ul></div>
    Email: <input name="Email" id="Email" type="email" value="" data-val-required="The Email field is required." data-val-email="The Email field is not a valid email address." data-val="true"> <br />
    <span class="field-validation-valid" data-valmsg-replace="true" data-valmsg-for="Email"></span><br />
    Password: <input name="Password" id="Password" type="password" data-val-required="The Password field is required." data-val="true"><br />
    <span class="field-validation-valid" data-valmsg-replace="true" data-valmsg-for="Password"></span><br />
    <button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## Auxiliar de Marca de Seleção

- Gera `select` e os elementos `option` associados para as propriedades do modelo.
- Tem uma alternativa de Auxiliar HTML `Html.DropDownListFor` e `Html.ListBoxFor`

O `Select Tag Helper` `asp-for` especifica o nome da propriedade do modelo para o elemento `select` e `asp-items` especifica os elementos `option`. Por exemplo:

```

<select asp-for="Country" asp-items="Model.Countries"></select>

```

Amostra:

```

using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace FormsTagHelper.ViewModels
{
    public class CountryViewModel
    {
        public string Country { get; set; }

        public List<SelectListItem> Countries { get; } = new List<SelectListItem>
        {
            new SelectListItem { Value = "MX", Text = "Mexico" },
            new SelectListItem { Value = "CA", Text = "Canada" },
            new SelectListItem { Value = "US", Text = "USA" },
        };
    }
}

```

O método `Index` inicializa o `CountryViewModel`, define o país selecionado e o transmite para a exibição `Index`.

```

public IActionResult Index()
{
    var model = new CountryViewModel();
    model.Country = "CA";
    return View(model);
}

```

O método `Index` HTTP POST exibe a seleção:

```

[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Index(CountryViewModel model)
{
    if (ModelState.IsValid)
    {
        var msg = model.Country + " selected";
        return RedirectToAction("IndexSuccess", new { message = msg });
    }

    // If we got this far, something failed; redisplay form.
    return View(model);
}

```

A exibição `Index`:

```

@model CountryViewModel

<form asp-controller="Home" asp-action="Index" method="post">
    <select asp-for="Country" asp-items="Model.Countries"></select>
    <br /><button type="submit">Register</button>
</form>

```

Que gera o seguinte HTML (com "CA" selecionado):

```

<form method="post" action="/">
    <select id="Country" name="Country">
        <option value="MX">Mexico</option>
        <option selected="selected" value="CA">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## NOTE

Não é recomendável usar `ViewBag` ou `ViewData` com o Auxiliar de Marca de Seleção. Um modelo de exibição é mais robusto para fornecer metadados MVC e, geralmente, menos problemático.

O valor do atributo `asp-for` é um caso especial e não requer um prefixo `Model`, os outros atributos do Auxiliar de marca requerem (como `asp-items`)

```
<select asp-for="Country" asp-items="Model.Countries"></select>
```

## Associação de enumeração

Geralmente, é conveniente usar `<select>` com uma propriedade `enum` e gerar os elementos `SelectListItem` dos valores `enum`.

Amostra:

```

public class CountryEnumViewModel
{
    public CountryEnum EnumCountry { get; set; }
}

```

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public enum CountryEnum
    {
        [Display(Name = "United Mexican States")]
        Mexico,
        [Display(Name = "United States of America")]
        USA,
        Canada,
        France,
        Germany,
        Spain
    }
}

```

O método `GetEnumSelectList` gera um objeto `selectList` para uma enumeração.

```

@model CountryEnumViewModel

<form asp-controller="Home" asp-action="IndexEnum" method="post">
    <select asp-for="EnumCountry"
        asp-items="Html.GetEnumSelectList<CountryEnum>()">
    </select>
    <br /><button type="submit">Register</button>
</form>

```

É possível decorar sua lista de enumeradores com o atributo `Display` para obter uma interface do usuário mais rica:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public enum CountryEnum
    {
        [Display(Name = "United Mexican States")]
        Mexico,
        [Display(Name = "United States of America")]
        USA,
        Canada,
        France,
        Germany,
        Spain
    }
}

```

O HTML a seguir é gerado:

```

<form method="post" action="/Home/IndexEnum">
    <select data-val="true" data-val-required="The EnumCountry field is required." id="EnumCountry" name="EnumCountry">
        <option value="0">United Mexican States</option>
        <option value="1">United States of America</option>
        <option value="2">Canada</option>
        <option value="3">France</option>
        <option value="4">Germany</option>
        <option selected="selected" value="5">Spain</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## Grupo de opções

O elemento HTML `<optgroup>` é gerado quando o modelo de exibição contém um ou mais objetos `SelectListGroup`.

O `CountryViewModelGroup` agrupa os elementos `SelectListItem` nos grupos "América do Norte" e "Europa":

```

public class CountryViewModelGroup
{
    public CountryViewModelGroup()
    {
        var NorthAmericaGroup = new SelectListGroup { Name = "North America" };
        var EuropeGroup = new SelectListGroup { Name = "Europe" };

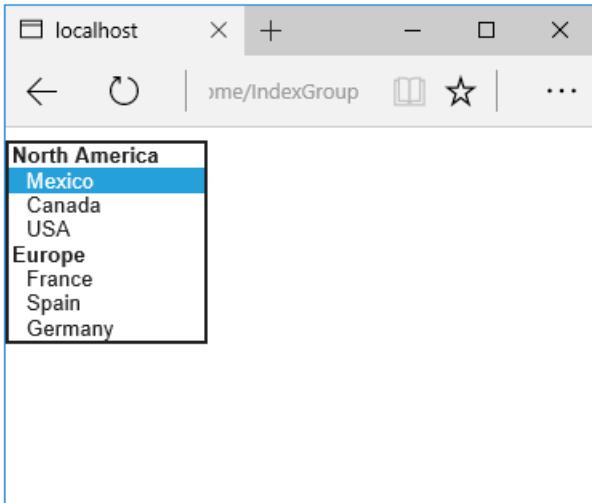
        Countries = new List<SelectListItem>
        {
            new SelectListItem
            {
                Value = "MEX",
                Text = "Mexico",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "CAN",
                Text = "Canada",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "US",
                Text = "USA",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "FR",
                Text = "France",
                Group = EuropeGroup
            },
            new SelectListItem
            {
                Value = "ES",
                Text = "Spain",
                Group = EuropeGroup
            },
            new SelectListItem
            {
                Value = "DE",
                Text = "Germany",
                Group = EuropeGroup
            }
        };
    }

    public string Country { get; set; }

    public List<SelectListItem> Countries { get; }
}

```

Os dois grupos são mostrados abaixo:



O HTML gerado:

```
<form method="post" action="/Home/IndexGroup">
    <select id="Country" name="Country">
        <optgroup label="North America">
            <option value="MEX">Mexico</option>
            <option value="CAN">Canada</option>
            <option value="US">USA</option>
        </optgroup>
        <optgroup label="Europe">
            <option value="FR">France</option>
            <option value="ES">Spain</option>
            <option value="DE">Germany</option>
        </optgroup>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="removed for brevity" />
</form>
```

## Seleção múltipla

O Auxiliar de Marca de Seleção gerará automaticamente o atributo `multiple = "multiple"` se a propriedade especificada no atributo `asp-for` for um `IEnumerable`. Por exemplo, considerando o seguinte modelo:

```
using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace FormsTagHelper.ViewModels
{
    public class CountryViewModel : IEnumerable
    {
        public IEnumerable<string> CountryCodes { get; set; }

        public List<SelectListItem> Countries { get; } = new List<SelectListItem>
        {
            new SelectListItem { Value = "MX", Text = "Mexico" },
            new SelectListItem { Value = "CA", Text = "Canada" },
            new SelectListItem { Value = "US", Text = "USA" },
            new SelectListItem { Value = "FR", Text = "France" },
            new SelectListItem { Value = "ES", Text = "Spain" },
            new SelectListItem { Value = "DE", Text = "Germany" }
        };
    }
}
```

Com a seguinte exibição:

```
@model CountryViewModelIEnumarable

<form asp-controller="Home" asp-action="IndexMultiSelect" method="post">
    <select asp-for="CountryCodes" asp-items="Model.Countries"></select>
    <br /><button type="submit">Register</button>
</form>
```

Gera o seguinte HTML:

```
<form method="post" action="/Home/IndexMultiSelect">
    <select id="CountryCodes"
        multiple="multiple"
        name="CountryCodes"><option value="MX">Mexico</option>
    <option value="CA">Canada</option>
    <option value="US">USA</option>
    <option value="FR">France</option>
    <option value="ES">Spain</option>
    <option value="DE">Germany</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## Nenhuma seleção

Se acabar usando a opção "não especificado" em várias páginas, você poderá criar um modelo para eliminar o HTML de repetição:

```
@model CountryViewModel

<form asp-controller="Home" asp-action="IndexEmpty" method="post">
    @Html.EditorForModel()
    <br /><button type="submit">Register</button>
</form>
```

O modelo *Views/Shared/EditorTemplates/CountryViewModel.cshtml*:

```
@model CountryViewModel

<select asp-for="Country" asp-items="Model.Countries">
    <option value="">--none--</option>
</select>
```

O acréscimo de elementos HTML `<option>` não está limitado ao caso de *Nenhuma seleção*. Por exemplo, o seguinte método de ação e exibição gerarão HTML semelhante ao código acima:

```
public IActionResult IndexOption(int id)
{
    var model = new CountryViewModel();
    model.Country = "CA";
    return View(model);
}
```

```
@model CountryViewModel

<form asp-controller="Home" asp-action="IndexEmpty" method="post">
    <select asp-for="Country">
        <option value="">&lt;none&gt;</option>
        <option value="MX">Mexico</option>
        <option value="CA">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
</form>
```

O elemento `<option>` correto será selecionado (contém o atributo `selected="selected"`) dependendo do valor atual de `Country`.

```
<form method="post" action="/Home/IndexEmpty">
    <select id="Country" name="Country">
        <option value="">&lt;none&gt;</option>
        <option value="MX">Mexico</option>
        <option value="CA" selected="selected">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## Recursos adicionais

- [Auxiliares de Marca no ASP.NET Core](#)
- [Elemento de formulário HTML](#)
- [Token de verificação de solicitação](#)
- [Model binding no ASP.NET Core](#)
- [Validação de modelo no ASP.NET Core MVC](#)
- [Interface IAttributeAdapter](#)
- [Snippets de código para este documento](#)

# Auxiliar de marca parcial no ASP.NET Core

04/02/2019 • 5 minutes to read • [Edit Online](#)

Por [Scott Addie](#)

Para obter uma visão geral dos Auxiliares de Marca, confira [Auxiliares de Marca no ASP.NET Core](#).

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Visão geral

O auxiliar de marca parcial é usado para renderizar uma [exibição parcial](#) em Páginas Razor e em aplicativos MVC. Considere que ele:

- Exige o ASP.NET Core 2.1 ou posterior.
- É uma alternativa à [sintaxe do HTML Helper](#).
- Renderiza a exibição parcial de forma assíncrona.

As opções do HTML Helper para renderizar uma exibição parcial incluem:

- `@await Html.PartialAsync`
- `@await Html.RenderPartialAsync`
- `@Html.Partial`
- `@Html.RenderPartial`

O modelo *Product* é usado nos exemplos ao longo deste documento:

```
namespace TagHelpersBuiltIn.Models
{
    public class Product
    {
        public int Number { get; set; }

        public string Name { get; set; }

        public string Description { get; set; }
    }
}
```

Segue um inventário dos atributos do auxiliar de marca parcial.

## name

O atributo `name` é necessário. Indica o nome ou o caminho da exibição parcial a ser renderizada. Quando é fornecido um nome de exibição parcial, o processo [descoberta de exibição](#) é iniciado. Esse processo é ignorado quando um caminho explícito é fornecido. Para todos os valores de `name` aceitáveis, consulte [Descoberta de exibição parcial](#).

A marcação a seguir usa um caminho explícito, indicando que `_ProductPartial.cshtml` deve ser carregado da pasta `Compartilhado`. Usando o atributo `for`, um modelo é passado para a exibição parcial para associação.

```
<partial name="Shared/_ProductPartial.cshtml"
       for="Product" />
```

## for

O atributo `for` atribui uma [ModelExpression](#) a ser avaliada em relação ao modelo atual. Uma `ModelExpression` infere a sintaxe `@Model.`. Por exemplo, `for="Product"` pode ser usado em vez de `for="@Model.Product"`. Esse comportamento de inferência padrão é substituído usando o símbolo `@` para definir uma expressão embutida.

A seguinte marcação carrega `_ProductPartial.cshtml`:

```
<partial name="_ProductPartial"
       for="Product" />
```

A exibição parcial é associada à propriedade `Product` do modelo de página associado:

```
using Microsoft.AspNetCore.Mvc.RazorPages;
using TagHelpersBuiltIn.Models;

namespace TagHelpersBuiltIn.Pages
{
    public class ProductModel : PageModel
    {
        public Product Product { get; set; }

        public void OnGet()
        {
            Product = new Product
            {
                Number = 1,
                Name = "Test product",
                Description = "This is a test product"
            };
        }
    }
}
```

## modelo

O atributo `model` atribui uma instância de modelo a ser passada para a exibição parcial. O atributo `model` não pode ser usado com o atributo `for`.

Na marcação a seguir, é criada uma nova instância do objeto `Product` que é passada ao atributo `model` para associação:

```
<partial name="_ProductPartial"
       model='new Product { Number = 1, Name = "Test product", Description = "This is a test" }' />
```

## view-data

O atributo `view-data` atribui um [ViewDataDictionary](#) a ser passado para a exibição parcial. A seguinte marcação faz com que toda a coleção ViewData fique acessível para a exibição parcial:

```

@{
    ViewData["IsNumberReadOnly"] = true;
}

<partial name="_ProductViewDataPartial"
    for="Product"
    view-data="ViewData" />

```

No código anterior, o valor da chave `IsNumberReadOnly` é definido como `true` e adicionado à coleção `ViewData`. Consequentemente, `ViewData["IsNumberReadOnly"]` se torna acessível dentro da exibição parcial a seguir:

```

@model TagHelpersBuiltIn.Models.Product

<div class="form-group">
    <label asp-for="Number"></label>
    @if ((bool)ViewData["IsNumberReadOnly"])
    {
        <input asp-for="Number" type="number" class="form-control" readonly />
    }
    else
    {
        <input asp-for="Number" type="number" class="form-control" />
    }
</div>
<div class="form-group">
    <label asp-for="Name"></label>
    <input asp-for="Name" type="text" class="form-control" />
</div>
<div class="form-group">
    <label asp-for="Description"></label>
    <textarea asp-for="Description" rows="4" cols="50" class="form-control"></textarea>
</div>

```

Neste exemplo, o valor de `ViewData["IsNumberReadOnly"]` determina se o campo *Número* é exibido como somente leitura.

## Migrar de um auxiliar HTML

Considere o seguinte exemplo de auxiliar HTML assíncrono. Uma coleção de produtos é iterada e exibida. Segundo o primeiro parâmetro do método `PartialAsync`, a exibição parcial `_ProductPartial.cshtml` é carregada. Uma instância do modelo `Product` é passada para a exibição parcial para associação.

```

@foreach (var product in Model.Products)
{
    @await Html.PartialAsync("_ProductPartial", product)
}

```

O auxiliar de marca parcial segue alcançando o mesmo comportamento de renderização assíncrona que o auxiliar HTML `PartialAsync`. Uma instância de modelo `Product` é atribuída ao atributo `model` para associação à exibição parcial.

```

@foreach (var product in Model.Products)
{
    <partial name="_ProductPartial" model="product" />
}

```

## Recursos adicionais

- Exibições parciais no ASP.NET Core
- Exibições no ASP.NET Core MVC

# Auxiliares de marca em formulários no ASP.NET Core

14/01/2019 • 28 minutes to read • [Edit Online](#)

Por [Rick Anderson](#), [Dave Paquette](#) e [Jerrie Pelser](#)

Este documento demonstra como é o trabalho com Formulários e os elementos HTML usados comumente em um Formulário. O elemento HTML [Formulário](#) fornece o mecanismo primário que os aplicativos Web usam para postar dados para o servidor. A maior parte deste documento descreve os [Auxiliares de marca](#) e como eles podem ajudar você a criar formulários HTML robustos de forma produtiva. É recomendável que você leia [Introdução ao auxiliares de marca](#) antes de ler este documento.

Em muitos casos, os Auxiliares HTML fornecem uma abordagem alternativa a um Auxiliar de Marca específico, mas é importante reconhecer que os Auxiliares de Marca não substituem os Auxiliares HTML e que não há um Auxiliar de Marca para cada Auxiliar HTML. Quando existe um Auxiliar HTML alternativo, ele é mencionado.

## O Auxiliar de marca de formulário

O Auxiliar de marca de [formulário](#):

- Gera o valor do atributo HTML `<FORM>` `action` para uma ação do controlador MVC ou uma rota nomeada
- Gera um [Token de verificação de solicitação](#) oculto para evitar a falsificação de solicitações entre sites (quando usado com o atributo `[ValidateAntiForgeryToken]` no método de ação HTTP Post)
- Fornece o atributo `asp-route-<Parameter Name>`, em que `<Parameter Name>` é adicionado aos valores de rota. Os parâmetros `routeValues` para `Html.BeginForm` e `Html.BeginRouteForm` fornecem funcionalidade semelhante.
- Tem uma alternativa de Auxiliar HTML `Html.BeginForm` e `Html.BeginRouteForm`

Amostra:

```
<form asp-controller="Demo" asp-action="Register" method="post">
    <!-- Input and Submit elements -->
</form>
```

O Auxiliar de marca de formulário acima gera o HTML a seguir:

```
<form method="post" action="/Demo/Register">
    <!-- Input and Submit elements -->
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

O tempo de execução do MVC gera o valor do atributo `action` dos atributos `asp-controller` e `asp-action` do Auxiliar de marca de formulário. O Auxiliar de marca de formulário também gera um [Token de verificação de solicitação](#) oculto para evitar a falsificação de solicitações entre sites (quando usado com o atributo `[ValidateAntiForgeryToken]` no método de ação HTTP Post). É difícil proteger um Formulário HTML puro contra falsificação de solicitações entre sites e o Auxiliar de marca de formulário fornece este serviço para você.

### Usando uma rota nomeada

O atributo do Auxiliar de Marca `asp-route` também pode gerar a marcação para o atributo HTML `action`. Um

aplicativo com uma [rota](#) chamada `register` poderia usar a seguinte marcação para a página de registro:

```
<form asp-route="register" method="post">
    <!-- Input and Submit elements -->
</form>
```

Muitas das exibições na pasta *Modos de Exibição/Conta* (gerada quando você cria um novo aplicativo Web com *Contas de usuário individuais*) contêm o atributo [asp-route-returnurl](#):

```
<form asp-controller="Account" asp-action="Login"
      asp-route-returnurl="@ViewData["ReturnUrl"]"
      method="post" class="form-horizontal" role="form">
```

#### NOTE

Com os modelos internos, `returnUrl` só é preenchido automaticamente quando você tenta acessar um recurso autorizado, mas não está autenticado ou autorizado. Quando você tenta fazer um acesso não autorizado, o middleware de segurança o redireciona para a página de logon com o `returnUrl` definido.

## O auxiliar de marca de entrada

O Auxiliar de marca de entrada associa um elemento HTML `<input>` a uma expressão de modelo em sua exibição do Razor.

Sintaxe:

```
<input asp-for="<Expression Name>" />
```

O auxiliar de marca de entrada:

- Gera os atributos HTML `id` e `name` para o nome da expressão especificada no atributo `asp-for`.  
`asp-for="Property1.Property2"` equivale a `m => m.Property1.Property2`. O nome da expressão é o que é usado para o valor do atributo `asp-for`. Consulte a seção [Nomes de expressão](#) para obter informações adicionais.
- Define o valor do atributo HTML `type` com base nos atributos de tipo de modelo e [anotação de dados](#) aplicados à propriedade de modelo
- O valor do atributo HTML `type` não será substituído quando um for especificado
- Gera atributos de validação [HTML5](#) de atributos de [anotação de dados](#) aplicados a propriedades de modelo
- Tem uma sobreposição de recursos de Auxiliar HTML com `Html.TextBoxFor` e `Html.EditorFor`. Consulte a seção **Alternativas de Auxiliar HTML ao Auxiliar de marca de entrada** para obter detalhes.
- Fornece tipagem forte. Se o nome da propriedade for alterado e você não atualizar o Auxiliar de marca, você verá um erro semelhante ao seguinte:

An error occurred during the compilation of a resource required to process this request. Please review the following specific error details and modify your source code appropriately.

Type expected

'RegisterViewModel' does not contain a definition for 'Email' and no extension method 'Email' accepting a first argument of type 'RegisterViewModel' could be found (are you missing a using directive or an assembly reference?)

O Auxiliar de marca `Input` define o atributo HTML `type` com base no tipo .NET. A tabela a seguir lista alguns tipos .NET comuns e o tipo HTML gerado (não estão listados todos os tipos .NET).

TIPO .NET	TIPO DE ENTRADA
Bool	<code>type="checkbox"</code>
Cadeia de Caracteres	<code>type="text"</code>
DateTime	<code>type="datetime-local"</code>
Byte	<code>type="number"</code>
int	<code>type="number"</code>
Single e Double	<code>type="number"</code>

A tabela a seguir mostra alguns atributos de [anotações de dados](#) comuns que o auxiliar de marca de entrada mapeará para tipos de entrada específicos (não são listados todos os atributos de validação):

ATRIBUTO	TIPO DE ENTRADA
<code>[EmailAddress]</code>	<code>type="email"</code>
<code>[Url]</code>	<code>type="url"</code>
<code>[HiddenInput]</code>	<code>type="hidden"</code>
<code>[Phone]</code>	<code>type="tel"</code>
<code>[DataType(DataType.Password)]</code>	<code>type="password"</code>
<code>[DataType(DataType.Date)]</code>	<code>type="date"</code>
<code>[DataType(DataType.Time)]</code>	<code>type="time"</code>

Amostra:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}

```

```

@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterInput" method="post">
    Email: <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    <button type="submit">Register</button>
</form>

```

O código acima gera o seguinte HTML:

```

<form method="post" action="/Demo/RegisterInput">
    Email:
    <input type="email" data-val="true"
           data-val-email="The Email Address field is not a valid email address."
           data-val-required="The Email Address field is required."
           id="Email" name="Email" value="" /> <br>
    Password:
    <input type="password" data-val="true"
           data-val-required="The Password field is required."
           id="Password" name="Password" /><br>
    <button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

As anotações de dados aplicadas às propriedades `Email` e `Password` geram metadados no modelo. O Auxiliar de marca de entrada consome os metadados do modelo e produz atributos HTML5 `data-val-*` (consulte [Validação de modelo](#)). Esses atributos descrevem os validadores a serem anexados aos campos de entrada. Isso fornece validação de [jQuery](#) e HTML5 discreto. Os atributos discretos têm o formato `data-val-rule="Error Message"`, em que a regra é o nome da regra de validação (como `data-val-required`, `data-val-email`, `data-val-maxlength` etc.). Se uma mensagem de erro for fornecida no atributo, ela será exibida como o valor para o atributo `data-val-rule`. Também há atributos do formulário `data-val-ruleName-argumentName="argumentValue"` que fornecem detalhes adicionais sobre a regra, por exemplo, `data-val-maxlength-max="1024"`.

## Alternativas de Auxiliar HTML ao Auxiliar de marca de entrada

`Html.TextBox`, `Html.TextBoxFor`, `Html.Editor` e `Html.EditorFor` têm recursos que se sobrepõem aos do Auxiliar de marca de entrada. O Auxiliar de marca de entrada define automaticamente o atributo `type`; `Html.TextBox` e `Html.TextBoxFor` não o fazem. `Html.Editor` e `Html.EditorFor` manipulam coleções, objetos complexos e modelos; o Auxiliar de marca de entrada não o faz. O Auxiliar de marca de entrada, `Html.EditorFor` e `Html.TextBoxFor` são fortemente tipados (eles usam expressões lambda); `Html.TextBox` e `Html.Editor` não usam (eles usam nomes de expressão).

## HtmlAttributes

`@Html.Editor()` e `@Html.EditorFor()` usam uma entrada `ViewDataDictionary` especial chamada `htmlAttributes` ao executar seus modelos padrão. Esse comportamento pode ser aumentado usando parâmetros `additional ViewData`. A chave "htmlAttributes" diferencia maiúsculas de minúsculas. A chave "htmlAttributes" é tratada de forma semelhante ao objeto `htmlAttributes` passado para auxiliares de entrada como `@Html.TextBox()`.

```
@Html.EditorFor(model => model.YourProperty,  
new { htmlAttributes = new { @class="myCssClass", style="Width:100px" } })
```

## Nomes de expressão

O valor do atributo `asp-for` é um `ModelExpression` e o lado direito de uma expressão lambda. Portanto, `asp-for="Property1"` se torna `m => m.Property1` no código gerado e é por isso você não precisa colocar o prefixo `Model`. Você pode usar o caractere "@" para iniciar uma expressão embutida e mover para antes de `m`:

```
@{  
    var joe = "Joe";  
}  
<input asp-for="@joe" />
```

Gera o seguinte:

```
<input type="text" id="joe" name="joe" value="Joe" />
```

Com propriedades de coleção, `asp-for="CollectionProperty[23].Member"` gera o mesmo nome que `asp-for="CollectionProperty[i].Member"` quando `i` tem o valor `23`.

Quando o ASP.NET Core MVC calcula o valor de `ModelExpression`, ele inspeciona várias fontes, inclusive o `ModelState`. Considere o `<input type="text" asp-for="@Name" />`. O atributo `value` calculado é o primeiro valor não nulo:

- Da entrada de `ModelState` com a chave "Name".
- Do resultado da expressão `Model.Name`.

## Navegando para propriedades filho

Você também pode navegar para propriedades filho usando o caminho da propriedade do modelo de exibição. Considere uma classe de modelo mais complexa que contém uma propriedade `Address` filho.

```
public class AddressViewModel  
{  
    public string AddressLine1 { get; set; }  
}
```

```
public class RegisterAddressViewModel  
{  
    public string Email { get; set; }  
  
    [DataType(DataType.Password)]  
    public string Password { get; set; }  
  
    public AddressViewModel Address { get; set; }  
}
```

Na exibição, associamos a `Address.AddressLine1`:

```
@model RegisterAddressViewModel

<form asp-controller="Demo" asp-action="RegisterAddress" method="post">
    Email: <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    Address: <input asp-for="Address.AddressLine1" /><br />
    <button type="submit">Register</button>
</form>
```

O HTML a seguir é gerado para `Address.AddressLine1`:

```
<input type="text" id="Address_AddressLine1" name="Address.AddressLine1" value="" />
```

## Nomes de expressão e coleções

Exemplo, um modelo que contém uma matriz de `Colors`:

```
public class Person
{
    public List<string> Colors { get; set; }

    public int Age { get; set; }
}
```

O método de ação:

```
public IActionResult Edit(int id, int colorIndex)
{
    ViewData["Index"] = colorIndex;
    return View(GetPerson(id));
}
```

O Razor a seguir mostra como você acessa um elemento `Color` específico:

```
@model Person
 @{
     var index = (int)ViewData["index"];
 }

<form asp-controller="ToDo" asp-action="Edit" method="post">
    @Html.EditorFor(m => m.Colors[index])
    <label asp-for="Age"></label>
    <input asp-for="Age" /><br />
    <button type="submit">Post</button>
</form>
```

O modelo `Views/Shared/EditorTemplates/String.cshtml`:

```
@model string

<label asp-for="@Model"></label>
<input asp-for="@Model" /> <br />
```

Exemplo usando `List<T>`:

```

public class ToDoItem
{
    public string Name { get; set; }

    public bool IsDone { get; set; }
}

```

O Razor a seguir mostra como iterar em uma coleção:

```

@model List<ToDoItem>

<form asp-controller="ToDo" asp-action="Edit" method="post">
    <table>
        <tr> <th>Name</th> <th>Is Done</th> </tr>

        @for (int i = 0; i < Model.Count; i++)
        {
            <tr>
                @Html.EditorFor(model => model[i])
            </tr>
        }

    </table>
    <button type="submit">Save</button>
</form>

```

O modelo *Views/Shared/EditorTemplates/ToDoItem.cshtml*:

```

@model ToDoItem

<td>
    <label asp-for="@Model.Name"></label>
    @Html.DisplayFor(model => model.Name)
</td>
<td>
    <input asp-for="@Model.IsDone" />
</td>

/*
This template replaces the following Razor which evaluates the indexer three times.
<td>
    <label asp-for="@Model[i].Name"></label>
    @Html.DisplayFor(model => model[i].Name)
</td>
<td>
    <input asp-for="@Model[i].IsDone" />
</td>
*/

```

`foreach` deve ser usado, se possível, quando o valor está prestes a ser usado em um contexto equivalente `asp-for` ou `Html.DisplayFor`. Em geral, `for` é melhor do que `foreach` (se o cenário permitir) porque não é necessário alocar um enumerador; no entanto, avaliar um indexador em uma expressão LINQ pode ser caro, o que deve ser minimizado.

#### NOTE

O código de exemplo comentado acima mostra como você substituiria a expressão lambda pelo operador `@` para acessar cada `ToDoItem` na lista.

## Auxiliar de marca de área de texto

O auxiliar de marca `Textarea Tag Helper` é semelhante ao Auxiliar de marca de entrada.

- Gera os atributos `id` e `name`, bem como os atributos de validação de dados do modelo para um elemento `<textarea>`.
- Fornece tipagem forte.
- Alternativa de Auxiliar HTML: `Html.TextAreaFor`

Amostra:

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class DescriptionViewModel
    {
        [MinLength(5)]
        [MaxLength(1024)]
        public string Description { get; set; }
    }
}
```

```
@model DescriptionViewModel

<form asp-controller="Demo" asp-action="RegisterTextArea" method="post">
    <textarea asp-for="Description"></textarea>
    <button type="submit">Test</button>
</form>
```

O HTML a seguir é gerado:

```
<form method="post" action="/Demo/RegisterTextArea">
    <textarea data-val="true"
        data-val-maxlength="The field Description must be a string or array type with a maximum length of
        1024."
        data-val-maxlength-max="1024"
        data-val-minlength="The field Description must be a string or array type with a minimum length of
        5."
        data-val-minlength-min="5"
        id="Description" name="Description">
    </textarea>
    <button type="submit">Test</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## O auxiliar de marca de rótulo

- Gera a legenda do rótulo e o atributo `for` em um elemento para um nome de expressão
- Alternativa de Auxiliar HTML: `Html.LabelFor`

O **Label Tag Helper** fornece os seguintes benefícios em comparação com um elemento de rótulo HTML puro:

- Você obtém automaticamente o valor do rótulo descritivo do atributo `Display`. O nome de exibição desejado pode mudar com o tempo e a combinação do atributo `Display` e do Auxiliar de Marca de Rótulo aplicará `Display` em qualquer lugar em que for usado.
- Menos marcação no código-fonte
- Tipagem forte com a propriedade de modelo.

Amostra:

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class SimpleViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }
    }
}
```

```
@model SimpleViewModel

<form asp-controller="Demo" asp-action="RegisterLabel" method="post">
    <label asp-for="Email"></label>
    <input asp-for="Email" /> <br />
</form>
```

O HTML a seguir é gerado para o elemento `<label>`:

```
<label for="Email">Email Address</label>
```

O Auxiliar de marca de rótulo gerou o valor do atributo `for` de "Email", que é a ID associada ao elemento `<input>`. Auxiliares de marca geram elementos `id` e `for` consistentes para que eles possam ser associados corretamente. A legenda neste exemplo é proveniente do atributo `Display`. Se o modelo não contivesse um atributo `Display`, a legenda seria o nome da propriedade da expressão.

## Os auxiliares de marca de validação

Há dois auxiliares de marca de validação. O **Validation Message Tag Helper** (que exibe uma mensagem de validação para uma única propriedade em seu modelo) e o **Validation Summary Tag Helper** (que exibe um resumo dos erros de validação). O **Input Tag Helper** adiciona atributos de validação do lado do cliente HTML5 para elementos de entrada baseados em atributos de anotação de dados em suas classes de modelo. A validação também é executada no servidor. O Auxiliar de marca de validação exibe essas mensagens de erro quando ocorre um erro de validação.

### O Auxiliar de marca de mensagem de validação

- Adiciona o atributo `data-valmsg-for="property"` [HTML5](#) ao elemento `span`, que anexa as mensagens de erro de validação no campo de entrada da propriedade do modelo especificado. Quando ocorre um erro de validação do lado do cliente, [jQuery](#) exibe a mensagem de erro no elemento `<span>`.

- A validação também é feita no servidor. Os clientes poderão ter o JavaScript desabilitado e parte da validação só pode ser feita no lado do servidor.

- Alternativa de Auxiliar HTML: `Html.ValidationMessageFor`

O `Validation Message Tag Helper` é usado com o atributo `asp-validation-for` em um elemento HTML `span`.

```
<span asp-validation-for="Email"></span>
```

O Auxiliar de marca de mensagem de validação gerará o HTML a seguir:

```
<span class="field-validation-valid"
      data-valmsg-for="Email"
      data-valmsg-replace="true"></span>
```

Geralmente, você usa o `Validation Message Tag Helper` após um Auxiliar de marca `Input` para a mesma propriedade. Fazer isso exibe as mensagens de erro de validação próximo à entrada que causou o erro.

#### NOTE

É necessário ter uma exibição com as referências de script `jQuery` e JavaScript corretas em vigor para a validação do lado do cliente. Consulte [Validação de Modelo](#) para obter mais informações.

Quando ocorre um erro de validação do lado do servidor (por exemplo, quando você tem validação do lado do servidor personalizada ou a validação do lado do cliente está desabilitada), o MVC coloca essa mensagem de erro como o corpo do elemento `<span>`.

```
<span class="field-validation-error" data-valmsg-for="Email"
      data-valmsg-replace="true">
    The Email Address field is required.
</span>
```

#### Auxiliar de marca de resumo de validação

- Tem como alvo elementos `<div>` com o atributo `asp-validation-summary`
- Alternativa de Auxiliar HTML: `@Html.ValidationSummary`

O `Validation Summary Tag Helper` é usado para exibir um resumo das mensagens de validação. O valor do atributo `asp-validation-summary` pode ser qualquer um dos seguintes:

ASP-VALIDATION-SUMMARY	MENSAGENS DE VALIDAÇÃO EXIBIDAS
ValidationSummary.All	Nível da propriedade e do modelo
ValidationSummary.ModelOnly	Modelo
ValidationSummary.None	Nenhum

#### Amostra

No exemplo a seguir, o modelo de dados é decorado com atributos `DataAnnotation`, o que gera mensagens de erro de validação no elemento `<input>`. Quando ocorre um erro de validação, o Auxiliar de marca de validação exibe a mensagem de erro:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}

```

```

@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterValidation" method="post">
    <div asp-validation-summary="ModelOnly"></div>
    Email: <input asp-for="Email" /> <br />
    <span asp-validation-for="Email"></span><br />
    Password: <input asp-for="Password" /><br />
    <span asp-validation-for="Password"></span><br />
    <button type="submit">Register</button>
</form>

```

O código HTML gerado (quando o modelo é válido):

```

<form action="/DemoReg/Register" method="post">
    <div class="validation-summary-valid" data-valmsg-summary="true">
        <ul><li style="display:none"></li></ul></div>
    Email: <input name="Email" id="Email" type="email" value="" data-val-required="The Email field is required." data-val-email="The Email field is not a valid email address." data-val="true"> <br />
    <span class="field-validation-valid" data-valmsg-replace="true" data-valmsg-for="Email"></span><br />
    Password: <input name="Password" id="Password" type="password" data-val-required="The Password field is required." data-val="true"><br />
    <span class="field-validation-valid" data-valmsg-replace="true" data-valmsg-for="Password"></span><br />
    <button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## Auxiliar de Marca de Seleção

- Gera `select` e os elementos `option` associados para as propriedades do modelo.
- Tem uma alternativa de Auxiliar HTML `Html.DropDownListFor` e `Html.ListBoxFor`

O `Select Tag Helper` `asp-for` especifica o nome da propriedade do modelo para o elemento `select` e `asp-items` especifica os elementos `option`. Por exemplo:

```

<select asp-for="Country" asp-items="Model.Countries"></select>

```

Amostra:

```

using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace FormsTagHelper.ViewModels
{
    public class CountryViewModel
    {
        public string Country { get; set; }

        public List<SelectListItem> Countries { get; } = new List<SelectListItem>
        {
            new SelectListItem { Value = "MX", Text = "Mexico" },
            new SelectListItem { Value = "CA", Text = "Canada" },
            new SelectListItem { Value = "US", Text = "USA" },
        };
    }
}

```

O método `Index` inicializa o `CountryViewModel`, define o país selecionado e o transmite para a exibição `Index`.

```

public IActionResult Index()
{
    var model = new CountryViewModel();
    model.Country = "CA";
    return View(model);
}

```

O método `Index` HTTP POST exibe a seleção:

```

[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Index(CountryViewModel model)
{
    if (ModelState.IsValid)
    {
        var msg = model.Country + " selected";
        return RedirectToAction("IndexSuccess", new { message = msg });
    }

    // If we got this far, something failed; redisplay form.
    return View(model);
}

```

A exibição `Index`:

```

@model CountryViewModel

<form asp-controller="Home" asp-action="Index" method="post">
    <select asp-for="Country" asp-items="Model.Countries"></select>
    <br /><button type="submit">Register</button>
</form>

```

Que gera o seguinte HTML (com "CA" selecionado):

```

<form method="post" action="/">
    <select id="Country" name="Country">
        <option value="MX">Mexico</option>
        <option selected="selected" value="CA">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## NOTE

Não é recomendável usar `ViewBag` ou `ViewData` com o Auxiliar de Marca de Seleção. Um modelo de exibição é mais robusto para fornecer metadados MVC e, geralmente, menos problemático.

O valor do atributo `asp-for` é um caso especial e não requer um prefixo `Model`, os outros atributos do Auxiliar de marca requerem (como `asp-items`)

```
<select asp-for="Country" asp-items="Model.Countries"></select>
```

## Associação de enumeração

Geralmente, é conveniente usar `<select>` com uma propriedade `enum` e gerar os elementos `SelectListItem` dos valores `enum`.

Amostra:

```

public class CountryEnumViewModel
{
    public CountryEnum EnumCountry { get; set; }
}

```

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public enum CountryEnum
    {
        [Display(Name = "United Mexican States")]
        Mexico,
        [Display(Name = "United States of America")]
        USA,
        Canada,
        France,
        Germany,
        Spain
    }
}

```

O método `GetEnumSelectList` gera um objeto `selectList` para uma enumeração.

```

@model CountryEnumViewModel

<form asp-controller="Home" asp-action="IndexEnum" method="post">
    <select asp-for="EnumCountry"
            asp-items="Html.GetEnumSelectList<CountryEnum>()">
    </select>
    <br /><button type="submit">Register</button>
</form>

```

É possível decorar sua lista de enumeradores com o atributo `Display` para obter uma interface do usuário mais rica:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public enum CountryEnum
    {
        [Display(Name = "United Mexican States")]
        Mexico,
        [Display(Name = "United States of America")]
        USA,
        Canada,
        France,
        Germany,
        Spain
    }
}

```

O HTML a seguir é gerado:

```

<form method="post" action="/Home/IndexEnum">
    <select data-val="true" data-val-required="The EnumCountry field is required."
            id="EnumCountry" name="EnumCountry">
        <option value="0">United Mexican States</option>
        <option value="1">United States of America</option>
        <option value="2">Canada</option>
        <option value="3">France</option>
        <option value="4">Germany</option>
        <option selected="selected" value="5">Spain</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## Grupo de opções

O elemento HTML `<optgroup>` é gerado quando o modelo de exibição contém um ou mais objetos `SelectListGroup`.

O `CountryViewModelGroup` agrupa os elementos `SelectListItem` nos grupos "América do Norte" e "Europa":

```

public class CountryViewModelGroup
{
    public CountryViewModelGroup()
    {
        var NorthAmericaGroup = new SelectListGroup { Name = "North America" };
        var EuropeGroup = new SelectListGroup { Name = "Europe" };

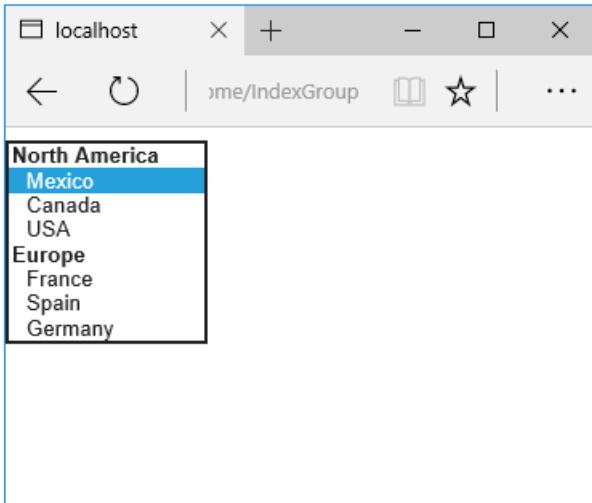
        Countries = new List<SelectListItem>
        {
            new SelectListItem
            {
                Value = "MEX",
                Text = "Mexico",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "CAN",
                Text = "Canada",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "US",
                Text = "USA",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "FR",
                Text = "France",
                Group = EuropeGroup
            },
            new SelectListItem
            {
                Value = "ES",
                Text = "Spain",
                Group = EuropeGroup
            },
            new SelectListItem
            {
                Value = "DE",
                Text = "Germany",
                Group = EuropeGroup
            }
        };
    }

    public string Country { get; set; }

    public List<SelectListItem> Countries { get; }
}

```

Os dois grupos são mostrados abaixo:



O HTML gerado:

```
<form method="post" action="/Home/IndexGroup">
    <select id="Country" name="Country">
        <optgroup label="North America">
            <option value="MEX">Mexico</option>
            <option value="CAN">Canada</option>
            <option value="US">USA</option>
        </optgroup>
        <optgroup label="Europe">
            <option value="FR">France</option>
            <option value="ES">Spain</option>
            <option value="DE">Germany</option>
        </optgroup>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="removed for brevity" />
</form>
```

## Seleção múltipla

O Auxiliar de Marca de Seleção gerará automaticamente o atributo `multiple = "multiple"` se a propriedade especificada no atributo `asp-for` for um `IEnumerable`. Por exemplo, considerando o seguinte modelo:

```
using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace FormsTagHelper.ViewModels
{
    public class CountryViewModel : IEnumerable
    {
        public IEnumerable<string> CountryCodes { get; set; }

        public List<SelectListItem> Countries { get; } = new List<SelectListItem>
        {
            new SelectListItem { Value = "MX", Text = "Mexico" },
            new SelectListItem { Value = "CA", Text = "Canada" },
            new SelectListItem { Value = "US", Text = "USA" },
            new SelectListItem { Value = "FR", Text = "France" },
            new SelectListItem { Value = "ES", Text = "Spain" },
            new SelectListItem { Value = "DE", Text = "Germany" }
        };
    }
}
```

Com a seguinte exibição:

```
@model CountryViewModelIEnumarable

<form asp-controller="Home" asp-action="IndexMultiSelect" method="post">
    <select asp-for="CountryCodes" asp-items="Model.Countries"></select>
    <br /><button type="submit">Register</button>
</form>
```

Gera o seguinte HTML:

```
<form method="post" action="/Home/IndexMultiSelect">
    <select id="CountryCodes"
        multiple="multiple"
        name="CountryCodes"><option value="MX">Mexico</option>
    <option value="CA">Canada</option>
    <option value="US">USA</option>
    <option value="FR">France</option>
    <option value="ES">Spain</option>
    <option value="DE">Germany</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## Nenhuma seleção

Se acabar usando a opção "não especificado" em várias páginas, você poderá criar um modelo para eliminar o HTML de repetição:

```
@model CountryViewModel

<form asp-controller="Home" asp-action="IndexEmpty" method="post">
    @Html.EditorForModel()
    <br /><button type="submit">Register</button>
</form>
```

O modelo *Views/Shared/EditorTemplates/CountryViewModel.cshtml*:

```
@model CountryViewModel

<select asp-for="Country" asp-items="Model.Countries">
    <option value="">--none--</option>
</select>
```

O acréscimo de elementos HTML `<option>` não está limitado ao caso de *Nenhuma seleção*. Por exemplo, o seguinte método de ação e exibição gerarão HTML semelhante ao código acima:

```
public IActionResult IndexOption(int id)
{
    var model = new CountryViewModel();
    model.Country = "CA";
    return View(model);
}
```

```
@model CountryViewModel

<form asp-controller="Home" asp-action="IndexEmpty" method="post">
    <select asp-for="Country">
        <option value="">&lt;none&gt;</option>
        <option value="MX">Mexico</option>
        <option value="CA">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
</form>
```

O elemento `<option>` correto será selecionado (contém o atributo `selected="selected"`) dependendo do valor atual de `Country`.

```
<form method="post" action="/Home/IndexEmpty">
    <select id="Country" name="Country">
        <option value="">&lt;none&gt;</option>
        <option value="MX">Mexico</option>
        <option value="CA" selected="selected">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## Recursos adicionais

- [Auxiliares de Marca no ASP.NET Core](#)
- [Elemento de formulário HTML](#)
- [Token de verificação de solicitação](#)
- [Model binding no ASP.NET Core](#)
- [Validação de modelo no ASP.NET Core MVC](#)
- [Interface IAttributeAdapter](#)
- [Snippets de código para este documento](#)

# Auxiliares de marca em formulários no ASP.NET Core

14/01/2019 • 28 minutes to read • [Edit Online](#)

Por [Rick Anderson](#), [Dave Paquette](#) e [Jerrie Pelser](#)

Este documento demonstra como é o trabalho com Formulários e os elementos HTML usados comumente em um Formulário. O elemento HTML [Formulário](#) fornece o mecanismo primário que os aplicativos Web usam para postar dados para o servidor. A maior parte deste documento descreve os [Auxiliares de marca](#) e como eles podem ajudar você a criar formulários HTML robustos de forma produtiva. É recomendável que você leia [Introdução ao auxiliares de marca](#) antes de ler este documento.

Em muitos casos, os Auxiliares HTML fornecem uma abordagem alternativa a um Auxiliar de Marca específico, mas é importante reconhecer que os Auxiliares de Marca não substituem os Auxiliares HTML e que não há um Auxiliar de Marca para cada Auxiliar HTML. Quando existe um Auxiliar HTML alternativo, ele é mencionado.

## O Auxiliar de marca de formulário

O Auxiliar de marca de [formulário](#):

- Gera o valor do atributo HTML `<FORM>` `action` para uma ação do controlador MVC ou uma rota nomeada
- Gera um [Token de verificação de solicitação](#) oculto para evitar a falsificação de solicitações entre sites (quando usado com o atributo `[ValidateAntiForgeryToken]` no método de ação HTTP Post)
- Fornece o atributo `asp-route-<Parameter Name>`, em que `<Parameter Name>` é adicionado aos valores de rota. Os parâmetros `routeValues` para `Html.BeginForm` e `Html.BeginRouteForm` fornecem funcionalidade semelhante.
- Tem uma alternativa de Auxiliar HTML `Html.BeginForm` e `Html.BeginRouteForm`

Amostra:

```
<form asp-controller="Demo" asp-action="Register" method="post">
    <!-- Input and Submit elements -->
</form>
```

O Auxiliar de marca de formulário acima gera o HTML a seguir:

```
<form method="post" action="/Demo/Register">
    <!-- Input and Submit elements -->
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

O tempo de execução do MVC gera o valor do atributo `action` dos atributos `asp-controller` e `asp-action` do Auxiliar de marca de formulário. O Auxiliar de marca de formulário também gera um [Token de verificação de solicitação](#) oculto para evitar a falsificação de solicitações entre sites (quando usado com o atributo `[ValidateAntiForgeryToken]` no método de ação HTTP Post). É difícil proteger um Formulário HTML puro contra falsificação de solicitações entre sites e o Auxiliar de marca de formulário fornece este serviço para você.

### Usando uma rota nomeada

O atributo do Auxiliar de Marca `asp-route` também pode gerar a marcação para o atributo HTML `action`. Um

aplicativo com uma [rota](#) chamada `register` poderia usar a seguinte marcação para a página de registro:

```
<form asp-route="register" method="post">
    <!-- Input and Submit elements -->
</form>
```

Muitas das exibições na pasta *Modos de Exibição/Conta* (gerada quando você cria um novo aplicativo Web com *Contas de usuário individuais*) contêm o atributo [asp-route-returnurl](#):

```
<form asp-controller="Account" asp-action="Login"
      asp-route-returnurl="@ViewData["ReturnUrl"]"
      method="post" class="form-horizontal" role="form">
```

#### NOTE

Com os modelos internos, `returnUrl` só é preenchido automaticamente quando você tenta acessar um recurso autorizado, mas não está autenticado ou autorizado. Quando você tenta fazer um acesso não autorizado, o middleware de segurança o redireciona para a página de logon com o `returnUrl` definido.

## O auxiliar de marca de entrada

O Auxiliar de marca de entrada associa um elemento HTML `<input>` a uma expressão de modelo em sua exibição do Razor.

Sintaxe:

```
<input asp-for="<Expression Name>" />
```

O auxiliar de marca de entrada:

- Gera os atributos HTML `id` e `name` para o nome da expressão especificada no atributo `asp-for`.  
`asp-for="Property1.Property2"` equivale a `m => m.Property1.Property2`. O nome da expressão é o que é usado para o valor do atributo `asp-for`. Consulte a seção [Nomes de expressão](#) para obter informações adicionais.
- Define o valor do atributo HTML `type` com base nos atributos de tipo de modelo e [anotação de dados](#) aplicados à propriedade de modelo
- O valor do atributo HTML `type` não será substituído quando um for especificado
- Gera atributos de validação [HTML5](#) de atributos de [anotação de dados](#) aplicados a propriedades de modelo
- Tem uma sobreposição de recursos de Auxiliar HTML com `Html.TextBoxFor` e `Html.EditorFor`. Consulte a seção **Alternativas de Auxiliar HTML ao Auxiliar de marca de entrada** para obter detalhes.
- Fornece tipagem forte. Se o nome da propriedade for alterado e você não atualizar o Auxiliar de marca, você verá um erro semelhante ao seguinte:

An error occurred during the compilation of a resource required to process this request. Please review the following specific error details and modify your source code appropriately.

Type expected

'RegisterViewModel' does not contain a definition for 'Email' and no extension method 'Email' accepting a first argument of type 'RegisterViewModel' could be found (are you missing a using directive or an assembly reference?)

O Auxiliar de marca `Input` define o atributo HTML `type` com base no tipo .NET. A tabela a seguir lista alguns tipos .NET comuns e o tipo HTML gerado (não estão listados todos os tipos .NET).

TIPO .NET	TIPO DE ENTRADA
Bool	<code>type="checkbox"</code>
Cadeia de Caracteres	<code>type="text"</code>
DateTime	<code>type="datetime-local"</code>
Byte	<code>type="number"</code>
int	<code>type="number"</code>
Single e Double	<code>type="number"</code>

A tabela a seguir mostra alguns atributos de [anotações de dados](#) comuns que o auxiliar de marca de entrada mapeará para tipos de entrada específicos (não são listados todos os atributos de validação):

ATRIBUTO	TIPO DE ENTRADA
<code>[EmailAddress]</code>	<code>type="email"</code>
<code>[Url]</code>	<code>type="url"</code>
<code>[HiddenInput]</code>	<code>type="hidden"</code>
<code>[Phone]</code>	<code>type="tel"</code>
<code>[DataType(DataType.Password)]</code>	<code>type="password"</code>
<code>[DataType(DataType.Date)]</code>	<code>type="date"</code>
<code>[DataType(DataType.Time)]</code>	<code>type="time"</code>

Amostra:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}

```

```

@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterInput" method="post">
    Email: <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    <button type="submit">Register</button>
</form>

```

O código acima gera o seguinte HTML:

```

<form method="post" action="/Demo/RegisterInput">
    Email:
    <input type="email" data-val="true"
           data-val-email="The Email Address field is not a valid email address."
           data-val-required="The Email Address field is required."
           id="Email" name="Email" value="" /> <br>
    Password:
    <input type="password" data-val="true"
           data-val-required="The Password field is required."
           id="Password" name="Password" /><br>
    <button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

As anotações de dados aplicadas às propriedades `Email` e `Password` geram metadados no modelo. O Auxiliar de marca de entrada consome os metadados do modelo e produz atributos HTML5 `data-val-*` (consulte [Validação de modelo](#)). Esses atributos descrevem os validadores a serem anexados aos campos de entrada. Isso fornece validação de [jQuery](#) e HTML5 discreto. Os atributos discretos têm o formato `data-val-rule="Error Message"`, em que a regra é o nome da regra de validação (como `data-val-required`, `data-val-email`, `data-val-maxlength` etc.). Se uma mensagem de erro for fornecida no atributo, ela será exibida como o valor para o atributo `data-val-rule`. Também há atributos do formulário `data-val-ruleName-argumentName="argumentValue"` que fornecem detalhes adicionais sobre a regra, por exemplo, `data-val-maxlength-max="1024"`.

## Alternativas de Auxiliar HTML ao Auxiliar de marca de entrada

`Html.TextBox`, `Html.TextBoxFor`, `Html.Editor` e `Html.EditorFor` têm recursos que se sobrepõem aos do Auxiliar de marca de entrada. O Auxiliar de marca de entrada define automaticamente o atributo `type`; `Html.TextBox` e `Html.TextBoxFor` não o fazem. `Html.Editor` e `Html.EditorFor` manipulam coleções, objetos complexos e modelos; o Auxiliar de marca de entrada não o faz. O Auxiliar de marca de entrada, `Html.EditorFor` e `Html.TextBoxFor` são fortemente tipados (eles usam expressões lambda); `Html.TextBox` e `Html.Editor` não usam (eles usam nomes de expressão).

## HtmlAttributes

`@Html.Editor()` e `@Html.EditorFor()` usam uma entrada `ViewDataDictionary` especial chamada `htmlAttributes` ao executar seus modelos padrão. Esse comportamento pode ser aumentado usando parâmetros `additional ViewData`. A chave "htmlAttributes" diferencia maiúsculas de minúsculas. A chave "htmlAttributes" é tratada de forma semelhante ao objeto `htmlAttributes` passado para auxiliares de entrada como `@Html.TextBox()`.

```
@Html.EditorFor(model => model.YourProperty,  
new { htmlAttributes = new { @class="myCssClass", style="Width:100px" } })
```

## Nomes de expressão

O valor do atributo `asp-for` é um `ModelExpression` e o lado direito de uma expressão lambda. Portanto, `asp-for="Property1"` se torna `m => m.Property1` no código gerado e é por isso você não precisa colocar o prefixo `Model`. Você pode usar o caractere "@" para iniciar uma expressão embutida e mover para antes de `m`:

```
@{  
    var joe = "Joe";  
}  
<input asp-for="@joe" />
```

Gera o seguinte:

```
<input type="text" id="joe" name="joe" value="Joe" />
```

Com propriedades de coleção, `asp-for="CollectionProperty[23].Member"` gera o mesmo nome que `asp-for="CollectionProperty[i].Member"` quando `i` tem o valor `23`.

Quando o ASP.NET Core MVC calcula o valor de `ModelExpression`, ele inspeciona várias fontes, inclusive o `ModelState`. Considere o `<input type="text" asp-for="@Name" />`. O atributo `value` calculado é o primeiro valor não nulo:

- Da entrada de `ModelState` com a chave "Name".
- Do resultado da expressão `Model.Name`.

## Navegando para propriedades filho

Você também pode navegar para propriedades filho usando o caminho da propriedade do modelo de exibição. Considere uma classe de modelo mais complexa que contém uma propriedade `Address` filho.

```
public class AddressViewModel  
{  
    public string AddressLine1 { get; set; }  
}
```

```
public class RegisterAddressViewModel  
{  
    public string Email { get; set; }  
  
    [DataType(DataType.Password)]  
    public string Password { get; set; }  
  
    public AddressViewModel Address { get; set; }  
}
```

Na exibição, associamos a `Address.AddressLine1`:

```
@model RegisterAddressViewModel

<form asp-controller="Demo" asp-action="RegisterAddress" method="post">
    Email: <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    Address: <input asp-for="Address.AddressLine1" /><br />
    <button type="submit">Register</button>
</form>
```

O HTML a seguir é gerado para `Address.AddressLine1`:

```
<input type="text" id="Address_AddressLine1" name="Address.AddressLine1" value="" />
```

## Nomes de expressão e coleções

Exemplo, um modelo que contém uma matriz de `Colors`:

```
public class Person
{
    public List<string> Colors { get; set; }

    public int Age { get; set; }
}
```

O método de ação:

```
public IActionResult Edit(int id, int colorIndex)
{
    ViewData["Index"] = colorIndex;
    return View(GetPerson(id));
}
```

O Razor a seguir mostra como você acessa um elemento `Color` específico:

```
@model Person
 @{
     var index = (int)ViewData["index"];
 }

<form asp-controller="ToDo" asp-action="Edit" method="post">
    @Html.EditorFor(m => m.Colors[index])
    <label asp-for="Age"></label>
    <input asp-for="Age" /><br />
    <button type="submit">Post</button>
</form>
```

O modelo `Views/Shared/EditorTemplates/String.cshtml`:

```
@model string

<label asp-for="@Model"></label>
<input asp-for="@Model" /> <br />
```

Exemplo usando `List<T>`:

```

public class ToDoItem
{
    public string Name { get; set; }

    public bool IsDone { get; set; }
}

```

O Razor a seguir mostra como iterar em uma coleção:

```

@model List<ToDoItem>

<form asp-controller="ToDo" asp-action="Edit" method="post">
    <table>
        <tr> <th>Name</th> <th>Is Done</th> </tr>

        @for (int i = 0; i < Model.Count; i++)
        {
            <tr>
                @Html.EditorFor(model => model[i])
            </tr>
        }

    </table>
    <button type="submit">Save</button>
</form>

```

O modelo *Views/Shared/EditorTemplates/ToDoItem.cshtml*:

```

@model ToDoItem

<td>
    <label asp-for="@Model.Name"></label>
    @Html.DisplayFor(model => model.Name)
</td>
<td>
    <input asp-for="@Model.IsDone" />
</td>

/*
This template replaces the following Razor which evaluates the indexer three times.
<td>
    <label asp-for="@Model[i].Name"></label>
    @Html.DisplayFor(model => model[i].Name)
</td>
<td>
    <input asp-for="@Model[i].IsDone" />
</td>
*/

```

`foreach` deve ser usado, se possível, quando o valor está prestes a ser usado em um contexto equivalente `asp-for` ou `Html.DisplayFor`. Em geral, `for` é melhor do que `foreach` (se o cenário permitir) porque não é necessário alocar um enumerador; no entanto, avaliar um indexador em uma expressão LINQ pode ser caro, o que deve ser minimizado.

#### NOTE

O código de exemplo comentado acima mostra como você substituiria a expressão lambda pelo operador `@` para acessar cada `ToDoItem` na lista.

## Auxiliar de marca de área de texto

O auxiliar de marca `Textarea Tag Helper` é semelhante ao Auxiliar de marca de entrada.

- Gera os atributos `id` e `name`, bem como os atributos de validação de dados do modelo para um elemento `<textarea>`.
- Fornece tipagem forte.
- Alternativa de Auxiliar HTML: `Html.TextAreaFor`

Amostra:

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class DescriptionViewModel
    {
        [MinLength(5)]
        [MaxLength(1024)]
        public string Description { get; set; }
    }
}
```

```
@model DescriptionViewModel

<form asp-controller="Demo" asp-action="RegisterTextArea" method="post">
    <textarea asp-for="Description"></textarea>
    <button type="submit">Test</button>
</form>
```

O HTML a seguir é gerado:

```
<form method="post" action="/Demo/RegisterTextArea">
    <textarea data-val="true"
        data-val-maxlength="The field Description must be a string or array type with a maximum length of
        1024."
        data-val-maxlength-max="1024"
        data-val-minlength="The field Description must be a string or array type with a minimum length of
        5."
        data-val-minlength-min="5"
        id="Description" name="Description">
    </textarea>
    <button type="submit">Test</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## O auxiliar de marca de rótulo

- Gera a legenda do rótulo e o atributo `for` em um elemento para um nome de expressão
- Alternativa de Auxiliar HTML: `Html.LabelFor`

O **Label Tag Helper** fornece os seguintes benefícios em comparação com um elemento de rótulo HTML puro:

- Você obtém automaticamente o valor do rótulo descritivo do atributo `Display`. O nome de exibição desejado pode mudar com o tempo e a combinação do atributo `Display` e do Auxiliar de Marca de Rótulo aplicará `Display` em qualquer lugar em que for usado.
- Menos marcação no código-fonte
- Tipagem forte com a propriedade de modelo.

Amostra:

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class SimpleViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }
    }
}
```

```
@model SimpleViewModel

<form asp-controller="Demo" asp-action="RegisterLabel" method="post">
    <label asp-for="Email"></label>
    <input asp-for="Email" /> <br />
</form>
```

O HTML a seguir é gerado para o elemento `<label>`:

```
<label for="Email">Email Address</label>
```

O Auxiliar de marca de rótulo gerou o valor do atributo `for` de "Email", que é a ID associada ao elemento `<input>`. Auxiliares de marca geram elementos `id` e `for` consistentes para que eles possam ser associados corretamente. A legenda neste exemplo é proveniente do atributo `Display`. Se o modelo não contivesse um atributo `Display`, a legenda seria o nome da propriedade da expressão.

## Os auxiliares de marca de validação

Há dois auxiliares de marca de validação. O **Validation Message Tag Helper** (que exibe uma mensagem de validação para uma única propriedade em seu modelo) e o **Validation Summary Tag Helper** (que exibe um resumo dos erros de validação). O **Input Tag Helper** adiciona atributos de validação do lado do cliente HTML5 para elementos de entrada baseados em atributos de anotação de dados em suas classes de modelo. A validação também é executada no servidor. O Auxiliar de marca de validação exibe essas mensagens de erro quando ocorre um erro de validação.

### O Auxiliar de marca de mensagem de validação

- Adiciona o atributo `data-valmsg-for="property"` [HTML5](#) ao elemento `span`, que anexa as mensagens de erro de validação no campo de entrada da propriedade do modelo especificado. Quando ocorre um erro de validação do lado do cliente, [jQuery](#) exibe a mensagem de erro no elemento `<span>`.

- A validação também é feita no servidor. Os clientes poderão ter o JavaScript desabilitado e parte da validação só pode ser feita no lado do servidor.

- Alternativa de Auxiliar HTML: `@Html.ValidationMessageFor`

O `Validation Message Tag Helper` é usado com o atributo `asp-validation-for` em um elemento HTML `span`.

```
<span asp-validation-for="Email"></span>
```

O Auxiliar de marca de mensagem de validação gerará o HTML a seguir:

```
<span class="field-validation-valid"
      data-valmsg-for="Email"
      data-valmsg-replace="true"></span>
```

Geralmente, você usa o `Validation Message Tag Helper` após um Auxiliar de marca `Input` para a mesma propriedade. Fazer isso exibe as mensagens de erro de validação próximo à entrada que causou o erro.

#### NOTE

É necessário ter uma exibição com as referências de script `jQuery` e JavaScript corretas em vigor para a validação do lado do cliente. Consulte [Validação de Modelo](#) para obter mais informações.

Quando ocorre um erro de validação do lado do servidor (por exemplo, quando você tem validação do lado do servidor personalizada ou a validação do lado do cliente está desabilitada), o MVC coloca essa mensagem de erro como o corpo do elemento `<span>`.

```
<span class="field-validation-error" data-valmsg-for="Email"
      data-valmsg-replace="true">
    The Email Address field is required.
</span>
```

#### Auxiliar de marca de resumo de validação

- Tem como alvo elementos `<div>` com o atributo `asp-validation-summary`
- Alternativa de Auxiliar HTML: `@Html.ValidationSummary`

O `Validation Summary Tag Helper` é usado para exibir um resumo das mensagens de validação. O valor do atributo `asp-validation-summary` pode ser qualquer um dos seguintes:

ASP-VALIDATION-SUMMARY	MENSAGENS DE VALIDAÇÃO EXIBIDAS
ValidationSummary.All	Nível da propriedade e do modelo
ValidationSummary.ModelOnly	Modelo
ValidationSummary.None	Nenhum

#### Amostra

No exemplo a seguir, o modelo de dados é decorado com atributos `DataAnnotation`, o que gera mensagens de erro de validação no elemento `<input>`. Quando ocorre um erro de validação, o Auxiliar de marca de validação exibe a mensagem de erro:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}

```

```

@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterValidation" method="post">
    <div asp-validation-summary="ModelOnly"></div>
    Email: <input asp-for="Email" /> <br />
    <span asp-validation-for="Email"></span><br />
    Password: <input asp-for="Password" /><br />
    <span asp-validation-for="Password"></span><br />
    <button type="submit">Register</button>
</form>

```

O código HTML gerado (quando o modelo é válido):

```

<form action="/DemoReg/Register" method="post">
    <div class="validation-summary-valid" data-valmsg-summary="true">
        <ul><li style="display:none"></li></ul></div>
    Email: <input name="Email" id="Email" type="email" value="" data-val-required="The Email field is required." data-val-email="The Email field is not a valid email address." data-val="true"> <br />
    <span class="field-validation-valid" data-valmsg-replace="true" data-valmsg-for="Email"></span><br />
    Password: <input name="Password" id="Password" type="password" data-val-required="The Password field is required." data-val="true"><br />
    <span class="field-validation-valid" data-valmsg-replace="true" data-valmsg-for="Password"></span><br />
    <button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## Auxiliar de Marca de Seleção

- Gera `select` e os elementos `option` associados para as propriedades do modelo.
- Tem uma alternativa de Auxiliar HTML `Html.DropDownListFor` e `Html.ListBoxFor`

O `Select Tag Helper` `asp-for` especifica o nome da propriedade do modelo para o elemento `select` e `asp-items` especifica os elementos `option`. Por exemplo:

```

<select asp-for="Country" asp-items="Model.Countries"></select>

```

Amostra:

```

using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace FormsTagHelper.ViewModels
{
    public class CountryViewModel
    {
        public string Country { get; set; }

        public List<SelectListItem> Countries { get; } = new List<SelectListItem>
        {
            new SelectListItem { Value = "MX", Text = "Mexico" },
            new SelectListItem { Value = "CA", Text = "Canada" },
            new SelectListItem { Value = "US", Text = "USA" },
        };
    }
}

```

O método `Index` inicializa o `CountryViewModel`, define o país selecionado e o transmite para a exibição `Index`.

```

public IActionResult Index()
{
    var model = new CountryViewModel();
    model.Country = "CA";
    return View(model);
}

```

O método `Index` HTTP POST exibe a seleção:

```

[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Index(CountryViewModel model)
{
    if (ModelState.IsValid)
    {
        var msg = model.Country + " selected";
        return RedirectToAction("IndexSuccess", new { message = msg });
    }

    // If we got this far, something failed; redisplay form.
    return View(model);
}

```

A exibição `Index`:

```

@model CountryViewModel

<form asp-controller="Home" asp-action="Index" method="post">
    <select asp-for="Country" asp-items="Model.Countries"></select>
    <br /><button type="submit">Register</button>
</form>

```

Que gera o seguinte HTML (com "CA" selecionado):

```

<form method="post" action="/">
    <select id="Country" name="Country">
        <option value="MX">Mexico</option>
        <option selected="selected" value="CA">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## NOTE

Não é recomendável usar `ViewBag` ou  `ViewData` com o Auxiliar de Marca de Seleção. Um modelo de exibição é mais robusto para fornecer metadados MVC e, geralmente, menos problemático.

O valor do atributo `asp-for` é um caso especial e não requer um prefixo `Model`, os outros atributos do Auxiliar de marca requerem (como `asp-items`)

```
<select asp-for="Country" asp-items="Model.Countries"></select>
```

## Associação de enumeração

Geralmente, é conveniente usar `<select>` com uma propriedade `enum` e gerar os elementos `SelectListItem` dos valores `enum`.

Amostra:

```

public class CountryEnumViewModel
{
    public CountryEnum EnumCountry { get; set; }
}

```

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public enum CountryEnum
    {
        [Display(Name = "United Mexican States")]
        Mexico,
        [Display(Name = "United States of America")]
        USA,
        Canada,
        France,
        Germany,
        Spain
    }
}

```

O método `GetEnumSelectList` gera um objeto `selectList` para uma enumeração.

```

@model CountryEnumViewModel

<form asp-controller="Home" asp-action="IndexEnum" method="post">
    <select asp-for="EnumCountry"
            asp-items="Html.GetEnumSelectList<CountryEnum>()">
    </select>
    <br /><button type="submit">Register</button>
</form>

```

É possível decorar sua lista de enumeradores com o atributo `Display` para obter uma interface do usuário mais rica:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public enum CountryEnum
    {
        [Display(Name = "United Mexican States")]
        Mexico,
        [Display(Name = "United States of America")]
        USA,
        Canada,
        France,
        Germany,
        Spain
    }
}

```

O HTML a seguir é gerado:

```

<form method="post" action="/Home/IndexEnum">
    <select data-val="true" data-val-required="The EnumCountry field is required." id="EnumCountry" name="EnumCountry">
        <option value="0">United Mexican States</option>
        <option value="1">United States of America</option>
        <option value="2">Canada</option>
        <option value="3">France</option>
        <option value="4">Germany</option>
        <option selected="selected" value="5">Spain</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## Grupo de opções

O elemento HTML `<optgroup>` é gerado quando o modelo de exibição contém um ou mais objetos `SelectListGroup`.

O `CountryViewModelGroup` agrupa os elementos `SelectListItem` nos grupos "América do Norte" e "Europa":

```

public class CountryViewModelGroup
{
    public CountryViewModelGroup()
    {
        var NorthAmericaGroup = new SelectListGroup { Name = "North America" };
        var EuropeGroup = new SelectListGroup { Name = "Europe" };

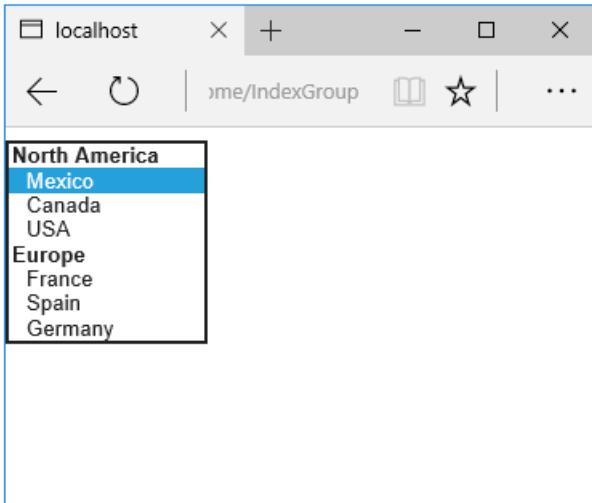
        Countries = new List<SelectListItem>
        {
            new SelectListItem
            {
                Value = "MEX",
                Text = "Mexico",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "CAN",
                Text = "Canada",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "US",
                Text = "USA",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "FR",
                Text = "France",
                Group = EuropeGroup
            },
            new SelectListItem
            {
                Value = "ES",
                Text = "Spain",
                Group = EuropeGroup
            },
            new SelectListItem
            {
                Value = "DE",
                Text = "Germany",
                Group = EuropeGroup
            }
        };
    }

    public string Country { get; set; }

    public List<SelectListItem> Countries { get; }
}

```

Os dois grupos são mostrados abaixo:



O HTML gerado:

```
<form method="post" action="/Home/IndexGroup">
    <select id="Country" name="Country">
        <optgroup label="North America">
            <option value="MEX">Mexico</option>
            <option value="CAN">Canada</option>
            <option value="US">USA</option>
        </optgroup>
        <optgroup label="Europe">
            <option value="FR">France</option>
            <option value="ES">Spain</option>
            <option value="DE">Germany</option>
        </optgroup>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="removed for brevity" />
</form>
```

## Seleção múltipla

O Auxiliar de Marca de Seleção gerará automaticamente o atributo `multiple = "multiple"` se a propriedade especificada no atributo `asp-for` for um `IEnumerable`. Por exemplo, considerando o seguinte modelo:

```
using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace FormsTagHelper.ViewModels
{
    public class CountryViewModel : IEnumerable
    {
        public IEnumerable<string> CountryCodes { get; set; }

        public List<SelectListItem> Countries { get; } = new List<SelectListItem>
        {
            new SelectListItem { Value = "MX", Text = "Mexico" },
            new SelectListItem { Value = "CA", Text = "Canada" },
            new SelectListItem { Value = "US", Text = "USA" },
            new SelectListItem { Value = "FR", Text = "France" },
            new SelectListItem { Value = "ES", Text = "Spain" },
            new SelectListItem { Value = "DE", Text = "Germany" }
        };
    }
}
```

Com a seguinte exibição:

```
@model CountryViewModelIEnumarable

<form asp-controller="Home" asp-action="IndexMultiSelect" method="post">
    <select asp-for="CountryCodes" asp-items="Model.Countries"></select>
    <br /><button type="submit">Register</button>
</form>
```

Gera o seguinte HTML:

```
<form method="post" action="/Home/IndexMultiSelect">
    <select id="CountryCodes"
        multiple="multiple"
        name="CountryCodes"><option value="MX">Mexico</option>
    <option value="CA">Canada</option>
    <option value="US">USA</option>
    <option value="FR">France</option>
    <option value="ES">Spain</option>
    <option value="DE">Germany</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## Nenhuma seleção

Se acabar usando a opção "não especificado" em várias páginas, você poderá criar um modelo para eliminar o HTML de repetição:

```
@model CountryViewModel

<form asp-controller="Home" asp-action="IndexEmpty" method="post">
    @Html.EditorForModel()
    <br /><button type="submit">Register</button>
</form>
```

O modelo *Views/Shared/EditorTemplates/CountryViewModel.cshtml*:

```
@model CountryViewModel

<select asp-for="Country" asp-items="Model.Countries">
    <option value="">--none--</option>
</select>
```

O acréscimo de elementos HTML `<option>` não está limitado ao caso de *Nenhuma seleção*. Por exemplo, o seguinte método de ação e exibição gerarão HTML semelhante ao código acima:

```
public IActionResult IndexOption(int id)
{
    var model = new CountryViewModel();
    model.Country = "CA";
    return View(model);
}
```

```
@model CountryViewModel

<form asp-controller="Home" asp-action="IndexEmpty" method="post">
    <select asp-for="Country">
        <option value="">&lt;none&gt;</option>
        <option value="MX">Mexico</option>
        <option value="CA">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
</form>
```

O elemento `<option>` correto será selecionado (contém o atributo `selected="selected"`) dependendo do valor atual de `Country`.

```
<form method="post" action="/Home/IndexEmpty">
    <select id="Country" name="Country">
        <option value="">&lt;none&gt;</option>
        <option value="MX">Mexico</option>
        <option value="CA" selected="selected">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## Recursos adicionais

- [Auxiliares de Marca no ASP.NET Core](#)
- [Elemento de formulário HTML](#)
- [Token de verificação de solicitação](#)
- [Model binding no ASP.NET Core](#)
- [Validação de modelo no ASP.NET Core MVC](#)
- [Interface IAttributeAdapter](#)
- [Snippets de código para este documento](#)

# Auxiliares de marca em formulários no ASP.NET Core

14/01/2019 • 28 minutes to read • [Edit Online](#)

Por [Rick Anderson](#), [Dave Paquette](#) e [Jerrie Pelser](#)

Este documento demonstra como é o trabalho com Formulários e os elementos HTML usados comumente em um Formulário. O elemento HTML [Formulário](#) fornece o mecanismo primário que os aplicativos Web usam para postar dados para o servidor. A maior parte deste documento descreve os [Auxiliares de marca](#) e como eles podem ajudar você a criar formulários HTML robustos de forma produtiva. É recomendável que você leia [Introdução ao auxiliares de marca](#) antes de ler este documento.

Em muitos casos, os Auxiliares HTML fornecem uma abordagem alternativa a um Auxiliar de Marca específico, mas é importante reconhecer que os Auxiliares de Marca não substituem os Auxiliares HTML e que não há um Auxiliar de Marca para cada Auxiliar HTML. Quando existe um Auxiliar HTML alternativo, ele é mencionado.

## O Auxiliar de marca de formulário

O Auxiliar de marca de [formulário](#):

- Gera o valor do atributo HTML `<FORM>` `action` para uma ação do controlador MVC ou uma rota nomeada
- Gera um [Token de verificação de solicitação](#) oculto para evitar a falsificação de solicitações entre sites (quando usado com o atributo `[ValidateAntiForgeryToken]` no método de ação HTTP Post)
- Fornece o atributo `asp-route-<Parameter Name>`, em que `<Parameter Name>` é adicionado aos valores de rota. Os parâmetros `routeValues` para `Html.BeginForm` e `Html.BeginRouteForm` fornecem funcionalidade semelhante.
- Tem uma alternativa de Auxiliar HTML `Html.BeginForm` e `Html.BeginRouteForm`

Amostra:

```
<form asp-controller="Demo" asp-action="Register" method="post">
    <!-- Input and Submit elements -->
</form>
```

O Auxiliar de marca de formulário acima gera o HTML a seguir:

```
<form method="post" action="/Demo/Register">
    <!-- Input and Submit elements -->
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

O tempo de execução do MVC gera o valor do atributo `action` dos atributos `asp-controller` e `asp-action` do Auxiliar de marca de formulário. O Auxiliar de marca de formulário também gera um [Token de verificação de solicitação](#) oculto para evitar a falsificação de solicitações entre sites (quando usado com o atributo `[ValidateAntiForgeryToken]` no método de ação HTTP Post). É difícil proteger um Formulário HTML puro contra falsificação de solicitações entre sites e o Auxiliar de marca de formulário fornece este serviço para você.

### Usando uma rota nomeada

O atributo do Auxiliar de Marca `asp-route` também pode gerar a marcação para o atributo HTML `action`. Um

aplicativo com uma [rota](#) chamada `register` poderia usar a seguinte marcação para a página de registro:

```
<form asp-route="register" method="post">
    <!-- Input and Submit elements -->
</form>
```

Muitas das exibições na pasta *Modos de Exibição/Conta* (gerada quando você cria um novo aplicativo Web com *Contas de usuário individuais*) contêm o atributo [asp-route-returnurl](#):

```
<form asp-controller="Account" asp-action="Login"
      asp-route-returnurl="@ViewData["ReturnUrl"]"
      method="post" class="form-horizontal" role="form">
```

#### NOTE

Com os modelos internos, `returnUrl` só é preenchido automaticamente quando você tenta acessar um recurso autorizado, mas não está autenticado ou autorizado. Quando você tenta fazer um acesso não autorizado, o middleware de segurança o redireciona para a página de logon com o `returnUrl` definido.

## O auxiliar de marca de entrada

O Auxiliar de marca de entrada associa um elemento HTML `<input>` a uma expressão de modelo em sua exibição do Razor.

Sintaxe:

```
<input asp-for="<Expression Name>" />
```

O auxiliar de marca de entrada:

- Gera os atributos HTML `id` e `name` para o nome da expressão especificada no atributo `asp-for`.  
`asp-for="Property1.Property2"` equivale a `m => m.Property1.Property2`. O nome da expressão é o que é usado para o valor do atributo `asp-for`. Consulte a seção [Nomes de expressão](#) para obter informações adicionais.
- Define o valor do atributo HTML `type` com base nos atributos de tipo de modelo e [anotação de dados](#) aplicados à propriedade de modelo
- O valor do atributo HTML `type` não será substituído quando um for especificado
- Gera atributos de validação [HTML5](#) de atributos de [anotação de dados](#) aplicados a propriedades de modelo
- Tem uma sobreposição de recursos de Auxiliar HTML com `Html.TextBoxFor` e `Html.EditorFor`. Consulte a seção **Alternativas de Auxiliar HTML ao Auxiliar de marca de entrada** para obter detalhes.
- Fornece tipagem forte. Se o nome da propriedade for alterado e você não atualizar o Auxiliar de marca, você verá um erro semelhante ao seguinte:

An error occurred during the compilation of a resource required to process this request. Please review the following specific error details and modify your source code appropriately.

Type expected

'RegisterViewModel' does not contain a definition for 'Email' and no extension method 'Email' accepting a first argument of type 'RegisterViewModel' could be found (are you missing a using directive or an assembly reference?)

O Auxiliar de marca `Input` define o atributo HTML `type` com base no tipo .NET. A tabela a seguir lista alguns tipos .NET comuns e o tipo HTML gerado (não estão listados todos os tipos .NET).

TIPO .NET	TIPO DE ENTRADA
Bool	<code>type="checkbox"</code>
Cadeia de Caracteres	<code>type="text"</code>
DateTime	<code>type="datetime-local"</code>
Byte	<code>type="number"</code>
int	<code>type="number"</code>
Single e Double	<code>type="number"</code>

A tabela a seguir mostra alguns atributos de [anotações de dados](#) comuns que o auxiliar de marca de entrada mapeará para tipos de entrada específicos (não são listados todos os atributos de validação):

ATRIBUTO	TIPO DE ENTRADA
<code>[EmailAddress]</code>	<code>type="email"</code>
<code>[Url]</code>	<code>type="url"</code>
<code>[HiddenInput]</code>	<code>type="hidden"</code>
<code>[Phone]</code>	<code>type="tel"</code>
<code>[DataType(DataType.Password)]</code>	<code>type="password"</code>
<code>[DataType(DataType.Date)]</code>	<code>type="date"</code>
<code>[DataType(DataType.Time)]</code>	<code>type="time"</code>

Amostra:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}

```

```

@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterInput" method="post">
    Email: <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    <button type="submit">Register</button>
</form>

```

O código acima gera o seguinte HTML:

```

<form method="post" action="/Demo/RegisterInput">
    Email:
    <input type="email" data-val="true"
           data-val-email="The Email Address field is not a valid email address."
           data-val-required="The Email Address field is required."
           id="Email" name="Email" value="" /> <br>
    Password:
    <input type="password" data-val="true"
           data-val-required="The Password field is required."
           id="Password" name="Password" /><br>
    <button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

As anotações de dados aplicadas às propriedades `Email` e `Password` geram metadados no modelo. O Auxiliar de marca de entrada consome os metadados do modelo e produz atributos HTML5 `data-val-*` (consulte [Validação de modelo](#)). Esses atributos descrevem os validadores a serem anexados aos campos de entrada. Isso fornece validação de [jQuery](#) e HTML5 discreto. Os atributos discretos têm o formato `data-val-rule="Error Message"`, em que a regra é o nome da regra de validação (como `data-val-required`, `data-val-email`, `data-val-maxlength` etc.). Se uma mensagem de erro for fornecida no atributo, ela será exibida como o valor para o atributo `data-val-rule`. Também há atributos do formulário `data-val-ruleName-argumentName="argumentValue"` que fornecem detalhes adicionais sobre a regra, por exemplo, `data-val-maxlength-max="1024"`.

## Alternativas de Auxiliar HTML ao Auxiliar de marca de entrada

`Html.TextBox`, `Html.TextBoxFor`, `Html.Editor` e `Html.EditorFor` têm recursos que se sobrepõem aos do Auxiliar de marca de entrada. O Auxiliar de marca de entrada define automaticamente o atributo `type`; `Html.TextBox` e `Html.TextBoxFor` não o fazem. `Html.Editor` e `Html.EditorFor` manipulam coleções, objetos complexos e modelos; o Auxiliar de marca de entrada não o faz. O Auxiliar de marca de entrada, `Html.EditorFor` e `Html.TextBoxFor` são fortemente tipados (eles usam expressões lambda); `Html.TextBox` e `Html.Editor` não usam (eles usam nomes de expressão).

## HtmlAttributes

`@Html.Editor()` e `@Html.EditorFor()` usam uma entrada `ViewDataDictionary` especial chamada `htmlAttributes` ao executar seus modelos padrão. Esse comportamento pode ser aumentado usando parâmetros `additional ViewData`. A chave "htmlAttributes" diferencia maiúsculas de minúsculas. A chave "htmlAttributes" é tratada de forma semelhante ao objeto `htmlAttributes` passado para auxiliares de entrada como `@Html.TextBox()`.

```
@Html.EditorFor(model => model.YourProperty,  
new { htmlAttributes = new { @class="myCssClass", style="Width:100px" } })
```

## Nomes de expressão

O valor do atributo `asp-for` é um `ModelExpression` e o lado direito de uma expressão lambda. Portanto, `asp-for="Property1"` se torna `m => m.Property1` no código gerado e é por isso você não precisa colocar o prefixo `Model`. Você pode usar o caractere "@" para iniciar uma expressão embutida e mover para antes de `m`:

```
@{  
    var joe = "Joe";  
}  
<input asp-for="@joe" />
```

Gera o seguinte:

```
<input type="text" id="joe" name="joe" value="Joe" />
```

Com propriedades de coleção, `asp-for="CollectionProperty[23].Member"` gera o mesmo nome que `asp-for="CollectionProperty[i].Member"` quando `i` tem o valor `23`.

Quando o ASP.NET Core MVC calcula o valor de `ModelExpression`, ele inspeciona várias fontes, inclusive o `ModelState`. Considere o `<input type="text" asp-for="@Name" />`. O atributo `value` calculado é o primeiro valor não nulo:

- Da entrada de `ModelState` com a chave "Name".
- Do resultado da expressão `Model.Name`.

## Navegando para propriedades filho

Você também pode navegar para propriedades filho usando o caminho da propriedade do modelo de exibição. Considere uma classe de modelo mais complexa que contém uma propriedade `Address` filho.

```
public class AddressViewModel  
{  
    public string AddressLine1 { get; set; }  
}
```

```
public class RegisterAddressViewModel  
{  
    public string Email { get; set; }  
  
    [DataType(DataType.Password)]  
    public string Password { get; set; }  
  
    public AddressViewModel Address { get; set; }  
}
```

Na exibição, associamos a `Address.AddressLine1`:

```
@model RegisterAddressViewModel

<form asp-controller="Demo" asp-action="RegisterAddress" method="post">
    Email: <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    Address: <input asp-for="Address.AddressLine1" /><br />
    <button type="submit">Register</button>
</form>
```

O HTML a seguir é gerado para `Address.AddressLine1`:

```
<input type="text" id="Address_AddressLine1" name="Address.AddressLine1" value="" />
```

## Nomes de expressão e coleções

Exemplo, um modelo que contém uma matriz de `Colors`:

```
public class Person
{
    public List<string> Colors { get; set; }

    public int Age { get; set; }
}
```

O método de ação:

```
public IActionResult Edit(int id, int colorIndex)
{
    ViewData["Index"] = colorIndex;
    return View(GetPerson(id));
}
```

O Razor a seguir mostra como você acessa um elemento `Color` específico:

```
@model Person
 @{
     var index = (int)ViewData["index"];
 }

<form asp-controller="ToDo" asp-action="Edit" method="post">
    @Html.EditorFor(m => m.Colors[index])
    <label asp-for="Age"></label>
    <input asp-for="Age" /><br />
    <button type="submit">Post</button>
</form>
```

O modelo `Views/Shared/EditorTemplates/String.cshtml`:

```
@model string

<label asp-for="@Model"></label>
<input asp-for="@Model" /> <br />
```

Exemplo usando `List<T>`:

```

public class ToDoItem
{
    public string Name { get; set; }

    public bool IsDone { get; set; }
}

```

O Razor a seguir mostra como iterar em uma coleção:

```

@model List<ToDoItem>

<form asp-controller="ToDo" asp-action="Edit" method="post">
    <table>
        <tr> <th>Name</th> <th>Is Done</th> </tr>

        @for (int i = 0; i < Model.Count; i++)
        {
            <tr>
                @Html.EditorFor(model => model[i])
            </tr>
        }

    </table>
    <button type="submit">Save</button>
</form>

```

O modelo *Views/Shared/EditorTemplates/ToDoItem.cshtml*:

```

@model ToDoItem

<td>
    <label asp-for="@Model.Name"></label>
    @Html.DisplayFor(model => model.Name)
</td>
<td>
    <input asp-for="@Model.IsDone" />
</td>

/*
This template replaces the following Razor which evaluates the indexer three times.
<td>
    <label asp-for="@Model[i].Name"></label>
    @Html.DisplayFor(model => model[i].Name)
</td>
<td>
    <input asp-for="@Model[i].IsDone" />
</td>
*/

```

`foreach` deve ser usado, se possível, quando o valor está prestes a ser usado em um contexto equivalente `asp-for` ou `Html.DisplayFor`. Em geral, `for` é melhor do que `foreach` (se o cenário permitir) porque não é necessário alocar um enumerador; no entanto, avaliar um indexador em uma expressão LINQ pode ser caro, o que deve ser minimizado.

#### NOTE

O código de exemplo comentado acima mostra como você substituiria a expressão lambda pelo operador `@` para acessar cada `ToDoItem` na lista.

## Auxiliar de marca de área de texto

O auxiliar de marca `Textarea Tag Helper` é semelhante ao Auxiliar de marca de entrada.

- Gera os atributos `id` e `name`, bem como os atributos de validação de dados do modelo para um elemento `<textarea>`.
- Fornece tipagem forte.
- Alternativa de Auxiliar HTML: `Html.TextAreaFor`

Amostra:

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class DescriptionViewModel
    {
        [MinLength(5)]
        [MaxLength(1024)]
        public string Description { get; set; }
    }
}
```

```
@model DescriptionViewModel

<form asp-controller="Demo" asp-action="RegisterTextArea" method="post">
    <textarea asp-for="Description"></textarea>
    <button type="submit">Test</button>
</form>
```

O HTML a seguir é gerado:

```
<form method="post" action="/Demo/RegisterTextArea">
    <textarea data-val="true"
        data-val-maxlength="The field Description must be a string or array type with a maximum length of
        1024."
        data-val-maxlength-max="1024"
        data-val-minlength="The field Description must be a string or array type with a minimum length of
        5."
        data-val-minlength-min="5"
        id="Description" name="Description">
    </textarea>
    <button type="submit">Test</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## O auxiliar de marca de rótulo

- Gera a legenda do rótulo e o atributo `for` em um elemento para um nome de expressão
- Alternativa de Auxiliar HTML: `Html.LabelFor`

O **Label Tag Helper** fornece os seguintes benefícios em comparação com um elemento de rótulo HTML puro:

- Você obtém automaticamente o valor do rótulo descritivo do atributo `Display`. O nome de exibição desejado pode mudar com o tempo e a combinação do atributo `Display` e do Auxiliar de Marca de Rótulo aplicará `Display` em qualquer lugar em que for usado.
- Menos marcação no código-fonte
- Tipagem forte com a propriedade de modelo.

Amostra:

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class SimpleViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }
    }
}
```

```
@model SimpleViewModel

<form asp-controller="Demo" asp-action="RegisterLabel" method="post">
    <label asp-for="Email"></label>
    <input asp-for="Email" /> <br />
</form>
```

O HTML a seguir é gerado para o elemento `<label>`:

```
<label for="Email">Email Address</label>
```

O Auxiliar de marca de rótulo gerou o valor do atributo `for` de "Email", que é a ID associada ao elemento `<input>`. Auxiliares de marca geram elementos `id` e `for` consistentes para que eles possam ser associados corretamente. A legenda neste exemplo é proveniente do atributo `Display`. Se o modelo não contivesse um atributo `Display`, a legenda seria o nome da propriedade da expressão.

## Os auxiliares de marca de validação

Há dois auxiliares de marca de validação. O **Validation Message Tag Helper** (que exibe uma mensagem de validação para uma única propriedade em seu modelo) e o **Validation Summary Tag Helper** (que exibe um resumo dos erros de validação). O **Input Tag Helper** adiciona atributos de validação do lado do cliente HTML5 para elementos de entrada baseados em atributos de anotação de dados em suas classes de modelo. A validação também é executada no servidor. O Auxiliar de marca de validação exibe essas mensagens de erro quando ocorre um erro de validação.

### O Auxiliar de marca de mensagem de validação

- Adiciona o atributo `data-valmsg-for="property"` **HTML5** ao elemento `span`, que anexa as mensagens de erro de validação no campo de entrada da propriedade do modelo especificado. Quando ocorre um erro de validação do lado do cliente, **jQuery** exibe a mensagem de erro no elemento `<span>`.

- A validação também é feita no servidor. Os clientes poderão ter o JavaScript desabilitado e parte da validação só pode ser feita no lado do servidor.

- Alternativa de Auxiliar HTML: `@Html.ValidationMessageFor`

O `Validation Message Tag Helper` é usado com o atributo `asp-validation-for` em um elemento HTML `span`.

```
<span asp-validation-for="Email"></span>
```

O Auxiliar de marca de mensagem de validação gerará o HTML a seguir:

```
<span class="field-validation-valid"
      data-valmsg-for="Email"
      data-valmsg-replace="true"></span>
```

Geralmente, você usa o `Validation Message Tag Helper` após um Auxiliar de marca `Input` para a mesma propriedade. Fazer isso exibe as mensagens de erro de validação próximo à entrada que causou o erro.

#### NOTE

É necessário ter uma exibição com as referências de script `jQuery` e JavaScript corretas em vigor para a validação do lado do cliente. Consulte [Validação de Modelo](#) para obter mais informações.

Quando ocorre um erro de validação do lado do servidor (por exemplo, quando você tem validação do lado do servidor personalizada ou a validação do lado do cliente está desabilitada), o MVC coloca essa mensagem de erro como o corpo do elemento `<span>`.

```
<span class="field-validation-error" data-valmsg-for="Email"
      data-valmsg-replace="true">
    The Email Address field is required.
</span>
```

#### Auxiliar de marca de resumo de validação

- Tem como alvo elementos `<div>` com o atributo `asp-validation-summary`
- Alternativa de Auxiliar HTML: `@Html.ValidationSummary`

O `Validation Summary Tag Helper` é usado para exibir um resumo das mensagens de validação. O valor do atributo `asp-validation-summary` pode ser qualquer um dos seguintes:

ASP-VALIDATION-SUMMARY	MENSAGENS DE VALIDAÇÃO EXIBIDAS
ValidationSummary.All	Nível da propriedade e do modelo
ValidationSummary.ModelOnly	Modelo
ValidationSummary.None	Nenhum

#### Amostra

No exemplo a seguir, o modelo de dados é decorado com atributos `DataAnnotation`, o que gera mensagens de erro de validação no elemento `<input>`. Quando ocorre um erro de validação, o Auxiliar de marca de validação exibe a mensagem de erro:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}

```

```

@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterValidation" method="post">
    <div asp-validation-summary="ModelOnly"></div>
    Email: <input asp-for="Email" /> <br />
    <span asp-validation-for="Email"></span><br />
    Password: <input asp-for="Password" /><br />
    <span asp-validation-for="Password"></span><br />
    <button type="submit">Register</button>
</form>

```

O código HTML gerado (quando o modelo é válido):

```

<form action="/DemoReg/Register" method="post">
    <div class="validation-summary-valid" data-valmsg-summary="true">
        <ul><li style="display:none"></li></ul></div>
    Email: <input name="Email" id="Email" type="email" value="" data-val-required="The Email field is required." data-val-email="The Email field is not a valid email address." data-val="true"> <br />
    <span class="field-validation-valid" data-valmsg-replace="true" data-valmsg-for="Email"></span><br />
    Password: <input name="Password" id="Password" type="password" data-val-required="The Password field is required." data-val="true"><br />
    <span class="field-validation-valid" data-valmsg-replace="true" data-valmsg-for="Password"></span><br />
    <button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## Auxiliar de Marca de Seleção

- Gera `select` e os elementos `option` associados para as propriedades do modelo.
- Tem uma alternativa de Auxiliar HTML `Html.DropDownListFor` e `Html.ListBoxFor`

O `Select Tag Helper` `asp-for` especifica o nome da propriedade do modelo para o elemento `select` e `asp-items` especifica os elementos `option`. Por exemplo:

```

<select asp-for="Country" asp-items="Model.Countries"></select>

```

Amostra:

```

using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace FormsTagHelper.ViewModels
{
    public class CountryViewModel
    {
        public string Country { get; set; }

        public List<SelectListItem> Countries { get; } = new List<SelectListItem>
        {
            new SelectListItem { Value = "MX", Text = "Mexico" },
            new SelectListItem { Value = "CA", Text = "Canada" },
            new SelectListItem { Value = "US", Text = "USA" },
        };
    }
}

```

O método `Index` inicializa o `CountryViewModel`, define o país selecionado e o transmite para a exibição `Index`.

```

public IActionResult Index()
{
    var model = new CountryViewModel();
    model.Country = "CA";
    return View(model);
}

```

O método `Index` HTTP POST exibe a seleção:

```

[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Index(CountryViewModel model)
{
    if (ModelState.IsValid)
    {
        var msg = model.Country + " selected";
        return RedirectToAction("IndexSuccess", new { message = msg });
    }

    // If we got this far, something failed; redisplay form.
    return View(model);
}

```

A exibição `Index`:

```

@model CountryViewModel

<form asp-controller="Home" asp-action="Index" method="post">
    <select asp-for="Country" asp-items="Model.Countries"></select>
    <br /><button type="submit">Register</button>
</form>

```

Que gera o seguinte HTML (com "CA" selecionado):

```

<form method="post" action="/">
    <select id="Country" name="Country">
        <option value="MX">Mexico</option>
        <option selected="selected" value="CA">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## NOTE

Não é recomendável usar `ViewBag` ou  `ViewData` com o Auxiliar de Marca de Seleção. Um modelo de exibição é mais robusto para fornecer metadados MVC e, geralmente, menos problemático.

O valor do atributo `asp-for` é um caso especial e não requer um prefixo `Model`, os outros atributos do Auxiliar de marca requerem (como `asp-items`)

```
<select asp-for="Country" asp-items="Model.Countries"></select>
```

## Associação de enumeração

Geralmente, é conveniente usar `<select>` com uma propriedade `enum` e gerar os elementos `SelectListItem` dos valores `enum`.

Amostra:

```

public class CountryEnumViewModel
{
    public CountryEnum EnumCountry { get; set; }
}

```

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public enum CountryEnum
    {
        [Display(Name = "United Mexican States")]
        Mexico,
        [Display(Name = "United States of America")]
        USA,
        Canada,
        France,
        Germany,
        Spain
    }
}

```

O método `GetEnumSelectList` gera um objeto `selectList` para uma enumeração.

```

@model CountryEnumViewModel

<form asp-controller="Home" asp-action="IndexEnum" method="post">
    <select asp-for="EnumCountry"
            asp-items="Html.GetEnumSelectList<CountryEnum>()">
    </select>
    <br /><button type="submit">Register</button>
</form>

```

É possível decorar sua lista de enumeradores com o atributo `Display` para obter uma interface do usuário mais rica:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public enum CountryEnum
    {
        [Display(Name = "United Mexican States")]
        Mexico,
        [Display(Name = "United States of America")]
        USA,
        Canada,
        France,
        Germany,
        Spain
    }
}

```

O HTML a seguir é gerado:

```

<form method="post" action="/Home/IndexEnum">
    <select data-val="true" data-val-required="The EnumCountry field is required."
            id="EnumCountry" name="EnumCountry">
        <option value="0">United Mexican States</option>
        <option value="1">United States of America</option>
        <option value="2">Canada</option>
        <option value="3">France</option>
        <option value="4">Germany</option>
        <option selected="selected" value="5">Spain</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

## Grupo de opções

O elemento HTML `<optgroup>` é gerado quando o modelo de exibição contém um ou mais objetos `SelectListGroup`.

O `CountryViewModelGroup` agrupa os elementos `SelectListItem` nos grupos "América do Norte" e "Europa":

```

public class CountryViewModelGroup
{
    public CountryViewModelGroup()
    {
        var NorthAmericaGroup = new SelectListGroup { Name = "North America" };
        var EuropeGroup = new SelectListGroup { Name = "Europe" };

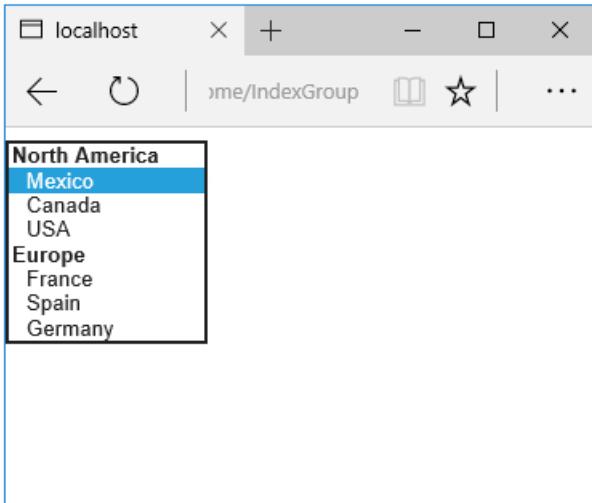
        Countries = new List<SelectListItem>
        {
            new SelectListItem
            {
                Value = "MEX",
                Text = "Mexico",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "CAN",
                Text = "Canada",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "US",
                Text = "USA",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "FR",
                Text = "France",
                Group = EuropeGroup
            },
            new SelectListItem
            {
                Value = "ES",
                Text = "Spain",
                Group = EuropeGroup
            },
            new SelectListItem
            {
                Value = "DE",
                Text = "Germany",
                Group = EuropeGroup
            }
        };
    }

    public string Country { get; set; }

    public List<SelectListItem> Countries { get; }
}

```

Os dois grupos são mostrados abaixo:



O HTML gerado:

```
<form method="post" action="/Home/IndexGroup">
    <select id="Country" name="Country">
        <optgroup label="North America">
            <option value="MEX">Mexico</option>
            <option value="CAN">Canada</option>
            <option value="US">USA</option>
        </optgroup>
        <optgroup label="Europe">
            <option value="FR">France</option>
            <option value="ES">Spain</option>
            <option value="DE">Germany</option>
        </optgroup>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="removed for brevity" />
</form>
```

## Seleção múltipla

O Auxiliar de Marca de Seleção gerará automaticamente o atributo `multiple = "multiple"` se a propriedade especificada no atributo `asp-for` for um `IEnumerable`. Por exemplo, considerando o seguinte modelo:

```
using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace FormsTagHelper.ViewModels
{
    public class CountryViewModel : IEnumerable
    {
        public IEnumerable<string> CountryCodes { get; set; }

        public List<SelectListItem> Countries { get; } = new List<SelectListItem>
        {
            new SelectListItem { Value = "MX", Text = "Mexico" },
            new SelectListItem { Value = "CA", Text = "Canada" },
            new SelectListItem { Value = "US", Text = "USA" },
            new SelectListItem { Value = "FR", Text = "France" },
            new SelectListItem { Value = "ES", Text = "Spain" },
            new SelectListItem { Value = "DE", Text = "Germany" }
        };
    }
}
```

Com a seguinte exibição:

```
@model CountryViewModelIEnumarable

<form asp-controller="Home" asp-action="IndexMultiSelect" method="post">
    <select asp-for="CountryCodes" asp-items="Model.Countries"></select>
    <br /><button type="submit">Register</button>
</form>
```

Gera o seguinte HTML:

```
<form method="post" action="/Home/IndexMultiSelect">
    <select id="CountryCodes"
        multiple="multiple"
        name="CountryCodes"><option value="MX">Mexico</option>
    <option value="CA">Canada</option>
    <option value="US">USA</option>
    <option value="FR">France</option>
    <option value="ES">Spain</option>
    <option value="DE">Germany</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## Nenhuma seleção

Se acabar usando a opção "não especificado" em várias páginas, você poderá criar um modelo para eliminar o HTML de repetição:

```
@model CountryViewModel

<form asp-controller="Home" asp-action="IndexEmpty" method="post">
    @Html.EditorForModel()
    <br /><button type="submit">Register</button>
</form>
```

O modelo *Views/Shared/EditorTemplates/CountryViewModel.cshtml*:

```
@model CountryViewModel

<select asp-for="Country" asp-items="Model.Countries">
    <option value="">--none--</option>
</select>
```

O acréscimo de elementos HTML `<option>` não está limitado ao caso de *Nenhuma seleção*. Por exemplo, o seguinte método de ação e exibição gerarão HTML semelhante ao código acima:

```
public IActionResult IndexOption(int id)
{
    var model = new CountryViewModel();
    model.Country = "CA";
    return View(model);
}
```

```
@model CountryViewModel

<form asp-controller="Home" asp-action="IndexEmpty" method="post">
    <select asp-for="Country">
        <option value="">&lt;none&gt;</option>
        <option value="MX">Mexico</option>
        <option value="CA">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
</form>
```

O elemento `<option>` correto será selecionado (contém o atributo `selected="selected"`) dependendo do valor atual de `Country`.

```
<form method="post" action="/Home/IndexEmpty">
    <select id="Country" name="Country">
        <option value="">&lt;none&gt;</option>
        <option value="MX">Mexico</option>
        <option value="CA" selected="selected">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## Recursos adicionais

- [Auxiliares de Marca no ASP.NET Core](#)
- [Elemento de formulário HTML](#)
- [Token de verificação de solicitação](#)
- [Model binding no ASP.NET Core](#)
- [Validação de modelo no ASP.NET Core MVC](#)
- [Interface IAttributeAdapter](#)
- [Snippets de código para este documento](#)

# Auxiliares de marca em formulários no ASP.NET Core

14/01/2019 • 28 minutes to read • [Edit Online](#)

Por Rick Anderson, Dave Paquette e Jerrie Pelser

Este documento demonstra como é o trabalho com Formulários e os elementos HTML usados comumente em um Formulário. O elemento HTML [Formulário](#) fornece o mecanismo primário que os aplicativos Web usam para postar dados para o servidor. A maior parte deste documento descreve os [Auxiliares de marca](#) e como eles podem ajudar você a criar formulários HTML robustos de forma produtiva. É recomendável que você leia [Introdução ao auxiliares de marca](#) antes de ler este documento.

Em muitos casos, os Auxiliares HTML fornecem uma abordagem alternativa a um Auxiliar de Marca específico, mas é importante reconhecer que os Auxiliares de Marca não substituem os Auxiliares HTML e que não há um Auxiliar de Marca para cada Auxiliar HTML. Quando existe um Auxiliar HTML alternativo, ele é mencionado.

## O Auxiliar de marca de formulário

O Auxiliar de marca de [formulário](#):

- Gera o valor do atributo HTML `<FORM> action` para uma ação do controlador MVC ou uma rota nomeada
- Gera um [Token de verificação de solicitação](#) oculto para evitar a falsificação de solicitações entre sites (quando usado com o atributo `[ValidateAntiForgeryToken]` no método de ação HTTP Post)
- Fornece o atributo `asp-route-<Parameter Name>`, em que `<Parameter Name>` é adicionado aos valores de rota. Os parâmetros `routeValues` para `Html.BeginForm` e `Html.BeginRouteForm` fornecem funcionalidade semelhante.
- Tem uma alternativa de Auxiliar HTML `Html.BeginForm` e `Html.BeginRouteForm`

Amostra:

```
<form asp-controller="Demo" asp-action="Register" method="post">
    <!-- Input and Submit elements -->
</form>
```

O Auxiliar de marca de formulário acima gera o HTML a seguir:

```
<form method="post" action="/Demo/Register">
    <!-- Input and Submit elements -->
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

O tempo de execução do MVC gera o valor do atributo `action` dos atributos `asp-controller` e `asp-action` do Auxiliar de marca de formulário. O Auxiliar de marca de formulário também gera um [Token de verificação de solicitação](#) oculto para evitar a falsificação de solicitações entre sites (quando

usado com o atributo `[ValidateAntiForgeryToken]` no método de ação HTTP Post). É difícil proteger um Formulário HTML puro contra falsificação de solicitações entre sites e o Auxiliar de marca de formulário fornece este serviço para você.

### Usando uma rota nomeada

O atributo do Auxiliar de Marca `asp-route` também pode gerar a marcação para o atributo HTML `action`. Um aplicativo com uma `rota` chamada `register` poderia usar a seguinte marcação para a página de registro:

```
<form asp-route="register" method="post">
    <!-- Input and Submit elements -->
</form>
```

Muitas das exibições na pasta *Modos de Exibição/Conta* (gerada quando você cria um novo aplicativo Web com *Contas de usuário individuais*) contêm o atributo `asp-route-returnurl`:

```
<form asp-controller="Account" asp-action="Login"
      asp-route-returnurl="@ViewData["ReturnUrl"]"
      method="post" class="form-horizontal" role="form">
```

#### NOTE

Com os modelos internos, `returnUrl` só é preenchido automaticamente quando você tenta acessar um recurso autorizado, mas não está autenticado ou autorizado. Quando você tenta fazer um acesso não autorizado, o middleware de segurança o redireciona para a página de logon com o `returnUrl` definido.

## O auxiliar de marca de entrada

O Auxiliar de marca de entrada associa um elemento HTML `<input>` a uma expressão de modelo em sua exibição do Razor.

Sintaxe:

```
<input asp-for="<Expression Name>" />
```

O auxiliar de marca de entrada:

- Gera os atributos HTML `id` e `name` para o nome da expressão especificada no atributo `asp-for`. `asp-for="Property1.Property2"` equivale a `m => m.Property1.Property2`. O nome da expressão é o que é usado para o valor do atributo `asp-for`. Consulte a seção [Nomes de expressão](#) para obter informações adicionais.
- Define o valor do atributo HTML `type` com base nos atributos de tipo de modelo e [anotação de dados](#) aplicados à propriedade de modelo
- O valor do atributo HTML `type` não será substituído quando um for especificado
- Gera atributos de validação [HTML5](#) de atributos de [anotação de dados](#) aplicados a propriedades de modelo
- Tem uma sobreposição de recursos de Auxiliar HTML com `Html.TextBoxFor` e `Html.EditorFor`. Consulte a seção **Alternativas de Auxiliar HTML ao Auxiliar de marca de entrada** para obter detalhes.

- Fornece tipagem forte. Se o nome da propriedade for alterado e você não atualizar o Auxiliar de marca, você verá um erro semelhante ao seguinte:

An error occurred during the compilation of a resource required to process this request. Please review the following specific error details and modify your source code appropriately.

Type expected  
'RegisterViewModel' does not contain a definition for 'Email' and no extension method 'Email' accepting a first argument of type 'RegisterViewModel' could be found (are you missing a using directive or an assembly reference?)

O Auxiliar de marca `Input` define o atributo HTML `type` com base no tipo .NET. A tabela a seguir lista alguns tipos .NET comuns e o tipo HTML gerado (não estão listados todos os tipos .NET).

TIPO .NET	TIPO DE ENTRADA
Bool	type="checkbox"
Cadeia de Caracteres	type="text"
DateTime	type="datetime-local"
Byte	type="number"
int	type="number"
Single e Double	type="number"

A tabela a seguir mostra alguns atributos de [anotações de dados](#) comuns que o auxiliar de marca de entrada mapeará para tipos de entrada específicos (não são listados todos os atributos de validação):

ATRIBUTO	TIPO DE ENTRADA
[EmailAddress]	type="email"
[Url]	type="url"
[HiddenInput]	type="hidden"
[Phone]	type="tel"
[DataType(DataType.Password)]	type="password"
[DataType(DataType.Date)]	type="date"
[DataType(DataType.Time)]	type="time"

Amostra:

```

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}

```

```

@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterInput" method="post">
    Email: <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    <button type="submit">Register</button>
</form>

```

O código acima gera o seguinte HTML:

```

<form method="post" action="/Demo/RegisterInput">
    Email:
    <input type="email" data-val="true"
           data-val-email="The Email Address field is not a valid email address."
           data-val-required="The Email Address field is required."
           id="Email" name="Email" value="" /> <br>
    Password:
    <input type="password" data-val="true"
           data-val-required="The Password field is required."
           id="Password" name="Password" /><br>
    <button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>

```

As anotações de dados aplicadas às propriedades `Email` e `Password` geram metadados no modelo. O Auxiliar de marca de entrada consome os metadados do modelo e produz atributos [HTML5 data-val-\\*](#) (consulte [Validação de modelo](#)). Esses atributos descrevem os validadores a serem anexados aos campos de entrada. Isso fornece validação de [jQuery](#) e [HTML5](#) discreto. Os atributos discretos têm o formato `data-val-rule="Error Message"`, em que a regra é o nome da regra de validação (como `data-val-required`, `data-val-email`, `data-val-maxlength` etc.) Se uma mensagem de erro for fornecida no atributo, ela será exibida como o valor para do atributo `data-val-rule`. Também há atributos do formulário `data-val-rulenName-argumentName="argumentValue"` que fornecem detalhes adicionais sobre a regra, por exemplo, `data-val-maxlength-max="1024"`.

## Alternativas de Auxiliar HTML ao Auxiliar de marca de entrada

`Html.TextBox`, `Html.TextBoxFor`, `Html.Editor` e `Html.EditorFor` têm recursos que se sobrepõem aos di Auxiliar de marca de entrada. O Auxiliar de marca de entrada define automaticamente o atributo `type`; `Html.TextBox` e `Html.TextBoxFor` não o fazem. `Html.Editor` e `Html.EditorFor` manipulam coleções, objetos complexos e modelos; o Auxiliar de marca de entrada não o faz. O Auxiliar de marca de entrada, `Html.EditorFor` e `Html.TextBoxFor` são fortemente tipados (eles usam expressões

lambda); `Html.TextBox` e `Html.Editor` não usam (eles usam nomes de expressão).

## HtmlAttributes

`@Html.Editor()` e `@Html.EditorFor()` usam uma entrada `ViewDataDictionary` especial chamada `htmlAttributes` ao executar seus modelos padrão. Esse comportamento pode ser aumentado usando parâmetros `additionalViewData`. A chave "htmlAttributes" diferencia maiúsculas de minúsculas. A chave "htmlAttributes" é tratada de forma semelhante ao objeto `htmlAttributes` passado para auxiliares de entrada como `@Html.TextBox()`.

```
@Html.EditorFor(model => model.YourProperty,  
new { htmlAttributes = new { @class="myCssClass", style="Width:100px" } })
```

## Nomes de expressão

O valor do atributo `asp-for` é um `ModelExpression` e o lado direito de uma expressão lambda. Portanto, `asp-for="Property1"` se torna `m => m.Property1` no código gerado e é por isso você não precisa colocar o prefixo `Model`. Você pode usar o caractere "@" para iniciar uma expressão embutida e mover para antes de `m`:

```
@{  
    var joe = "Joe";  
}  
<input asp-for="@joe" />
```

Gera o seguinte:

```
<input type="text" id="joe" name="joe" value="Joe" />
```

Com propriedades de coleção, `asp-for="CollectionProperty[23].Member"` gera o mesmo nome que `asp-for="CollectionProperty[i].Member"` quando `i` tem o valor `23`.

Quando o ASP.NET Core MVC calcula o valor de `ModelExpression`, ele inspeciona várias fontes, inclusive o `ModelState`. Considere o `<input type="text" asp-for="@Name" />`. O atributo `value` calculado é o primeiro valor não nulo:

- Da entrada de `ModelState` com a chave "Name".
- Do resultado da expressão `Model.Name`.

## Navegando para propriedades filho

Você também pode navegar para propriedades filho usando o caminho da propriedade do modelo de exibição. Considere uma classe de modelo mais complexa que contém uma propriedade `Address` filho.

```
public class AddressViewModel  
{  
    public string AddressLine1 { get; set; }  
}
```

```

public class RegisterAddressViewModel
{
    public string Email { get; set; }

    [DataType(DataType.Password)]
    public string Password { get; set; }

    public AddressViewModel Address { get; set; }
}

```

Na exibição, associamos a `Address.AddressLine1`:

```

@model RegisterAddressViewModel

<form asp-controller="Demo" asp-action="RegisterAddress" method="post">
    Email: <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    Address: <input asp-for="Address.AddressLine1" /><br />
    <button type="submit">Register</button>
</form>

```

O HTML a seguir é gerado para `Address.AddressLine1`:

```
<input type="text" id="Address_AddressLine1" name="Address.AddressLine1" value="" />
```

## Nomes de expressão e coleções

Exemplo, um modelo que contém uma matriz de `Colors`:

```

public class Person
{
    public List<string> Colors { get; set; }

    public int Age { get; set; }
}

```

O método de ação:

```

public IActionResult Edit(int id, int colorIndex)
{
    ViewData["Index"] = colorIndex;
    return View(GetPerson(id));
}

```

O Razor a seguir mostra como você acessa um elemento `Color` específico:

```

@model Person
 @{
    var index = (int)ViewData["index"];
}

<form asp-controller="ToDo" asp-action="Edit" method="post">
    @Html.EditorFor(m => m.Colors[index])
    <label asp-for="Age"></label>
    <input asp-for="Age" /><br />
    <button type="submit">Post</button>
</form>

```

O modelo *Views/Shared/EditorTemplates/String.cshtml*:

```
@model string

<label asp-for="@Model"></label>
<input asp-for="@Model" /> <br />
```

Exemplo usando `List<T>`:

```
public class ToDoItem
{
    public string Name { get; set; }

    public bool IsDone { get; set; }
}
```

O Razor a seguir mostra como iterar em uma coleção:

```
@model List<ToDoItem>

<form asp-controller="ToDo" asp-action="Edit" method="post">
    <table>
        <tr> <th>Name</th> <th>Is Done</th> </tr>

        @for (int i = 0; i < Model.Count; i++)
        {
            <tr>
                @Html.EditorFor(model => model[i])
            </tr>
        }

    </table>
    <button type="submit">Save</button>
</form>
```

O modelo *Views/Shared/EditorTemplates/ToDoItem.cshtml*:

```
@model ToDoItem

<td>
    <label asp-for="@Model.Name"></label>
    @Html.DisplayFor(model => model.Name)
</td>
<td>
    <input asp-for="@Model.IsDone" />
</td>

@*
    This template replaces the following Razor which evaluates the indexer three times.
<td>
    <label asp-for="@Model[i].Name"></label>
    @Html.DisplayFor(model => model[i].Name)
</td>
<td>
    <input asp-for="@Model[i].IsDone" />
</td>
*@

    
```

`foreach` deve ser usado, se possível, quando o valor está prestes a ser usado em um contexto equivalente `asp-for` ou `Html.DisplayFor`. Em geral, `for` é melhor do que `foreach` (se o cenário

permitir) porque não é necessário alocar um enumerador; no entanto, avaliar um indexador em uma expressão LINQ pode ser caro, o que deve ser minimizado.

#### NOTE

O código de exemplo comentado acima mostra como você substituiria a expressão lambda pelo operador `@` para acessar cada `ToDoItem` na lista.

## Auxiliar de marca de área de texto

O auxiliar de marca `Textarea Tag Helper` é semelhante ao Auxiliar de marca de entrada.

- Gera os atributos `id` e `name`, bem como os atributos de validação de dados do modelo para um elemento `<textarea>`.
- Fornece tipagem forte.
- Alternativa de Auxiliar HTML: `Html.TextAreaFor`

Amostra:

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class DescriptionViewModel
    {
        [MinLength(5)]
        [MaxLength(1024)]
        public string Description { get; set; }
    }
}
```

```
@model DescriptionViewModel

<form asp-controller="Demo" asp-action="RegisterTextArea" method="post">
    <textarea asp-for="Description"></textarea>
    <button type="submit">Test</button>
</form>
```

O HTML a seguir é gerado:

```
<form method="post" action="/Demo/RegisterTextArea">
    <textarea data-val="true"
        data-val-maxlength="The field Description must be a string or array type with a maximum length
        of 1024."
        data-val-maxlength-max="1024"
        data-val-minlength="The field Description must be a string or array type with a minimum length
        of 5."
        data-val-minlength-min="5"
        id="Description" name="Description">
    </textarea>
    <button type="submit">Test</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## O auxiliar de marca de rótulo

- Gera a legenda do rótulo e o atributo `for` em um elemento para um nome de expressão
- Alternativa de Auxiliar HTML: `Html.LabelFor`.

O `Label Tag Helper` fornece os seguintes benefícios em comparação com um elemento de rótulo HTML puro:

- Você obtém automaticamente o valor do rótulo descritivo do atributo `Display`. O nome de exibição desejado pode mudar com o tempo e a combinação do atributo `Display` e do Auxiliar de Marca de Rótulo aplicará `Display` em qualquer lugar em que for usado.
- Menos marcação no código-fonte
- Tipagem forte com a propriedade de modelo.

Amostra:

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class SimpleViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }
    }
}
```

```
@model SimpleViewModel

<form asp-controller="Demo" asp-action="RegisterLabel" method="post">
    <label asp-for="Email"></label>
    <input asp-for="Email" /> <br />
</form>
```

O HTML a seguir é gerado para o elemento `<label>`:

```
<label for="Email">Email Address</label>
```

O Auxiliar de marca de rótulo gerou o valor do atributo `for` de "Email", que é a ID associada ao elemento `<input>`. Auxiliares de marca geram elementos `id` e `for` consistentes para que eles possam ser associados corretamente. A legenda neste exemplo é proveniente do atributo `Display`. Se o modelo não contivesse um atributo `Display`, a legenda seria o nome da propriedade da expressão.

## Os auxiliares de marca de validação

Há dois auxiliares de marca de validação. O `Validation Message Tag Helper` (que exibe uma mensagem de validação para uma única propriedade em seu modelo) e o `Validation Summary Tag Helper` (que exibe um resumo dos erros de validação). O `Input Tag Helper` adiciona atributos de validação do lado do cliente HTML5 para elementos de entrada baseados em atributos de anotação de dados em suas classes de modelo. A validação também é executada no

servidor. O Auxiliar de marca de validação exibe essas mensagens de erro quando ocorre um erro de validação.

### O Auxiliar de marca de mensagem de validação

- Adiciona o atributo `data-valmsg-for="property"` [HTML5](#) ao elemento `span`, que anexa as mensagens de erro de validação no campo de entrada da propriedade do modelo especificado. Quando ocorre um erro de validação do lado do cliente, [jQuery](#) exibe a mensagem de erro no elemento `<span>`.
- A validação também é feita no servidor. Os clientes poderão ter o JavaScript desabilitado e parte da validação só pode ser feita no lado do servidor.
- Alternativa de Auxiliar HTML: `@Html.ValidationMessageFor`

O [Validation Message Tag Helper](#) é usado com o atributo `asp-validation-for` em um elemento HTML `span`.

```
<span asp-validation-for="Email"></span>
```

O Auxiliar de marca de mensagem de validação gerará o HTML a seguir:

```
<span class="field-validation-valid"
      data-valmsg-for="Email"
      data-valmsg-replace="true"></span>
```

Geralmente, você usa o [Validation Message Tag Helper](#) após um Auxiliar de marca [Input](#) para a mesma propriedade. Fazer isso exibe as mensagens de erro de validação próximo à entrada que causou o erro.

#### NOTE

É necessário ter uma exibição com as referências de script [jQuery](#) e JavaScript corretas em vigor para a validação do lado do cliente. Consulte [Validação de Modelo](#) para obter mais informações.

Quando ocorre um erro de validação do lado do servidor (por exemplo, quando você tem validação do lado do servidor personalizada ou a validação do lado do cliente está desabilitada), o MVC coloca essa mensagem de erro como o corpo do elemento `<span>`.

```
<span class="field-validation-error" data-valmsg-for="Email"
      data-valmsg-replace="true">
    The Email Address field is required.
</span>
```

### Auxiliar de marca de resumo de validação

- Tem como alvo elementos `<div>` com o atributo `asp-validation-summary`
- Alternativa de Auxiliar HTML: `@Html.ValidationSummary`

O [Validation Summary Tag Helper](#) é usado para exibir um resumo das mensagens de validação. O valor do atributo `asp-validation-summary` pode ser qualquer um dos seguintes:

ASP-VALIDATION-SUMMARY	MENSAGENS DE VALIDAÇÃO EXIBIDAS
ValidationSummary.All	Nível da propriedade e do modelo
ValidationSummary.ModelOnly	Modelo
ValidationSummary.None	Nenhum

### Amostra

No exemplo a seguir, o modelo de dados é decorado com atributos `DataAnnotation`, o que gera mensagens de erro de validação no elemento `<input>`. Quando ocorre um erro de validação, o Auxiliar de marca de validação exibe a mensagem de erro:

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}
```

```
@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterValidation" method="post">
    <div asp-validation-summary="ModelOnly"></div>
    Email: <input asp-for="Email" /> <br />
    <span asp-validation-for="Email"></span><br />
    Password: <input asp-for="Password" /><br />
    <span asp-validation-for="Password"></span><br />
    <button type="submit">Register</button>
</form>
```

O código HTML gerado (quando o modelo é válido):

```
<form action="/DemoReg/Register" method="post">
    <div class="validation-summary-valid" data-valmsg-summary="true">
        <ul><li style="display:none"></li></ul>
    </div>
    Email: <input name="Email" id="Email" type="email" value="" data-val-required="The Email field is required." data-val-email="The Email field is not a valid email address." data-val="true"> <br>
    <span class="field-validation-valid" data-valmsg-replace="true" data-valmsg-for="Email"></span><br />
    Password: <input name="Password" id="Password" type="password" data-val-required="The Password field is required." data-val="true"><br />
    <span class="field-validation-valid" data-valmsg-replace="true" data-valmsg-for="Password"></span><br />
    <button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## Auxiliar de Marca de Seleção

- Gera `select` e os elementos `option` associados para as propriedades do modelo.
- Tem uma alternativa de Auxiliar HTML `Html.DropDownListFor` e `Html.ListBoxFor`

O `Select Tag Helper` `asp-for` especifica o nome da propriedade do modelo para o elemento `select` `asp-items` especifica os elementos `option`. Por exemplo:

```
<select asp-for="Country" asp-items="Model.Countries"></select>
```

Amostra:

```
using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace FormsTagHelper.ViewModels
{
    public class CountryViewModel
    {
        public string Country { get; set; }

        public List<SelectListItem> Countries { get; } = new List<SelectListItem>
        {
            new SelectListItem { Value = "MX", Text = "Mexico" },
            new SelectListItem { Value = "CA", Text = "Canada" },
            new SelectListItem { Value = "US", Text = "USA" },
        };
    }
}
```

O método `Index` inicializa o `CountryViewModel`, define o país selecionado e o transmite para a exibição `Index`.

```
public IActionResult Index()
{
    var model = new CountryViewModel();
    model.Country = "CA";
    return View(model);
}
```

O método `Index` HTTP POST exibe a seleção:

```
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Index(CountryViewModel model)
{
    if (ModelState.IsValid)
    {
        var msg = model.Country + " selected";
        return RedirectToAction("IndexSuccess", new { message = msg });
    }

    // If we got this far, something failed; redisplay form.
    return View(model);
}
```

A exibição `Index`:

```
@model CountryViewModel

<form asp-controller="Home" asp-action="Index" method="post">
    <select asp-for="Country" asp-items="Model.Countries"></select>
    <br /><button type="submit">Register</button>
</form>
```

Que gera o seguinte HTML (com "CA" selecionado):

```
<form method="post" action="/">
    <select id="Country" name="Country">
        <option value="MX">Mexico</option>
        <option selected="selected" value="CA">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

#### NOTE

Não é recomendável usar `ViewBag` ou `ViewData` com o Auxiliar de Marca de Seleção. Um modelo de exibição é mais robusto para fornecer metadados MVC e, geralmente, menos problemático.

O valor do atributo `asp-for` é um caso especial e não requer um prefixo `Model`, os outros atributos do Auxiliar de marca requerem (como `asp-items`)

```
<select asp-for="Country" asp-items="Model.Countries"></select>
```

#### Associação de enumeração

Geralmente, é conveniente usar `<select>` com uma propriedade `enum` e gerar os elementos `SelectListItem` dos valores `enum`.

Amostra:

```
public class CountryEnumViewModel
{
    public CountryEnum EnumCountry { get; set; }
}
```

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public enum CountryEnum
    {
        [Display(Name = "United Mexican States")]
        Mexico,
        [Display(Name = "United States of America")]
        USA,
        Canada,
        France,
        Germany,
        Spain
    }
}
```

O método `GetEnumSelectList` gera um objeto `SelectList` para uma enumeração.

```
@model CountryEnumViewModel

<form asp-controller="Home" asp-action="IndexEnum" method="post">
    <select asp-for="EnumCountry"
            asp-items="Html.GetEnumSelectList<CountryEnum>()">
    </select>
    <br /><button type="submit">Register</button>
</form>
```

É possível decorar sua lista de enumeradores com o atributo `Display` para obter uma interface do usuário mais rica:

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public enum CountryEnum
    {
        [Display(Name = "United Mexican States")]
        Mexico,
        [Display(Name = "United States of America")]
        USA,
        Canada,
        France,
        Germany,
        Spain
    }
}
```

O HTML a seguir é gerado:

```
<form method="post" action="/Home/IndexEnum">
    <select data-val="true" data-val-required="The EnumCountry field is required."
            id="EnumCountry" name="EnumCountry">
        <option value="0">United Mexican States</option>
        <option value="1">United States of America</option>
        <option value="2">Canada</option>
        <option value="3">France</option>
        <option value="4">Germany</option>
        <option selected="selected" value="5">Spain</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## Grupo de opções

O elemento HTML `<optgroup>` é gerado quando o modelo de exibição contém um ou mais objetos `SelectListGroup`.

O `CountryViewModelGroup` agrupa os elementos `SelectListItem` nos grupos "América do Norte" e "Europa":

```

public class CountryViewModelGroup
{
    public CountryViewModelGroup()
    {
        var NorthAmericaGroup = new SelectListGroup { Name = "North America" };
        var EuropeGroup = new SelectListGroup { Name = "Europe" };

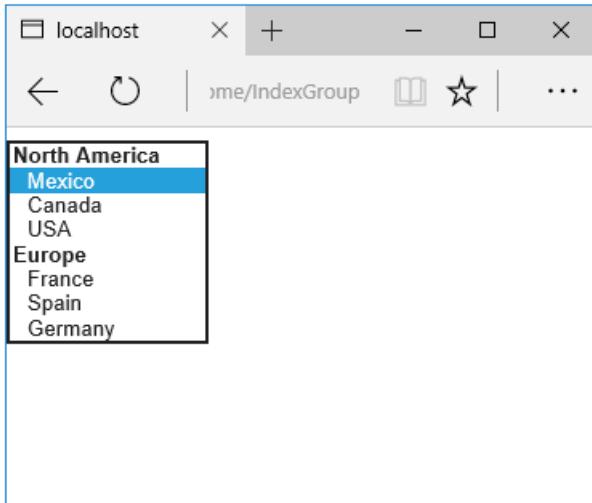
        Countries = new List<SelectListItem>
        {
            new SelectListItem
            {
                Value = "MEX",
                Text = "Mexico",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "CAN",
                Text = "Canada",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "US",
                Text = "USA",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "FR",
                Text = "France",
                Group = EuropeGroup
            },
            new SelectListItem
            {
                Value = "ES",
                Text = "Spain",
                Group = EuropeGroup
            },
            new SelectListItem
            {
                Value = "DE",
                Text = "Germany",
                Group = EuropeGroup
            }
        };
    }

    public string Country { get; set; }

    public List<SelectListItem> Countries { get; }
}

```

Os dois grupos são mostrados abaixo:



O HTML gerado:

```
<form method="post" action="/Home/IndexGroup">
    <select id="Country" name="Country">
        <optgroup label="North America">
            <option value="MEX">Mexico</option>
            <option value="CAN">Canada</option>
            <option value="US">USA</option>
        </optgroup>
        <optgroup label="Europe">
            <option value="FR">France</option>
            <option value="ES">Spain</option>
            <option value="DE">Germany</option>
        </optgroup>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

### Seleção múltipla

O Auxiliar de Marca de Seleção gerará automaticamente o atributo `multiple = "multiple"` se a propriedade especificada no atributo `asp-for` for um `IEnumerable`. Por exemplo, considerando o seguinte modelo:

```
using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace FormsTagHelper.ViewModels
{
    public class CountryViewModel : IEnumerable
    {
        public IEnumerable<string> CountryCodes { get; set; }

        public List<SelectListItem> Countries { get; } = new List<SelectListItem>
        {
            new SelectListItem { Value = "MX", Text = "Mexico" },
            new SelectListItem { Value = "CA", Text = "Canada" },
            new SelectListItem { Value = "US", Text = "USA" },
            new SelectListItem { Value = "FR", Text = "France" },
            new SelectListItem { Value = "ES", Text = "Spain" },
            new SelectListItem { Value = "DE", Text = "Germany" }
        };
    }
}
```

Com a seguinte exibição:

```
@model CountryViewModelIEnumerable

<form asp-controller="Home" asp-action="IndexMultiSelect" method="post">
    <select asp-for="CountryCodes" asp-items="Model.Countries"></select>
    <br /><button type="submit">Register</button>
</form>
```

Gera o seguinte HTML:

```
<form method="post" action="/Home/IndexMultiSelect">
    <select id="CountryCodes"
        multiple="multiple"
        name="CountryCodes"><option value="MX">Mexico</option>
    <option value="CA">Canada</option>
    <option value="US">USA</option>
    <option value="FR">France</option>
    <option value="ES">Spain</option>
    <option value="DE">Germany</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

### Nenhuma seleção

Se acabar usando a opção "não especificado" em várias páginas, você poderá criar um modelo para eliminar o HTML de repetição:

```
@model CountryViewModel

<form asp-controller="Home" asp-action="IndexEmpty" method="post">
    @Html.EditorForModel()
    <br /><button type="submit">Register</button>
</form>
```

O modelo *Views/Shared/EditorTemplates/CountryViewModel.cshtml*:

```
@model CountryViewModel

<select asp-for="Country" asp-items="Model.Countries">
    <option value="">--none--</option>
</select>
```

O acréscimo de elementos HTML `<option>` não está limitado ao caso de *Nenhuma seleção*. Por exemplo, o seguinte método de ação e exibição gerarão HTML semelhante ao código acima:

```
public IActionResult IndexOption(int id)
{
    var model = new CountryViewModel();
    model.Country = "CA";
    return View(model);
}
```

```
@model CountryViewModel

<form asp-controller="Home" asp-action="IndexEmpty" method="post">
    <select asp-for="Country">
        <option value="">&lt;none&gt;</option>
        <option value="MX">Mexico</option>
        <option value="CA">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
</form>
```

O elemento `<option>` correto será selecionado (contém o atributo `selected="selected"`) dependendo do valor atual de `Country`.

```
<form method="post" action="/Home/IndexEmpty">
    <select id="Country" name="Country">
        <option value="">&lt;none&gt;</option>
        <option value="MX">Mexico</option>
        <option value="CA" selected="selected">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>" />
</form>
```

## Recursos adicionais

- [Auxiliares de Marca no ASP.NET Core](#)
- [Elemento de formulário HTML](#)
- [Token de verificação de solicitação](#)
- [Model binding no ASP.NET Core](#)
- [Validação de modelo no ASP.NET Core MVC](#)
- [Interface IAttributeAdapter](#)
- [Snippets de código para este documento](#)

# Layout no ASP.NET Core

29/10/2018 • 11 minutes to read • [Edit Online](#)

Por [Steve Smith](#) e [Dave Brock](#)

Páginas e exibições com frequência compartilham elementos visuais e programáticos. Este artigo demonstra como:

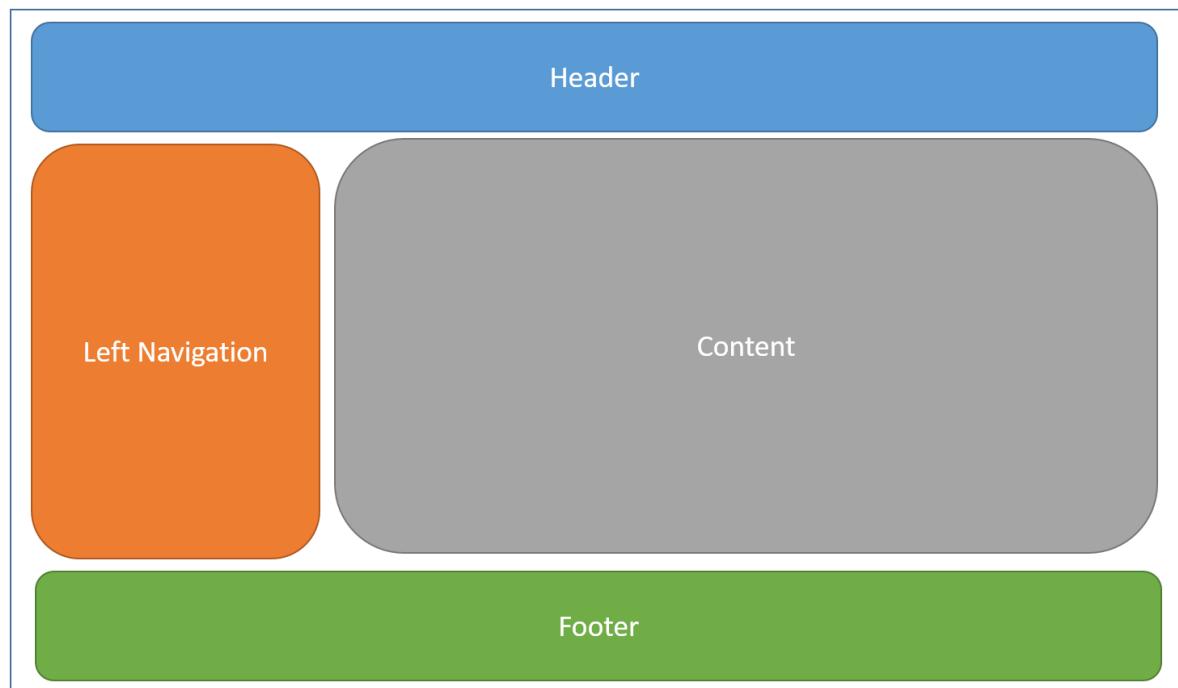
- Usar layouts comuns.
- Compartilhar diretivas.
- Executar o código comum antes de renderizar páginas ou modos de exibição.

Este documento discute os layouts para as duas abordagens diferentes ao ASP.NET Core MVC: Razor Pages e controladores com exibições. Para este tópico, as diferenças são mínimas:

- Razor Pages estão na pasta *Pages*.
- Controladores com exibições usam uma pasta *Views* pasta exibições.

## O que é um layout

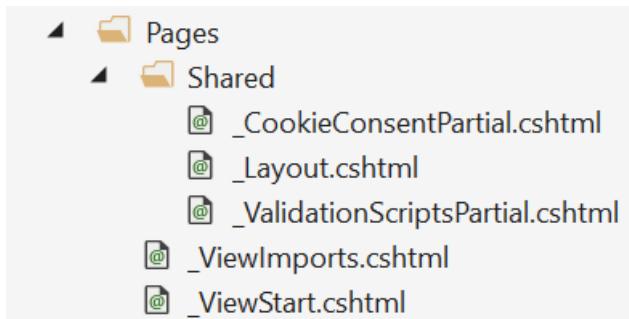
A maioria dos aplicativos Web tem um layout comum que fornece aos usuários uma experiência consistente durante sua navegação de uma página a outra. O layout normalmente inclui elementos comuns de interface do usuário, como o cabeçalho do aplicativo, elementos de menu ou de navegação e rodapé.



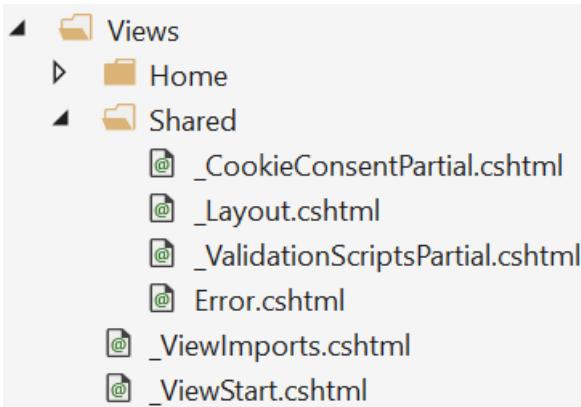
Estruturas HTML comuns, como scripts e folhas de estilo, também são usadas com frequência por muitas páginas em um aplicativo. Todos esses elementos compartilhados podem ser definidos em um arquivo de *layout*, que pode então ser referenciado por qualquer exibição usada no aplicativo. Os layouts reduzem o código duplicado em exibições, ajudando-os a seguir o princípio DRY (*Don't Repeat Yourself*).

Por convenção, o layout padrão de um aplicativo ASP.NET Core é chamado `_Layout.cshtml`. O arquivo de layout para novos projetos do ASP.NET Core criados com os modelos:

- Razor Pages: *Pages/Shared/\_Layout.cshtml*



- Controlador com exibições: *Views/Shared/\_Layout.cshtml*



O layout define um modelo de nível superior para exibições no aplicativo. Os aplicativos não exigem um layout. Os aplicativos podem definir mais de um layout, com diferentes exibições especificando diferentes layouts.

O código a seguir mostra o arquivo de layout para um modelo de projeto criado com um controlador e exibições:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ ViewData["Title"] - WebApplication1</title>

    <environment include="Development">
        <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
        <link rel="stylesheet" href("~/css/site.css" />
    </environment>
    <environment exclude="Development">
        <link rel="stylesheet"
        href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/css/bootstrap.min.css"
            asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"
            asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-
            test-value="absolute" />
        <link rel="stylesheet" href "~/css/site.min.css" asp-append-version="true" />
    </environment>
</head>
<body>
    <nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-
target=".navbar-collapse">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
            </div>
            <div class="collapse navbar-collapse" id="navbar">
                <ul class="nav navbar-nav">
                    <li class="active"><a href="#">Home</a></li>
                    <li><a href="#">About</a></li>
                    <li><a href="#">Contact</a></li>
                </ul>
            </div>
        </div>
    </nav>
    <div class="container">
        <partial name="CookieConsentPartial" />
        <div class="row">
            <div class="col-md-3">
                <h2>My Application</h2>
                <p>This is the main content area.</p>
            </div>
            <div class="col-md-9">
                <h3>About Page</h3>
                <p>This page contains information about the application.</p>
            </div>
        </div>
    </div>
</body>
</html>
```

```

        <span class="icon-bar"></span>
    </button>
    <a asp-page="/Index" class="navbar-brand">WebApplication1</a>
</div>
<div class="navbar-collapse collapse">
    <ul class="nav navbar-nav">
        <li><a asp-page="/Index">Home</a></li>
        <li><a asp-page="/About">About</a></li>
        <li><a asp-page="/Contact">Contact</a></li>
    </ul>
</div>
</div>
</nav>

<partial name="_CookieConsentPartial" />

<div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
        <p>&copy; 2018 - WebApplication1</p>
    </footer>
</div>

<environment include="Development">
    <script src="~/lib/jquery/dist/jquery.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.js"></script>
    <script src="~/js/site.js" asp-append-version="true"></script>
</environment>
<environment exclude="Development">
    <script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-3.3.1.min.js"
        asp-fallback-src="~/lib/jquery/dist/jquery.min.js"
        asp-fallback-test="window.jQuery"
        crossorigin="anonymous"
        integrity="sha384-tsQFqpEReu7ZLhBV2VZlAu7zc0V+rXbY1F2cqB8txI/8aZajjp4Bqd+V6D5IgvKT">
    </script>
    <script src="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/bootstrap.min.js"
        asp-fallback-src="~/lib/bootstrap/dist/js/bootstrap.min.js"
        asp-fallback-test="window.jQuery && window.jQuery.fn && window.jQuery.fn.modal"
        crossorigin="anonymous"
        integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWvXzUPnCJA712mCWNIpG9mGCD8wGNICPD7Txa">
    </script>
    <script src="~/js/site.min.js" asp-append-version="true"></script>
</environment>

    @RenderSection("Scripts", required: false)
</body>
</html>

```

## Especificando um layout

As exibições do Razor têm uma propriedade `Layout`. As exibições individuais especificam um layout com a configuração dessa propriedade:

```

@{
    Layout = "_Layout";
}

```

O layout especificado pode usar um caminho completo (por exemplo, `/Pages/Shared/_Layout.cshtml` ou `/Views/Shared/_Layout.cshtml`) ou um nome parcial (exemplo: `_Layout`). Quando um nome parcial for fornecido, o mecanismo de exibição do Razor pesquisará o arquivo de layout usando seu processo de descoberta padrão. A pasta em que o método do manipulador (ou controlador) existe é pesquisada primeiro, seguida pela pasta `Shared`. Esse processo de descoberta é idêntico àquele usado para descobrir

exibições parciais.

Por padrão, todo layout precisa chamar `RenderBody`. Sempre que a chamada a `RenderBody` for feita, o conteúdo da exibição será renderizado.

## Seções

Um layout, opcionalmente, pode referenciar uma ou mais seções, chamando `RenderSection`. As seções fornecem uma maneira de organizar o local em que determinados elementos da página devem ser colocados. Cada chamada a `RenderSection` pode especificar se essa seção é obrigatória ou opcional:

```
@section Scripts {
    @RenderSection("Scripts", required: false)
}
```

Se uma seção obrigatória não for encontrada, uma exceção será gerada. As exibições individuais especificam o conteúdo a ser renderizado em uma seção usando a sintaxe Razor `@section`. Se uma página ou exibição definir uma seção, ela precisará ser renderizada (ou ocorrerá um erro).

Uma definição `@section` de exemplo na exibição do Razor Pages:

```
@section Scripts {
    <script type="text/javascript" src="/scripts/main.js"></script>
}
```

No código anterior, `scripts/main.js` é adicionado à seção `scripts` em uma página ou exibição. Outras páginas ou exibições no mesmo aplicativo podem não exigir esse script e não definirão uma seção de scripts.

A marcação a seguir usa o [Auxiliar de Marca Parcial](#) para renderizar `_ValidationScriptsPartial.cshtml`:

```
@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
```

A marcação anterior foi gerada pela [Identidade de scaffolding](#).

As seções definidas em uma página ou exibição estão disponíveis apenas em sua página de layout imediata. Elas não podem ser referenciadas em parciais, componentes de exibição ou outras partes do sistema de exibição.

## Ignorando seções

Por padrão, o corpo e todas as seções de uma página de conteúdo precisam ser renderizados pela página de layout. O mecanismo de exibição do Razor impõe isso acompanhando se o corpo e cada seção foram renderizados.

Para instruir o mecanismo de exibição a ignorar o corpo ou as seções, chame os métodos `IgnoreBody` e `IgnoreSection`.

O corpo e cada seção em uma página do Razor precisam ser renderizados ou ignorados.

## Importando diretivas compartilhadas

Exibições e páginas podem usar diretivas do Razor para importar namespaces e usar [injeção de dependência](#). Diretivas compartilhadas por diversas exibições podem ser especificadas em um arquivo `_ViewImports.cshtml` comum. O arquivo `_ViewImports` dá suporte às seguintes diretivas:

- `@addTagHelper`
- `@removeTagHelper`
- `@tagHelperPrefix`
- `@using`
- `@model`
- `@inherits`
- `@inject`

O arquivo não dá suporte a outros recursos do Razor, como funções e definições de seção.

Um arquivo `_ViewImports.cshtml` de exemplo:

```
@using WebApplication1
@using WebApplication1.Models
@using WebApplication1.Models.AccountViewModels
@using WebApplication1.Models.ManageViewModels
@using Microsoft.AspNetCore.Identity
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

O arquivo `_ViewImports.cshtml` para um aplicativo ASP.NET Core MVC normalmente é colocado na pasta *Pages* (ou *Views*). Um arquivo `_ViewImports.cshtml` pode ser colocado em qualquer pasta, caso em que só será aplicado a páginas ou exibições nessa pasta e em suas subpastas. Arquivos `_ViewImports` são processados começando no nível raiz e, em seguida, para cada pasta até o local da página ou da exibição em si. As configurações `_ViewImports` especificadas no nível raiz podem ser substituídas no nível da pasta.

Por exemplo, suponha:

- O arquivo `_ViewImports.cshtml` no nível de raiz inclui `@model MyModel1` e `@addTagHelper *, MyTagHelper1`.
- Um arquivo `_ViewImports.cshtml` de subpasta inclui `@model MyModel2` e `@addTagHelper *, MyTagHelper2`.

Páginas e exibições na subpasta terão acesso a Auxiliares de Marca e ao modelo `MyModel2`.

Se vários arquivos `_ViewImports.cshtml` forem encontrados na hierarquia de arquivos, o comportamento combinado das diretivas será:

- `@addTagHelper`, `@removeTagHelper`: todos são executados, em ordem
- `@tagHelperPrefix`: o mais próximo à exibição substitui todos os outros
- `@model`: o mais próximo à exibição substitui todos os outros
- `@inherits`: o mais próximo à exibição substitui todos os outros
- `@using`: todos são incluídos; duplicatas são ignoradas
- `@inject`: para cada propriedade, o mais próximo à exibição substitui todos os outros com o mesmo nome de propriedade

## Executando o código antes de cada exibição

Código que precisa ser executado antes de cada exibição ou página precisa ser colocada no arquivo `_ViewStart.cshtml`. Por convenção, o arquivo `_ViewStart.cshtml` está localizado na pasta *Pages* (ou *Views*). As instruções listadas em `_ViewStart.cshtml` são executadas antes de cada exibição completa (não layouts nem exibições parciais). Como `_ViewImports.cshtml`, `_ViewStart.cshtml` é hierárquico. Se um arquivo `_ViewStart.cshtml` for definido na pasta de exibições ou páginas, ele será executado depois daquele definido na raiz da pasta *Pages* (ou *Views*) (se houver).

Um arquivo `_ViewStart.cshtml` de amostra:

```
@{  
    Layout = "_Layout";  
}
```

O arquivo acima especifica que todas as exibições usarão o layout `_Layout.cshtml`.

`_ViewStart.cshtml` é `_ViewImports.cshtml` normalmente **não** são colocados na pasta `/Pages/Shared` (ou `/Views/Shared`). As versões no nível do aplicativo desses arquivos devem ser colocadas diretamente na pasta `/Pages` (ou `/Views`).

# Arquivos estáticos no ASP.NET Core

08/01/2019 • 15 minutes to read • [Edit Online](#)

Por [Rick Anderson](#) e [Scott Addie](#)

Arquivos estáticos, como HTML, CSS, imagens e JavaScript, são ativos que um aplicativo ASP.NET Core fornece diretamente para os clientes. Algumas etapas de configuração são necessárias para habilitar o fornecimento desses arquivos.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Fornecer arquivos estáticos

Os arquivos estáticos são armazenados no diretório raiz Web do projeto. O diretório padrão é `<content_root>/wwwroot`, mas pode ser alterado por meio do método [UseWebRoot](#). Consulte [Raiz de conteúdo](#) e [Diretório base](#) para obter mais informações.

O host Web do aplicativo deve ser informado do diretório raiz do conteúdo.

O método `WebHost.CreateDefaultBuilder` define a raiz do conteúdo como o diretório atual:

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>();
}
```

Defina a raiz do conteúdo com o diretório atual invocando [UseContentRoot](#) dentro de `Program.Main`:

```
public class Program
{
    public static void Main(string[] args)
    {
        var host = new WebHostBuilder()
            .UseKestrel()
            .UseContentRoot(Directory.GetCurrentDirectory())
            .UseIISIntegration()
            .UseStartup<Startup>()
            .UseApplicationInsights()
            .Build();

        host.Run();
    }
}
```

Os arquivos estáticos são acessíveis por meio de um caminho relativo ao diretório base. Por exemplo, o modelo de projeto do **Aplicativo Web** contém várias pastas dentro da pasta `wwwroot`:

- **wwwroot**
  - **css**

- o **images**
- o **js**

O formato de URI para acessar um arquivo na subpasta *images* é `http://<server_address>/images/<image_file_name>`. Por exemplo, `http://localhost:9189/images/banner3.svg`.

Se estiver direcionando o .NET Framework, adicione o pacote [Microsoft.AspNetCore.StaticFiles](#) ao projeto. Se você estiver direcionando para o .NET Core, o [metapacote Microsoft.AspNetCore.App](#) incluirá esse pacote.

Se estiver direcionando o .NET Framework, adicione o pacote [Microsoft.AspNetCore.StaticFiles](#) ao projeto. Se estiver direcionando o .NET Core, o [metapacote Microsoft.AspNetCore.All](#) incluirá esse pacote.

Adicione o pacote [Microsoft.AspNetCore.StaticFiles](#) ao projeto.

Configure o [middleware](#) que permite o fornecimento de arquivos estáticos.

### Fornecer arquivos dentro do diretório base

Invoca o método `UseStaticFiles` dentro de `Startup.Configure`:

```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles();
}
```

A sobrecarga do método `UseStaticFiles` sem parâmetro marca os arquivos no diretório base como fornecíveis. A seguinte marcação referencia `wwwroot/images/banner1.svg`:

```

```

No código anterior, o caractere til `~/` aponta para o diretório base. Para obter mais informações, confira [Diretório base](#).

### Fornecer arquivos fora do diretório base

Considere uma hierarquia de diretórios na qual os arquivos estáticos a serem atendidos residam fora do diretório base:

- **wwwroot**
  - o **css**
  - o **images**
  - o **js**
- **MyStaticFiles**
  - o **images**
    - o *banner1.svg*

Uma solicitação pode acessar o arquivo *banner1.svg* configurando o middleware de arquivos estáticos da seguinte maneira:

```

public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles(); // For the wwwroot folder

    app.UseStaticFiles(new StaticFileOptions
    {
        FileProvider = new PhysicalFileProvider(
            Path.Combine(Directory.GetCurrentDirectory(), "MyStaticFiles")),
        RequestPath = "/StaticFiles"
    });
}

```

No código anterior, a hierarquia de diretórios `MyStaticFiles` é exposta publicamente por meio do segmento do URI `StaticFiles`. Uma solicitação para `http://<server_address>/StaticFiles/images/banner1.svg` atende ao arquivo `banner1.svg`.

A seguinte marcação referencia `MyStaticFiles/images/banner1.svg`:

```

```

### Definir cabeçalhos de resposta HTTP

Um objeto `StaticFileOptions` pode ser usado para definir cabeçalhos de resposta HTTP. Além de configurar o fornecimento de arquivos estáticos do diretório base, o seguinte código define o cabeçalho `Cache-Control`:

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    var cachePeriod = env.IsDevelopment() ? "600" : "604800";
    app.UseStaticFiles(new StaticFileOptions
    {
        OnPrepareResponse = ctx =>
        {
            // Requires the following import:
            // using Microsoft.AspNetCore.Http;
            ctx.Context.Response.Headers.Append("Cache-Control", $"public, max-age={cachePeriod}");
        }
    });
}

```

O método `HeaderDictionaryExtensions.Append` existe no pacote `Microsoft.AspNetCore.Http`.

Os arquivos se tornaram armazenáveis em cache publicamente por 10 minutos (600 segundos) no ambiente de desenvolvimento:



## Autorização de arquivo estático

O middleware de arquivos estáticos não fornece verificações de autorização. Todos os arquivos atendidos, incluindo aqueles em `wwwroot`, estão acessíveis publicamente. Para fornecer arquivos com base na autorização:

- Armazene-os fora do `wwwroot` e de qualquer diretório acessível ao middleware de arquivos estáticos.
- Forneça-os por meio de um método de ação ao qual a autorização é aplicada. Retorne um objeto

## FileResult:

```
[Authorize]
public IActionResult BannerImage()
{
    var file = Path.Combine(Directory.GetCurrentDirectory(),
                           "MyStaticFiles", "images", "banner1.svg");

    return PhysicalFile(file, "image/svg+xml");
}
```

## Habilitar navegação no diretório

A navegação no diretório permite que os usuários do aplicativo Web vejam uma listagem de diretório e arquivos em um diretório especificado. A navegação no diretório está desabilitada por padrão por motivos de segurança (consulte [Considerações](#)). Habilite a navegação no diretório invocando o método

[UseDirectoryBrowser](#) em `Startup.Configure`:

```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles(); // For the wwwroot folder

    app.UseStaticFiles(new StaticFileOptions
    {
        FileProvider = new PhysicalFileProvider(
            Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "images")),
        RequestPath = "/MyImages"
    });

    app.UseDirectoryBrowser(new DirectoryBrowserOptions
    {
        FileProvider = new PhysicalFileProvider(
            Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "images")),
        RequestPath = "/MyImages"
    });
}
```

Adicione os serviços necessários invocando o método [AddDirectoryBrowser](#) em `Startup.ConfigureServices`:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDirectoryBrowser();
}
```

O código anterior permite a navegação no diretório da pasta `wwwroot/images` usando a URL `http://<server_address>/MyImages`, com links para cada arquivo e pasta:

Name	Size	Last Modified
banner1.svg	9,679	5/13/2016 5:51:18 AM +00:00
banner2.svg	8,394	5/13/2016 5:51:18 AM +00:00
banner3.svg	10,872	5/13/2016 5:51:18 AM +00:00
banner4.svg	12,291	5/13/2016 5:51:18 AM +00:00

Consulte [Considerações](#) para saber mais sobre os riscos de segurança ao habilitar a navegação.

Observe as duas chamadas `UseStaticFiles` no exemplo a seguir. A primeira chamada permite o fornecimento de arquivos estáticos na pasta `wwwroot`. A segunda chamada habilita a navegação no diretório da pasta `wwwroot/images` usando a URL `http://<server_address>/MyImages`:

```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles(); // For the wwwroot folder

    app.UseStaticFiles(new StaticFileOptions
    {
        FileProvider = new PhysicalFileProvider(
            Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "images")),
        RequestPath = "/MyImages"
    });

    app.UseDirectoryBrowser(new DirectoryBrowserOptions
    {
        FileProvider = new PhysicalFileProvider(
            Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "images")),
        RequestPath = "/MyImages"
    });
}
```

## Fornecer um documento padrão

Definir uma home page padrão fornece ao visitantes um ponto de partida lógico de ao visitar o site. Para fornecer uma página padrão sem qualificar totalmente o URI do usuário, chame o `UseDefaultFiles` método de `Startup.Configure`:

```
public void Configure(IApplicationBuilder app)
{
    app.UseDefaultFiles();
    app.UseStaticFiles();
}
```

### IMPORTANT

`UseDefaultFiles` deve ser chamado antes de `UseStaticFiles` para fornecer o arquivo padrão. `UseDefaultFiles` é um rewriter de URL que, na verdade, não fornece o arquivo. Habilite o middleware de arquivos estáticos por meio de `UseStaticFiles` para fornecer o arquivo.

Com `UseDefaultFiles`, as solicitações para uma pasta pesquisam:

- `default.htm`
- `default.html`
- `index.htm`
- `index.html`

O primeiro arquivo encontrado na lista é fornecido como se a solicitação fosse o URI totalmente qualificado. A URL do navegador continua refletindo o URI solicitado.

O seguinte código altera o nome de arquivo padrão para `mydefault.html`:

```
public void Configure(IApplicationBuilder app)
{
    // Serve my app-specific default file, if present.
    DefaultFilesOptions options = new DefaultFilesOptions();
    options.DefaultFileNames.Clear();
    options.DefaultFileNames.Add("mydefault.html");
    app.UseDefaultFiles(options);
    app.UseStaticFiles();
}
```

## UseFileServer

`UseFileServer` combina a funcionalidade de `UseStaticFiles`, `UseDefaultFiles` e `UseDirectoryBrowser`.

O código a seguir permite o fornecimento de arquivos estáticos e do arquivo padrão. A navegação no diretório não está habilitada.

```
app.UseFileServer();
```

O seguinte código baseia-se na sobrecarga sem parâmetros, habilitando a navegação no diretório:

```
app.UseFileServer(enableDirectoryBrowsing: true);
```

Considere a seguinte hierarquia de diretórios:

- **wwwroot**
  - **css**
  - **images**
  - **js**
- **MyStaticFiles**
  - **images**
    - *banner1.svg*
    - *default.html*

O seguinte código habilita arquivos estáticos, arquivo padrão e a navegação no diretório de `MyStaticFiles`:

```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles(); // For the wwwroot folder

    app.UseFileServer(new FileServerOptions
    {
        FileProvider = new PhysicalFileProvider(
            Path.Combine(Directory.GetCurrentDirectory(), "MyStaticFiles")),
        RequestPath = "/StaticFiles",
        EnableDirectoryBrowsing = true
    });
}
```

`AddDirectoryBrowser` precisa ser chamado quando o valor da propriedade `EnableDirectoryBrowsing` é `true`:

```

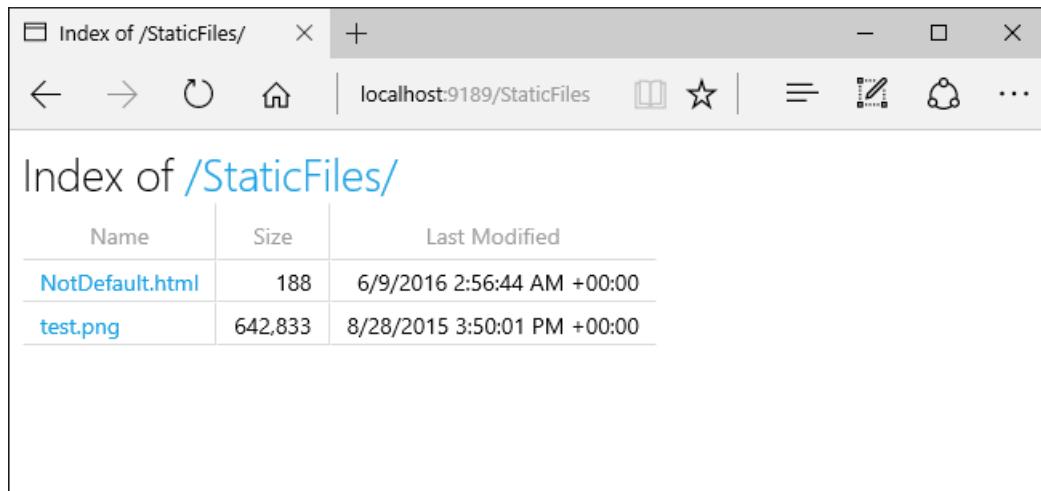
public void ConfigureServices(IServiceCollection services)
{
    services.AddDirectoryBrowser();
}

```

Com o uso da hierarquia de arquivos e do código anterior, as URLs são resolvidas da seguinte maneira:

URI	RESPOSTA
<code>http://&lt;server_address&gt;/StaticFiles/images/banner1.svg</code>	MyStaticFiles/images/banner1.svg
<code>http://&lt;server_address&gt;/StaticFiles</code>	MyStaticFiles/default.html

Se nenhum arquivo nomeado como padrão existir no diretório *MyStaticFiles*, `http://<server_address>/StaticFiles` retornará a listagem de diretórios com links clicáveis:



#### NOTE

`UseDefaultFiles` e `UseDirectoryBrowser` usam a URL `http://<server_address>/StaticFiles` sem a barra "" à direita para disparar um redirecionamento do lado do cliente para `http://<server_address>/StaticFiles/`. Observe a adição da barra "" à direita. URLs relativas dentro dos documentos são consideradas inválidas sem uma barra "" à direita.

## FileExtensionContentTypeProvider

A classe `FileExtensionContentTypeProvider` contém uma propriedade `Mappings` que serve como um mapeamento de extensões de arquivo para tipos de conteúdo MIME. Na amostra a seguir, várias extensões de arquivo são registradas para tipos MIME conhecidos. A extensão `.rtf` é substituída, e `.mp4` é removida.

```

public void Configure(IApplicationBuilder app)
{
    // Set up custom content types - associating file extension to MIME type
    var provider = new FileExtensionContentTypeProvider();
    // Add new mappings
    provider.Mappings[".myapp"] = "application/x-msdownload";
    provider.Mappings[".htm3"] = "text/html";
    provider.Mappings[".image"] = "image/png";
    // Replace an existing mapping
    provider.Mappings[".rtf"] = "application/x-msdownload";
    // Remove MP4 videos.
    provider.Mappings.Remove(".mp4");

    app.UseStaticFiles(new StaticFileOptions
    {
        FileProvider = new PhysicalFileProvider(
            Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "images")),
        RequestPath = "/MyImages",
        ContentTypeProvider = provider
    });

    app.UseDirectoryBrowser(new DirectoryBrowserOptions
    {
        FileProvider = new PhysicalFileProvider(
            Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "images")),
        RequestPath = "/MyImages"
    });
}

```

Consulte [Tipos de conteúdo MIME](#).

## Tipos de conteúdo não padrão

O middleware de arquivo estático compreende quase 400 tipos de conteúdo de arquivo conhecidos. Se o usuário solicitar um arquivo com um tipo desconhecido, o middleware de arquivo estático passará a solicitação para o próximo middleware no pipeline. Se nenhum middleware manipular a solicitação, uma resposta **404 Não Encontrado** será retornada. Se a navegação no diretório estiver habilitada, um link para o arquivo será exibido na lista do diretório.

O seguinte código habilita o fornecimento de tipos desconhecidos e renderiza o arquivo desconhecido como uma imagem:

```

public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles(new StaticFileOptions
    {
        ServeUnknownFileTypes = true,
        DefaultContentType = "image/png"
    });
}

```

Com o código anterior, uma solicitação para um arquivo com um tipo de conteúdo desconhecido é retornada como uma imagem.

### WARNING

A habilitação de `ServeUnknownFileTypes` é um risco de segurança. Ela está desabilitada por padrão, e seu uso não é recomendado. `FileExtensionContentTypeProvider` fornece uma alternativa mais segura para o fornecimento de arquivos com extensões não padrão.

## Considerações

### WARNING

`UseDirectoryBrowser` e `UseStaticFiles` podem causar a perda de segredos. A desabilitação da navegação no diretório em produção é altamente recomendada. Examine com atenção os diretórios que são habilitados por meio de `UseStaticFiles` ou `UseDirectoryBrowser`. Todo o diretório e seus subdiretórios se tornam publicamente acessíveis. Armazene arquivos adequados para fornecimento ao público em um diretório dedicado, como `<content_root>/wwwroot`. Separe esses arquivos das exibições MVC, Páginas do Razor (somente 2.x), arquivos de configuração, etc.

- As URLs para o conteúdo exposto com `UseDirectoryBrowser` e `UseStaticFiles` estão sujeitas à diferenciação de maiúsculas e minúsculas e a restrições de caracteres do sistema de arquivos subjacente. Por exemplo, o Windows diferencia maiúsculas de minúsculas, o macOS e o Linux não.
- Os aplicativos ASP.NET Core hospedados no IIS usam o [Módulo do ASP.NET Core](#) para encaminhar todas as solicitações ao aplicativo, inclusive as solicitações de arquivo estático. O manipulador de arquivos estáticos do IIS não é usado. Ele não tem nenhuma possibilidade de manipular as solicitações antes que elas sejam manipuladas pelo módulo.
- Conclua as etapas seguinte no Gerenciador do IIS para remover o manipulador de arquivos estáticos no IIS no nível do servidor ou do site:
  - Navegue para o recurso **Módulos**.
  - Selecione **StaticFileModule** na lista.
  - Clique em **Remover** na barra lateral **Ações**.

### WARNING

Se o manipulador de arquivo estático do IIS estiver habilitado e o Módulo do ASP.NET Core não estiver configurado corretamente, os arquivos estáticos serão atendidos. Isso acontece, por exemplo, se o arquivo `web.config` não foi implantado.

- Coloque arquivos de código (incluindo `.cs` e `.cshtml`) fora do diretório base do projeto do aplicativo. Portanto, uma separação lógica é criada entre o conteúdo do lado do cliente do aplicativo e o código baseado em servidor. Isso impede a perda de código do lado do servidor.

## Recursos adicionais

- [Middleware](#)
- [Introdução ao ASP.NET Core](#)

# Model binding no ASP.NET Core

22/11/2018 • 16 minutes to read • [Edit Online](#)

Por [Rachel Appel](#)

## Introdução ao model binding

O model binding do ASP.NET Core MVC mapeia dados de solicitações HTTP para parâmetros de método de ação. Os parâmetros podem ser tipos simples, como cadeias de caracteres, inteiros ou floats, ou podem ser tipos complexos. Esse é um ótimo recurso do MVC, pois o mapeamento dos dados de entrada para um equivalente é um cenário repetido com frequência, independentemente do tamanho ou da complexidade dos dados. O MVC resolve esse problema com a abstração da associação, de modo que os desenvolvedores não precisem continuar reconfigurando uma versão ligeiramente diferente desse mesmo código em cada aplicativo. A escrita de seu próprio texto para tipar o código de conversor é entediante e propensa a erros.

## Como funciona o model binding

Quando o MVC recebe uma solicitação HTTP, ele encaminha-a a um método de ação específico de um controlador. Ele determina qual método de ação será executado com base nos dados de rota e, em seguida, ele associa valores da solicitação HTTP aos parâmetros desse método de ação. Por exemplo, considere a seguinte URL:

```
http://contoso.com/movies/edit/2
```

Como o modelo de rota é semelhante a isto, `{controller=Home}/{action=Index}/{id?}`, `movies/edit/2` encaminha para o controlador `Movies` e seu método de ação `Edit`. Ele também aceita um parâmetro opcional chamado `id`. O código para o método de ação deve ter esta aparência:

```
public IActionResult Edit(int? id)
```

Observação: as cadeias de caracteres na rota de URL não diferenciam maiúsculas de minúsculas.

O MVC tentará associar os dados de solicitação aos parâmetros de ação por nome. O MVC procurará valores para cada parâmetro usando o nome do parâmetro e os nomes de suas propriedades configuráveis públicas. No exemplo acima, o único parâmetro de ação é chamado `id`, que o MVC associa ao valor com o mesmo nome nos valores de rota. Além dos valores de rota, o MVC associará dados de várias partes da solicitação e faz isso em uma ordem específica. Veja abaixo uma lista das fontes de dados na ordem em que o model binding examina-as:

1. `Form values`: esses são valores de formulário que entram na solicitação HTTP usando o método POST. (incluindo as solicitações POST jQuery).
2. `Route values`: o conjunto de valores de rota fornecido pelo [Roteamento](#)
3. `Query strings`: a parte da cadeia de caracteres de consulta do URI.

Observação: valores de formulário, dados de rota e cadeias de consulta são todos armazenados como pares nome-valor.

Como o model binding solicitou uma chave chamada `id` e não há nada chamado `id` nos valores de

formulário, ela passou para os valores de rota procurando essa chave. Em nosso exemplo, isso é uma correspondência. A associação ocorre e o valor é convertido no inteiro 2. A mesma solicitação que usa `Edit(string id)` converterá na cadeia de caracteres "2".

Até agora, o exemplo usa tipos simples. No MVC, os tipos simples são qualquer tipo primitivo do .NET ou um tipo com um conversor de tipo de cadeia de caracteres. Se o parâmetro do método de ação for uma classe, como o tipo `Movie`, que contém tipos simples e complexos como propriedades, o model binding do MVC ainda o manipulará perfeitamente. Ele usa a reflexão e recursão para percorrer as propriedades de tipos complexos procurando correspondências. O model binding procura o padrão `nome_do_parâmetro.nome_da_propriedade` para associar valores a propriedades. Se ela não encontrar os valores correspondentes desse formulário, ela tentará associar usando apenas o nome da propriedade. Para esses tipos como tipos `collection`, o model binding procura correspondências para `parameter_name[index]` ou apenas `[index]`. O model binding trata tipos `Dictionary` da mesma forma, solicitando `parameter_name[key]` ou apenas `[key]`, desde que as chaves sejam tipos simples. As chaves compatíveis correspondem o HTML dos nomes de campo e os auxiliares de marca gerados para o mesmo tipo de modelo. Isso permite a ida e vinda dos valores, de modo que os campos de formulário permaneçam preenchidos com a entrada do usuário para sua conveniência, por exemplo, quando os dados associados de uma criação ou edição não são aprovados na validação.

Para possibilitar o model binding, a classe deve ter um construtor padrão público e propriedades públicas graváveis para associar. Quando o model binding ocorrer, será criada uma instância da classe usando o construtor padrão público e, em seguida, as propriedades poderão ser definidas.

Quando um parâmetro é associado, o model binding para de procurar valores com esse nome e ele passa para associar o próximo parâmetro. Caso contrário, o comportamento padrão do model binding define parâmetros com seus valores padrão, dependendo de seu tipo:

- `T[]`: com a exceção de matrizes do tipo `byte[]`, a associação define parâmetros do tipo `T[]` como `Array.Empty<T>()`. Matrizes do tipo `byte[]` são definidas como `null`.
- Tipos de referência: a associação cria uma instância de uma classe com o construtor padrão sem configurar propriedades. No entanto, o model binding define parâmetros `string` como `null`.
- Tipos que permitem valor nulo: os tipos que permitem valor nulo são definidos como `null`. No exemplo acima, o model binding define `id` como `null`, pois ele é do tipo `int?`.
- Tipos de valor: os tipos de valor que não permitem valor nulo do tipo `T` são definidos como `default(T)`. Por exemplo, o model binding definirá um parâmetro `int id` como 0. Considere o uso da validação de modelo ou de tipos que permitem valor nulo, em vez de depender de valores padrão.

Se a associação falhar, o MVC não gerará um erro. Todas as ações que aceitam a entrada do usuário devem verificar a propriedade `ModelState.IsValid`.

Observação: cada entrada na propriedade `ModelState` do controlador é uma `ModelStateEntry` que contém uma propriedade `Errors`. Raramente é necessário consultar essa coleção por conta própria. Use `ModelState.IsValid` em seu lugar.

Além disso, há alguns tipos de dados especiais que o MVC precisa considerar ao realizar o model binding:

- `IFormFile`, `IEnumerable<IFormFile>`: um ou mais arquivos carregados que fazem parte da solicitação HTTP.

- `CancellationToken` : usado para cancelar a atividade em controladores assíncronos.

Esses tipos podem ser associados a parâmetros de ação ou a propriedades em um tipo de classe.

Após a conclusão do model binding, a [Validação](#) ocorrerá. O model binding padrão funciona bem para a maioria dos cenários de desenvolvimento. Também é extensível e, portanto, se você tiver necessidades exclusivas, poderá personalizar o comportamento interno.

## Personalizar o comportamento do model binding com atributos

O MVC contém vários atributos que podem ser usados para direcionar seu comportamento de model binding padrão para outra fonte. Por exemplo, você pode especificar se a associação é obrigatória para uma propriedade ou se ela nunca deve ocorrer usando os atributos `[BindRequired]` ou `[BindNever]`. Como alternativa, você pode substituir a fonte de dados padrão e especificar a fonte de dados do associador de modelos. Veja abaixo uma lista dos atributos de model binding:

- `[BindRequired]` : esse atributo adiciona um erro de estado do modelo se a associação não pode ocorrer.
- `[BindNever]` : instrui o associador de modelos a nunca associar a esse parâmetro.
- `[FromHeader]` , `[FromQuery]` , `[FromRoute]` , `[FromForm]` : use-os para especificar a origem da associação exata que deseja aplicar.
- `[FromServices]` : esse atributo usa a [injeção de dependência](#) para associar parâmetros de serviços.
- `[FromBody]` : use os formatadores configurados para associar dados do corpo da solicitação. O formatador é selecionado de acordo com o tipo de conteúdo da solicitação.
- `[ModelBinder]` : usado para substituir o associador de modelos padrão, a origem da associação e o nome.

Os atributos são ferramentas muito úteis quando você precisa substituir o comportamento padrão do model binding.

## Personalizar a validação e o model binding globalmente

O comportamento do sistema de validação e model binding é orientado pelo [ModelMetadata](#) que descreve:

- Como um modelo deve ser associado.
- Como ocorre a validação no tipo e em suas propriedades.

Os aspectos do comportamento do sistema podem ser configurados globalmente adicionando um provedor de detalhes a [MvcOptions.ModelMetadataDetailsProviders](#). O MVC tem alguns provedores de detalhes internos que permitem configurar o comportamento, como desabilitar a validação ou o model binding de determinados tipos.

Para desabilitar o model binding em todos os modelos de um determinado tipo, adicione um `ExcludeBindingMetadataProvider` em `Startup.ConfigureServices`. Por exemplo, para desabilitar o model binding em todos os modelos do tipo `System.Version`:

```
services.AddMvc().AddMvcOptions(options =>
    options.ModelMetadataDetailsProviders.Add(
        new ExcludeBindingMetadataProvider(typeof(System.Version))));
```

Para desabilitar a validação nas propriedades de um determinado tipo, adicione um `SuppressChildValidationMetadataProvider` em `Startup.ConfigureServices`. Por exemplo, para desabilitar a validação nas propriedades do tipo `System.Guid`:

```
services.AddMvc().AddMvcOptions(options =>
    options.ModelMetadataDetailsProviders.Add(
        new SuppressChildValidationMetadataProvider(typeof(System.Guid))));
```

## Associar dados formatados do corpo da solicitação

Os dados de solicitação podem ser recebidos em uma variedade de formatos, incluindo JSON, XML e muitos outros. Quando você usa o atributo `[FromBody]` para indicar que deseja associar um parâmetro a dados no corpo da solicitação, o MVC usa um conjunto configurado de formatadores para manipular os dados de solicitação de acordo com seu tipo de conteúdo. Por padrão, o MVC inclui uma classe `JsonInputFormatter` para manipular dados JSON, mas você pode adicionar outros formatadores para manipular XML e outros formatos personalizados.

### NOTE

Pode haver, no máximo, um parâmetro por ação decorado com `[FromBody]`. O tempo de execução do ASP.NET Core MVC delega a responsabilidade de ler o fluxo da solicitação ao formatador. Depois que o fluxo da solicitação é lido para um parâmetro, geralmente, não é possível ler o fluxo da solicitação novamente para associar outros parâmetros `[FromBody]`.

### NOTE

O `JsonInputFormatter` é o formatador padrão e se baseia no [Json.NET](#).

O ASP.NET Core seleciona formatadores de entrada com base no cabeçalho `Content-Type` e no tipo do parâmetro, a menos que haja um atributo aplicado a ele com outra especificação. Se deseja usar XML ou outro formato, configure-o no arquivo `Startup.cs`, mas talvez você precise obter primeiro uma referência a `Microsoft.AspNetCore.Mvc.Formatters.Xml` usando o NuGet. O código de inicialização deverá ter uma aparência semelhante a esta:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc()
        .AddXmlSerializerFormatters();
}
```

O código no arquivo `Startup.cs` contém um método `ConfigureServices` com um argumento `services` que você pode usar para criar serviços para o aplicativo ASP.NET Core. No exemplo, estamos adicionando um formatador XML como um serviço que o MVC fornecerá para este aplicativo. O argumento `options` passado para o método `AddMvc` permite que você adicione e gerencie filtros, formatadores e outras opções do sistema no MVC após a inicialização do aplicativo. Em seguida, aplique o atributo `Consumes` a classes do controlador ou métodos de ação para trabalhar com o formato desejado.

### Model binding personalizado

Estenda o model binding escrevendo seus próprios associadores de modelos personalizados. Saiba mais sobre o [model binding personalizado](#).

# Validação de modelo no ASP.NET Core MVC

21/01/2019 • 32 minutes to read • [Edit Online](#)

Por [Rachel Appel](#)

## Introdução à validação do modelo

Antes que um aplicativo armazene dados em um banco de dados, ele deve validá-los. Os dados precisam ser verificados quanto a possíveis ameaças de segurança, se estão formatados corretamente por tipo e tamanho e precisam estar em conformidade com as regras. A validação é necessária, embora possa ser redundante e monótona de ser implementada. No MVC, a validação ocorre no cliente e no servidor.

Felizmente, o .NET abstraiu a validação para atributos de validação. Esses atributos contêm o código de validação, reduzindo a quantidade de código que precisa ser escrita.

No ASP.NET Core 2.2 e posterior, o tempo de execução do ASP.NET Core encurta a validação (ignora) se puder determinar que um determinado gráfico de modelo não requer validação. Ignorar a validação pode fornecer melhorias significativas no desempenho ao validar modelos que não podem ou não tem nenhum validador associado. A validação ignorada inclui objetos, como coleções de primitivos (`byte[]`, `string[]`, `Dictionary<string, string>` etc.) ou grafos de objeto complexo sem nenhum validador.

[Exibir ou baixar amostra do GitHub.](#)

## Atributos de validação

Os atributos de validação são uma maneira de configurar a validação do modelo; portanto, é conceitualmente semelhante à validação em campos de tabelas de banco de dados. Isso inclui restrições, como atribuição de tipos de dados ou campos obrigatórios. Outros tipos de validação incluem a aplicação de padrões a dados para impor regras de negócios, como um cartão de crédito, número de telefone ou endereço de email. Os atributos de validação simplificam e facilitam o uso da imposição desses requisitos.

Os atributos de validação são especificados no nível da propriedade:

```
[Required]  
public string MyProperty { get; set; }
```

Veja abaixo um modelo `Movie` anotado de um aplicativo que armazena informações sobre filmes e programas de TV. A maioria das propriedades é obrigatória e várias propriedades de cadeia de caracteres têm requisitos de tamanho. Além disso, há uma restrição de intervalo numérico em vigor para a propriedade `Price` de 0 a US\$ 999,99, juntamente com um atributo de validação personalizado.

```

public class Movie
{
    public int Id { get; set; }

    [Required]
    [StringLength(100)]
    public string Title { get; set; }

    [ClassicMovie(1960)]
    [DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }

    [Required]
    [StringLength(1000)]
    public string Description { get; set; }

    [Range(0, 999.99)]
    public decimal Price { get; set; }

    [Required]
    public Genre Genre { get; set; }

    public bool Preorder { get; set; }
}

```

A simples leitura do modelo revela as regras sobre os dados para esse aplicativo, facilitando a manutenção do código. Veja abaixo vários atributos de validação internos populares:

- `[CreditCard]` : valida se a propriedade tem um formato de cartão de crédito.
- `[Compare]` : valida se duas propriedades em um modelo são correspondentes.
- `[EmailAddress]` : valida se a propriedade tem um formato de email.
- `[Phone]` : valida se a propriedade tem um formato de telefone.
- `[Range]` : valida se o valor da propriedade está dentro do intervalo especificado.
- `[RegularExpression]` : valida se os dados correspondem à expressão regular especificada.
- `[Required]` : torna uma propriedade obrigatória.
- `[StringLength]` : valida se uma propriedade de cadeia de caracteres tem, no máximo, o tamanho máximo especificado.
- `[Url]` : valida se a propriedade tem um formato de URL.

O MVC dá suporte a qualquer atributo derivado de `ValidationAttribute` para fins de validação. Muitos atributos de validação úteis podem ser encontrados no namespace [System.ComponentModel.DataAnnotations](#).

Pode haver ocasiões em que você precisa de mais recursos do que os atributos internos fornecem. Para essas ocasiões, é possível criar atributos de validação personalizados pela derivação de `ValidationAttribute` ou alteração do modelo para implementar `IValidatableObject`.

## Observações sobre o uso do atributo Required

Os [tipos de valor](#) que não permitem valores nulos (como `decimal`, `int`, `float` e `DateTime`) são inherentemente necessários e não precisam do atributo `Required`. O aplicativo não executa nenhuma verificação de validação do lado do servidor para tipos que não permitem valor nulo marcados como `Required`.

O model binding do MVC, que não está preocupado com a validação nem com atributos de validação, rejeita o envio de um campo de formulário que contém um valor ausente ou espaço em branco para um tipo que não permite valor nulo. Na ausência de um atributo `BindRequired` na propriedade de destino, o model binding ignora dados ausentes para tipos que não permitem valor nulo, nos quais o campo de formulário está ausente nos dados de formulário de entrada.

O [atributo `BindRequired`](#) (veja também [Model binding no ASP.NET Core](#)) é útil para garantir que os dados do formulário sejam preenchidos. Quando aplicado a uma propriedade, o sistema de model binding exige um valor para essa propriedade. Quando aplicado a um tipo, o sistema de model binding exige valores para todas as propriedades desse tipo.

Quando você usa um tipo `Nullable<T>` (por exemplo, `decimal?` ou `System.Nullable<decimal>`) e marca-o como `Required`, é realizada uma verificação de validação do lado do servidor, como se a propriedade fosse um tipo que permite valor nulo padrão (para exemplo, uma `string`).

A validação do lado do cliente exige um valor para um campo de formulário que corresponde a uma propriedade de modelo que foi marcada como `Required` e para uma propriedade de tipo que não permite valor nulo que ainda não foi marcada como `Required`. `Required` pode ser usado para controlar a mensagem de erro de validação do lado do cliente.

## Validação de nó de nível superior

Os nós de nível superior incluem:

- Parâmetros de ação
- Propriedades do controlador
- Parâmetros do manipulador de página
- Propriedades do modelo de página

Os nós de nível superior associados ao modelo são validados além de validar as propriedades do modelo. No exemplo a seguir do aplicativo de exemplo, o método `VerifyPhone` usa o [RegularExpressionAttribute](#) para validar os dados de usuário no campo de telefone de um formulário:

```
[AcceptVerbs("Get", "Post")]
public IActionResult VerifyPhone(
    [RegularExpression(@"^\d{3}-\d{3}-\d{4}$")] string phone)
{
    if (!ModelState.IsValid)
    {
        return Json($"Phone {phone} has an invalid format. Format: ###-###-####");
    }

    return Json(true);
}
```

Os nós de nível superior podem usar [BindRequiredAttribute](#) com atributos de validação. No exemplo a seguir do aplicativo de exemplo, o método `CheckAge` especifica que o parâmetro `age` deve ser associado da cadeia de caracteres de consulta quando o formulário é enviado:

```
[HttpPost]
public IActionResult CheckAge(
    [BindRequired, FromQuery] int age)
```

Na página Verificar Idade (`CheckAge.cshtml`), há dois formulários. O primeiro formulário envia um

valor `Age` de `99` como uma cadeia de caracteres de consulta:

`https://localhost:5001/Users/CheckAge?Age=99`.

Quando um parâmetro `age` formatado corretamente da cadeia de caracteres de consulta é enviado, o formulário é validado.

O segundo formulário na página Verificar Idade envia o valor `Age` no corpo da solicitação e a validação falha. A associação falha porque o parâmetro `age` deve vir de uma cadeia de caracteres de consulta.

A validação é habilitada por padrão e é controlada pela propriedade `AllowValidatingTopLevelNodes` de `MvcOptions`. Para desabilitar a validação do nó de nível superior, defina `AllowValidatingTopLevelNodes` como `false` nas opções de MVC (`Startup.ConfigureServices`):

```
services.AddMvc(options =>
{
    options.MaxModelValidationErrors = 50;
    options.AllowValidatingTopLevelNodes = false;
})
.SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
```

## Estado do modelo

O estado do modelo representa erros de validação nos valores de formulário HTML enviados.

O MVC continuará validando campos até atingir o número máximo de erros (200 por padrão). Você pode configurar esse número com o seguinte código no `Startup.ConfigureServices`:

```
services.AddMvc(options => options.MaxModelValidationErrors = 50)
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
```

## Erros de estado do modelo do identificador

A validação de modelo ocorre antes da execução de uma ação de controlador. É responsabilidade da ação inspecionar `ModelState.IsValid` e reagir adequadamente. Em muitos casos, a resposta apropriada é retornar uma resposta de erro, de preferência, fornecendo detalhes sobre o motivo pelo qual houve falha na validação do modelo.

Quando `ModelState.IsValid` for avaliada como `false` em controladores de API Web usando o atributo `[ApiController]`, uma resposta automática HTTP 400 contendo detalhes do problema será retornada. Para obter mais informações, veja [Respostas automáticas HTTP 400](#).

Alguns aplicativos optarão por seguir uma convenção padrão para lidar com erros de validação do modelo, caso em que um filtro pode ser um local adequado para implementar uma política como essa. É necessário testar o comportamento das ações com estados de modelo válidos e inválidos.

## Validação manual

Após a conclusão da validação e de model binding, recomendamos repetir partes disso. Por exemplo, um usuário pode ter inserido um texto em um campo que espera um inteiro ou talvez você precise calcular um valor da propriedade de um modelo.

Talvez seja necessário executar a validação manualmente. Para fazer isso, chame o método `TryValidateModel`, conforme mostrado aqui:

```
TryValidateModel(movie);
```

## Validação personalizada

Os atributos de validação funcionam para a maior parte das necessidades de validação. No entanto, algumas regras de validação são específicas ao negócio. As regras podem não ser técnicas comuns de validação de dados, como garantir que um campo seja obrigatório ou que ele esteja em conformidade com um intervalo de valores. Para esses cenários, os atributos de validação personalizados são uma ótima solução. É fácil criar seus próprios atributos de validação personalizados no MVC. Basta herdar do `ValidationAttribute` e substituir o método `IsValid`. O método `IsValid` aceita dois parâmetros: o primeiro é um objeto chamado `value` e o segundo é um objeto `ValidationContext` chamado `validationContext`. `Value` refere-se ao valor real do campo que o validador personalizado está validando.

Na amostra a seguir, uma regra de negócios indica que os usuários não podem definir o gênero como *Clássico* para um filme lançado após 1960. O atributo `[ClassicMovie]` verifica o gênero primeiro e, se ele for um clássico, verificará a data de lançamento para ver se ela é posterior a 1960. Se ele tiver sido lançado após 1960, a validação falhará. O atributo aceita um parâmetro de inteiro que representa o ano que pode ser usado para validar os dados. Capture o valor do parâmetro no construtor do atributo, conforme mostrado aqui:

```
public class ClassicMovieAttribute : ValidationAttribute, IClientModelValidator
{
    private int _year;

    public ClassicMovieAttribute(int year)
    {
        _year = year;
    }

    protected override ValidationResult IsValid(
        object value, ValidationContext validationContext)
    {
        Movie movie = (Movie)validationContext.ObjectInstance;

        if (movie.Genre == Genre.Classic && movie.ReleaseDate.Year > _year)
        {
            return new ValidationResult(GetErrorMessage());
        }

        return ValidationResult.Success;
    }
}
```

A variável `movie` acima representa um objeto `Movie` que contém os dados do envio do formulário a serem validados. Nesse caso, o código de validação verifica a data e o gênero no método `IsValid` da classe `ClassicMovieAttribute` de acordo com as regras. Após a validação bem-sucedida, o `IsValid` retorna um código `ValidationResult.Success`. Quando a validação falha, um `ValidationResult` com uma mensagem erro será retornado:

```
private string GetErrorMessage()
{
    return $"Classic movies must have a release year earlier than {_year}.";
```

Quando um usuário modificar o campo `Genre` e enviar o formulário, o método `IsValid` da

`ClassicMovieAttribute` verificará se o filme é um clássico. Como qualquer atributo interno, aplique o atributo `ClassicMovieAttribute` a uma propriedade como `ReleaseDate` para garantir que a validação ocorra, conforme mostrado no exemplo de código anterior. Como o exemplo funciona apenas com tipos `Movie`, a melhor opção é usar `IValidatableObject`, conforme mostrado no parágrafo a seguir.

Como alternativa, esse mesmo código pode ser colocado no modelo implementando o método `Validate` na interface `IValidatableObject`. Embora os atributos de validação personalizados funcionem bem para validar propriedades individuais, a implementação de `IValidatableObject` pode ser usada para implementar a validação no nível da classe como visto aqui.

```
public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
{
    if (Genre == Genre.Classic && ReleaseDate.Year > _classicYear)
    {
        yield return new ValidationResult(
            $"Classic movies must have a release year earlier than {_classicYear}." ,
            new[] { "ReleaseDate" });
    }
}
```

## Validação do lado do cliente

A validação do lado do cliente é uma ótima conveniência para usuários. Ela economiza tempo que de outra forma seria gasto aguardando uma viagem de ida e volta para o servidor. Em termos de negócios, até mesmo algumas frações de segundos multiplicados por centenas de vezes por dia chega a ser muito tempo, despesa e frustração. A validação imediata e simples permite que os usuários trabalhem com mais eficiência e façam contribuições e produzam entradas e saídas de melhor qualidade.

É necessário ter uma exibição com as referências de script JavaScript corretas em vigor para a validação do lado do cliente para que ela funcione como visto aqui.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
```

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-validate/1.17.0/jquery.validate.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-validation-unobtrusive/3.2.11/jquery.validate.unobtrusive.min.js"></script>
```

O script do [jQuery Unobtrusive Validation](#) é uma biblioteca de front-end personalizada da Microsoft que se baseia no popular plug-in [jQuery Validate](#). Sem o jQuery Unobtrusive Validation, você precisará codificar a mesma lógica de validação em dois locais: uma vez nos atributos de validação do lado do servidor nas propriedades do modelo e, em seguida, novamente nos scripts do lado do cliente (os exemplos para o método `validate()` do jQuery Validate mostram a complexidade que isso pode gerar). Em vez disso, os [Auxiliares de Marca](#) e os [Auxiliares HTML](#) do MVC podem usar os atributos de validação e os metadados de tipo das propriedades do modelo para renderizar [atributos de dados](#) HTML 5 nos elementos de formulário que precisam de validação. O MVC gera os atributos `data-` para atributos internos e personalizados. O jQuery Unobtrusive Validation analisa os atributos `data-` e passa a lógica para o jQuery Validate, "copiando" efetivamente a lógica de validação do lado do servidor para o cliente. Exiba erros de validação no cliente usando os auxiliares de marca relevantes, conforme mostrado aqui:

```

<div class="form-group">
    <label asp-for="ReleaseDate" class="col-md-2 control-label"></label>
    <div class="col-md-10">
        <input asp-for="ReleaseDate" class="form-control" />
        <span asp-validation-for="ReleaseDate" class="text-danger"></span>
    </div>
</div>

```

Os auxiliares de marca acima renderizam o HTML abaixo. Observe que os atributos `data-` na saída HTML correspondem aos atributos de validação da propriedade `ReleaseDate`. O atributo `data-val-required` abaixo contém uma mensagem de erro a ser exibida se o usuário não preencher o campo de data de lançamento. O jQuery Unobtrusive Validation passa esse valor para o método `required()` do jQuery Validate, que, por sua vez, exibe essa mensagem no elemento `<span>` complementar.

```

<form action="/Movies/Create" method="post">
    <div class="form-horizontal">
        <h4>Movie</h4>
        <div class="text-danger"></div>
        <div class="form-group">
            <label class="col-md-2 control-label" for="ReleaseDate">ReleaseDate</label>
            <div class="col-md-10">
                <input class="form-control" type="datetime"
                    data-val="true" data-val-required="The ReleaseDate field is required."
                    id="ReleaseDate" name="ReleaseDate" value="" />
                <span class="text-danger field-validation-valid"
                    data-valmsg-for="ReleaseDate" data-valmsg-replace="true"></span>
            </div>
        </div>
    </div>
</form>

```

A validação do lado do cliente impede o envio até que o formulário seja válido. O botão Enviar executa o JavaScript que envia o formulário ou exibe mensagens de erro.

O MVC determina os valores de atributo de tipo com base no tipo de dados .NET de uma propriedade, possivelmente, substituído com o uso de atributos `[DataType]`. O atributo `[DataType]` base não faz nenhuma validação real do lado do servidor. Os navegadores escolhem suas próprias mensagens de erro e exibem esses erros da maneira desejada. No entanto, o pacote Unobtrusive Validation do jQuery pode substituir as mensagens e exibi-las de forma consistente com as outras. Isso ocorre de forma mais evidente quando os usuários aplicam subclasses `[DataType]`, como `[EmailAddress]`.

### Adicionar validação a formulários dinâmicos

Como o jQuery Unobtrusive Validation passa parâmetros e a lógica de validação para o jQuery Validate quando a página é carregada pela primeira vez, os formulários gerados dinamicamente não exibem a validação de forma automática. Em vez disso, é necessário instruir o jQuery Unobtrusive Validation a analisar o formulário dinâmico imediatamente depois de criá-lo. Por exemplo, o código abaixo mostra como você pode configurar a validação do lado do cliente em um formulário adicionado por meio do AJAX.

```

$.get({
    url: "https://url/that/returns/a/form",
    dataType: "html",
    error: function(jqXHR, textStatus, errorThrown) {
        alert(textStatus + ": Couldn't add form. " + errorThrown);
    },
    success: function(newFormHTML) {
        var container = document.getElementById("form-container");
        container.insertAdjacentHTML("beforeend", newFormHTML);
        var forms = container.getElementsByTagName("form");
        var newForm = forms[forms.length - 1];
        $.validator.unobtrusive.parse(newForm);
    }
})

```

O método `$.validator.unobtrusive.parse()` aceita um seletor do jQuery para um argumento seu. Esse método instrui o jQuery Unobtrusive Validation a analisar os atributos `data-` de formulários nesse seletor. Os valores desses atributos são então passados para o plug-in jQuery Validate, de modo que o formulário exiba as regras de validação do lado do cliente desejadas.

### Adicionar validação a controles dinâmicos

Também atualize as regras de validação em um formulário quando controles individuais, como `<input/>`s e `<select/>`s, são gerados dinamicamente. Não é possível passar seletores para esses elementos para o método `parse()` diretamente, porque o formulário adjacente já foi analisado e não será atualizado. Em vez disso, primeiro remova os dados de validação existentes e, em seguida, analise novamente todo o formulário, conforme mostrado abaixo:

```

$.get({
    url: "https://url/that/returns/a/control",
    dataType: "html",
    error: function(jqXHR, textStatus, errorThrown) {
        alert(textStatus + ": Couldn't add control. " + errorThrown);
    },
    success: function(newInputHTML) {
        var form = document.getElementById("my-form");
        form.insertAdjacentHTML("beforeend", newInputHTML);
        $(form).removeData("validator") // Added by jQuery Validate
            .removeData("unobtrusiveValidation"); // Added by jQuery Unobtrusive
        Validation
        $.validator.unobtrusive.parse(form);
    }
})

```

## IClientModelValidator

Você pode criar uma lógica do lado do cliente para o atributo personalizado, e o [unobtrusive validation](#), que cria um adaptador para a [validação do jQuery](#), será executado no cliente automaticamente como parte da validação. A primeira etapa é controlar quais atributos de dados são adicionados, pela implementação da interface `IClientModelValidator`, conforme mostrado aqui:

```

public void AddValidation(ClientModelValidationContext context)
{
    if (context == null)
    {
        throw new ArgumentNullException(nameof(context));
    }

    MergeAttribute(context.Attributes, "data-val", "true");
    MergeAttribute(context.Attributes, "data-val-classicmovie", GetErrorMessage());

    var year = _year.ToString(CultureInfo.InvariantCulture);
    MergeAttribute(context.Attributes, "data-val-classicmovie-year", year);
}

```

Atributos que implementam essa interface podem adicionar atributos HTML a campos gerados. O exame da saída do elemento `ReleaseDate` revela um HTML semelhante ao exemplo anterior, exceto que agora há um atributo `data-val-classicmovie` que foi definido no método `AddValidation` de `IClientModelValidator`.

```

<input class="form-control" type="datetime"
      data-val="true"
      data-val-classicmovie="Classic movies must have a release year earlier than 1960."
      data-val-classicmovie-year="1960"
      data-val-required="The ReleaseDate field is required."
      id="ReleaseDate" name="ReleaseDate" value="" />

```

A validação não invasiva usa os dados nos atributos `data-` para exibir mensagens de erro. No entanto, o jQuery não reconhece regras ou mensagens até que você adicione-as ao objeto `validator` do jQuery. Isso é mostrado no exemplo a seguir, que adiciona um método de validação de cliente `classicmovie` personalizado ao objeto `validator` do jQuery. Para obter uma explicação sobre o método `unobtrusive.adapters.add`, confira [Unobtrusive Client Validation no ASP.NET MVC](#).

```

$.validator.addMethod('classicmovie',
    function (value, element, params) {
        // Get element value. Classic genre has value '0'.
        var genre = $(params[0]).val(),
            year = params[1],
            date = new Date(value);
        if (genre && genre.length > 0 && genre[0] === '0') {
            // Since this is a classic movie, invalid if release date is after given year.
            return date.getFullYear() <= year;
        }

        return true;
});

$.validator.unobtrusive.adapters.add('classicmovie',
    ['year'],
    function (options) {
        var element = $(options.form).find('select#Genre')[0];
        options.rules['classicmovie'] = [element, parseInt(options.params['year'])];
        options.messages['classicmovie'] = options.message;
    });

```

Com o código anterior, o método `classicmovie` executa a validação do lado do cliente na data de lançamento do filme. A mensagem de erro é exibida se o método retornar `false`.

## Validação remota

A validação remota é um ótimo recurso a ser usado quando você precisa validar os dados no cliente em relação aos dados no servidor. Por exemplo, o aplicativo pode precisar verificar se um email ou nome de usuário já está em uso e precisa consultar uma grande quantidade de dados para fazer isso. O download de conjuntos de dados grandes para validar um ou alguns campos consome muitos recursos. Isso também pode expor informações confidenciais. Uma alternativa é fazer uma solicitação de ida e volta para validar um campo.

Implemente a validação remota em um processo de duas etapas. Primeiro, é necessário anotar o modelo com o atributo `[Remote]`. O atributo `[Remote]` aceita várias sobrecargas que podem ser usadas para direcionar o JavaScript do lado do cliente para o código apropriado a ser chamado. O exemplo abaixo aponta para o método de ação `VerifyEmail` do controlador `Users`.

```
[Remote(action: "VerifyEmail", controller: "Users")]
public string Email { get; set; }
```

A segunda etapa é colocar o código de validação no método de ação correspondente, conforme definido no atributo `[Remote]`. De acordo com a documentação do método jQuery `Validateremoto`, a resposta do servidor deve ser uma cadeia JSON que seja:

- `"true"` para elementos válidos.
- `"false"`, `undefined` ou `null` para elementos inválidos, usando a mensagem de erro padrão.

Se a resposta do servidor for uma cadeia de caracteres (por exemplo, `"That name is already taken, try peter123 instead"`), a cadeia de caracteres é exibida como uma mensagem de erro personalizada no lugar da cadeia de caracteres padrão.

A definição do método `VerifyEmail` segue essas regras, conforme mostrado abaixo. Ela retorna uma mensagem de erro de validação se o email já está sendo usado ou `true` se o email está livre e encapsula o resultado em um objeto `JsonResult`. O lado do cliente pode então usar o valor retornado para continuar ou exibir o erro, se necessário.

```
[AcceptVerbs("Get", "Post")]
public IActionResult VerifyEmail(string email)
{
    if (!_userRepository.VerifyEmail(email))
    {
        return Json($"Email {email} is already in use.");
    }

    return Json(true);
}
```

Agora, quando os usuários inserem um email, o JavaScript na exibição faz uma chamada remota para ver se o email já está sendo usado e, em caso afirmativo, exibe a mensagem de erro. Caso contrário, o usuário pode enviar o formulário como de costume.

A propriedade `AdditionalFields` do atributo `[Remote]` é útil para validação de combinações de campos em relação aos dados no servidor. Por exemplo, se o modelo `User` acima tiver duas propriedades adicionais chamadas `FirstName` e `LastName`, é recomendável verificar se não há usuários que já têm esse par de nomes. Defina as novas propriedades, conforme mostrado no seguinte código:

```
[Remote(action: "VerifyName", controller: "Users", AdditionalFields = nameof(LastName))]
public string FirstName { get; set; }
[Remote(action: "VerifyName", controller: "Users", AdditionalFields = nameof(FirstName))]
public string LastName { get; set; }
```

`AdditionalFields` poderia ter sido definido de forma explícita com as cadeias de caracteres `"FirstName"` e `"LastName"`, mas o uso do operador `nameof` dessa forma, simplifica a refatoração posterior. Em seguida, o método de ação para executar a validação precisa aceitar dois argumentos, um para o valor de `FirstName` e outro para o valor de `LastName`.

```
[AcceptVerbs("Get", "Post")]
public IActionResult VerifyName(string firstName, string lastName)
{
    if (!userRepository.VerifyName(firstName, lastName))
    {
        return Json(data: $"A user named {firstName} {lastName} already exists.");
    }

    return Json(data: true);
}
```

Agora, quando os usuários inserem um nome e sobrenome, o JavaScript:

- Faz uma chamada remota para ver se esse par de nomes já está sendo usado.
- Se o par já está sendo usado, uma mensagem de erro é exibida.
- Caso contrário, o usuário pode enviar o formulário.

Se você precisa validar dois ou mais campos adicionais com o atributo `[Remote]`, forneça-os como uma lista delimitada por vírgula. Por exemplo, para adicionar uma propriedade `MiddleName` ao modelo, defina o atributo `[Remote]`, conforme mostrado no seguinte código:

```
[Remote(action: "VerifyName", controller: "Users", AdditionalFields = nameof(FirstName) + ","
+ nameof(LastName))]
public string MiddleName { get; set; }
```

`AdditionalFields`, como todos os argumentos de atributo, deve ser uma expressão de constante. Portanto, você não deve usar uma [cadeia de caracteres interpolada](#) ou chamar `Join` para inicializar `AdditionalFields`. Para cada campo adicional adicionado ao atributo `[Remote]`, é necessário adicionar outro argumento ao método de ação do controlador correspondente.

# Referência da sintaxe Razor para ASP.NET Core

18/01/2019 • 25 minutes to read • [Edit Online](#)

Por [Rick Anderson](#), [Luke Latham](#), [Taylor Mullen](#) e [Dan Vicarel](#)

Razor é uma sintaxe de marcação para inserir código baseado em servidor em páginas da Web. A sintaxe Razor é composta pela marcação Razor, por C# e por HTML. Arquivos que contêm Razor geralmente têm a extensão de arquivo `.cshtml`.

## Renderizando HTML

A linguagem padrão do Razor padrão é o HTML. Renderizar HTML da marcação Razor não é diferente de renderizar HTML de um arquivo HTML. A marcação HTML em arquivos Razor `.cshtml` é renderizada pelo servidor inalterado.

## Sintaxe Razor

O Razor dá suporte a C# e usa o símbolo `@` para fazer a transição de HTML para C#. O Razor avalia expressões em C# e as renderiza na saída HTML.

Quando um símbolo `@` é seguido por uma [palavra-chave reservada do Razor](#), ele faz a transição para marcação específica do Razor. Caso contrário, ele faz a transição para C# simples.

Para fazer o escape de um símbolo `@` na marcação Razor, use um segundo símbolo `@`:

```
<p>@@Username</p>
```

O código é renderizado em HTML com um único símbolo `@`:

```
<p>@Username</p>
```

Conteúdo e atributos HTML que contêm endereços de email não tratam o símbolo `@` como um caractere de transição. Os endereços de email no exemplo a seguir não são alterados pela análise do Razor:

```
<a href="mailto:Support@contoso.com">Support@contoso.com</a>
```

## Expressões Razor implícitas

Expressões Razor implícitas são iniciadas por `@` seguido do código C#:

```
<p>@DateTime.Now</p>
<p>@DateTime.IsLeapYear(2016)</p>
```

Com exceção da palavra-chave C# `await`, expressões implícitas não devem conter espaços. Se a instrução C# tiver uma terminação clara, espaços podem ser misturados:

```
<p>@await DoSomething("hello", "world")</p>
```

Expressões implícitas **não podem** conter elementos genéricos de C#, pois caracteres dentro de colchetes (`<>`) são interpretados como uma marca HTML. O código a seguir é **inválido**:

```
<p>@GenericMethod<int>()</p>
```

O código anterior gera um erro de compilador semelhante a um dos seguintes:

- O elemento "int" não foi fechado. Todos os elementos devem ter fechamento automático ou ter uma marca de fim correspondente.
- Não é possível converter o grupo de métodos "GenericMethod" em um "object" de tipo não delegado. Você pretendia invocar o método?

Chamadas de método genérico devem ser encapsuladas em uma [expressão Razor explícita](#) ou em um [bloco de código Razor](#).

## Expressões Razor explícitas

Expressões Razor explícitas consistem de um símbolo `@` com parênteses equilibradas. Para renderizar a hora da última semana, a seguinte marcação Razor é usada:

```
<p>Last week this time: @(DateTime.Now - TimeSpan.FromDays(7))</p>
```

Qualquer conteúdo dentro dos parênteses `@()` é avaliado e renderizado para a saída.

Expressões implícitas, descritas na seção anterior, geralmente não podem conter espaços. No código a seguir, uma semana não é subtraída da hora atual:

```
<p>Last week: @DateTime.Now - TimeSpan.FromDays(7)</p>
```

O código renderiza o HTML a seguir:

```
<p>Last week: 7/7/2016 4:39:52 PM - TimeSpan.FromDays(7)</p>
```

Expressões explícitas podem ser usadas para concatenar texto com um resultado de expressão:

```
@{  
    var joe = new Person("Joe", 33);  
}  
  
<p>Age@{(joe.Age)}</p>
```

Sem a expressão explícita, `<p>Age@joe.Age</p>` é tratado como um endereço de email e `<p>Age@joe.Age</p>` é renderizado. Quando escrito como uma expressão explícita, `<p>Age33</p>` é renderizado.

Expressões explícitas podem ser usadas para renderizar a saída de métodos genéricos em arquivos `.cshtml`. A marcação a seguir mostra como corrigir o erro mostrado anteriormente causado pelos colchetes de um C# genérico. O código é escrito como uma expressão explícita:

```
<p>@{GenericMethod<int>}()</p>
```

## Codificação de expressão

Expressões em C# que são avaliadas como uma cadeia de caracteres estão codificadas em HTML. Expressões em C# que são avaliadas como `IHtmlContent` são renderizadas diretamente por meio `IHtmlContent.WriteTo`. Expressões em C# que não são avaliadas como `IHtmlContent` são convertidas em uma cadeia de caracteres por `ToString` e codificadas antes que sejam renderizadas.

```
@("<span>Hello World</span>")
```

O código renderiza o HTML a seguir:

```
&lt;span&gt;Hello World&lt;/span&gt;
```

O HTML é mostrado no navegador como:

```
<span>Hello World</span>
```

A saída `HtmlHelper.Raw` não é codificada, mas renderizada como marcação HTML.

#### WARNING

Usar `HtmlHelper.Raw` em uma entrada do usuário que não está limpa é um risco de segurança. A entrada do usuário pode conter JavaScript mal-intencionado ou outras formas de exploração. Limpar a entrada do usuário é difícil. Evite usar `HtmlHelper.Raw` com a entrada do usuário.

```
@Html.Raw("<span>Hello World</span>")
```

O código renderiza o HTML a seguir:

```
<span>Hello World</span>
```

## Blocos de código Razor

Blocos de código Razor são iniciados por `@` e delimitados por `{}`. Diferente das expressões, o código C# dentro de blocos de código não é renderizado. Blocos de código e expressões em uma exibição compartilham o mesmo escopo e são definidos em ordem:

```
@{  
    var quote = "The future depends on what you do today. - Mahatma Gandhi";  
}  
  
<p>@quote</p>  
  
#{@  
    quote = "Hate cannot drive out hate, only love can do that. - Martin Luther King, Jr.>";  
}  
  
<p>@quote</p>
```

O código renderiza o HTML a seguir:

```
<p>The future depends on what you do today. - Mahatma Gandhi</p>
<p>Hate cannot drive out hate, only love can do that. - Martin Luther King, Jr.</p>
```

## Transições implícitas

A linguagem padrão em um bloco de código é C#, mas a página Razor pode fazer a transição de volta para HTML:

```
@{
    var inCSharp = true;
    <p>Now in HTML, was in C# @inCSharp</p>
}
```

## Transição delimitada explícita

Para definir uma subseção de um bloco de código que deve renderizar HTML, circunde os caracteres para renderização com as marca Razor **<text>**:

```
@for (var i = 0; i < people.Length; i++)
{
    var person = people[i];
    <text>Name: @person.Name</text>
}
```

Use essa abordagem para renderizar HTML que não está circundado por uma marca HTML. Sem uma marca HTML ou Razor, ocorrerá um erro de tempo de execução do Razor.

A marca **<text>** é útil para controlar o espaço em branco ao renderizar conteúdo:

- Somente o conteúdo entre a marca **<text>** é renderizado.
- Não aparece nenhum espaço em branco antes ou depois da marca **<text>** na saída HTML.

## Transição de linha explícita com @:

Para renderizar o restante de uma linha inteira como HTML dentro de um bloco de código, use a sintaxe **@:** :

```
@for (var i = 0; i < people.Length; i++)
{
    var person = people[i];
    @:Name: @person.Name
}
```

Sem o **@:** no código, será gerado um erro de tempo de execução do Razor.

Aviso: Caracteres **@** extras em um arquivo Razor podem causar erros do compilador em instruções mais adiante no bloco. Esses erros do compilador podem ser difíceis de entender porque o erro real ocorre antes do erro relatado. Esse erro é comum após combinar várias expressões implícitas/explícitas em um bloco de código único.

# Estruturas de controle

Estruturas de controle são uma extensão dos blocos de código. Todos os aspectos dos blocos de código (transição para marcação, C# embutido) também se aplicam às seguintes estruturas:

## Condicionais @if, else if, else e @switch

**@if** controla quando o código é executado:

```
@if (value % 2 == 0)
{
    <p>The value was even.</p>
}
```

`else` e `else if` não exigem o símbolo `@`:

```
@if (value % 2 == 0)
{
    <p>The value was even.</p>
}
else if (value >= 1337)
{
    <p>The value is large.</p>
}
else
{
    <p>The value is odd and small.</p>
}
```

A marcação a seguir mostra como usar uma instrução switch:

```
@switch (value)
{
    case 1:
        <p>The value is 1!</p>
        break;
    case 1337:
        <p>Your number is 1337!</p>
        break;
    default:
        <p>Your number wasn't 1 or 1337.</p>
        break;
}
```

### Loop de `@for`, `@foreach`, `@while` e `@do while`

O HTML no modelo pode ser renderizado com instruções de controle em loop. Para renderizar uma lista de pessoas:

```
@{
    var people = new Person[]
    {
        new Person("Weston", 33),
        new Person("Johnathon", 41),
        ...
    };
}
```

Há suporte para as seguintes instruções em loop:

`@for`

```
@for (var i = 0; i < people.Length; i++)
{
    var person = people[i];
    <p>Name: @person.Name</p>
    <p>Age: @person.Age</p>
}
```

### @foreach

```
@foreach (var person in people)
{
    <p>Name: @person.Name</p>
    <p>Age: @person.Age</p>
}
```

### @while

```
@{ var i = 0; }
@while (i < people.Length)
{
    var person = people[i];
    <p>Name: @person.Name</p>
    <p>Age: @person.Age</p>

    i++;
}
```

### @do while

```
@{ var i = 0; }
@do
{
    var person = people[i];
    <p>Name: @person.Name</p>
    <p>Age: @person.Age</p>

    i++;
} while (i < people.Length);
```

## @using composto

Em C#, uma instrução `using` é usada para garantir que um objeto seja descartado. No Razor, o mesmo mecanismo é usado para criar Auxiliares HTML que têm conteúdo adicional. No código a seguir, os Auxiliares HTML renderizam uma marca de formulário com a instrução `@using`:

```
@using (Html.BeginForm())
{
    <div>
        email:
        <input type="email" id="Email" value="">
        <button>Register</button>
    </div>
}
```

Ações no nível de escopo podem ser executadas com [Auxiliares de marca](#).

## @try, catch, finally

O tratamento de exceções é semelhante ao de C#:

```
@try
{
    throw new InvalidOperationException("You did something invalid.");
}
catch (Exception ex)
{
    <p>The exception message: @ex.Message</p>
}
finally
{
    <p>The finally statement.</p>
}
```

## @lock

O Razor tem a capacidade de proteger seções críticas com instruções de bloqueio:

```
@lock (SomeLock)
{
    // Do critical section work
}
```

## Comentários

O Razor dá suporte a comentários em C# e HTML:

```
@{
    /* C# comment */
    // Another C# comment
}
<!-- HTML comment -->
```

O código renderiza o HTML a seguir:

```
<!-- HTML comment -->
```

Comentários em Razor são removidos pelo servidor antes que a página da Web seja renderizada. O Razor usa `@* *@` para delimitar comentários. O código a seguir é comentado, de modo que o servidor não renderiza nenhuma marcação:

```
@*
{@
    /* C# comment */
    // Another C# comment
}
<!-- HTML comment -->
*@
```

## Diretivas

Diretivas de Razor são representadas por expressões implícitas com palavras-chave reservadas após o símbolo `@`. Uma diretiva geralmente altera o modo como uma exibição é analisada ou habilita uma funcionalidade diferente.

Compreender como o Razor gera código para uma exibição torna mais fácil entender como as diretivas funcionam.

```

@{
    var quote = "Getting old ain't for wimpy! - Anonymous";
}

<div>Quote of the Day: @quote</div>

```

O código gera uma classe semelhante à seguinte:

```

public class _Views_Something_cshtml : RazorPage<dynamic>
{
    public override async Task ExecuteAsync()
    {
        var output = "Getting old ain't for wimpy! - Anonymous";

        WriteLiteral("/r/n<div>Quote of the Day: ");
        Write(output);
        WriteLiteral("</div>");
    }
}

```

Mais adiante neste artigo, a seção [Inspecionar a classe do Razor C# gerada para uma exibição](#) explica como exibir essa classe gerada.

## @using

A diretiva `@using` adiciona a diretiva `using` de C# à exibição gerada:

```

@using System.IO
 @{
    var dir = Directory.GetCurrentDirectory();
}
<p>@dir</p>

```

## @model

A diretiva `@model` especifica o tipo do modelo passado para uma exibição:

```

@model TypeNameOfModel

```

Em um aplicativo do ASP.NET Core MCV criado com contas de usuário individuais, a exibição `Views/Account/Login.cshtml` contém a declaração de modelo a seguir:

```

@model LoginViewModel

```

A classe gerada herda de `RazorPage<dynamic>`:

```

public class _Views_Account_Login_cshtml : RazorPage<LoginViewModel>

```

O Razor expõe uma propriedade `Model` para acessar o modelo passado para a exibição:

```

<div>The Login Email: @Model.Email</div>

```

A diretiva `@model` especifica o tipo dessa propriedade. A diretiva especifica o `T` em `RazorPage<T>` da classe gerada da qual a exibição deriva. Se a diretiva `@model` não for especificada, a propriedade `Model` será do tipo `dynamic`. O valor do modelo é passado do controlador para a exibição. Para obter mais informações, confira

Modelos fortemente tipados e a @palavra-chave do modelo.

## @inherits

A diretiva `@inherits` fornece controle total da classe que a exibição herda:

```
@inherits TypeNameOfClassToInheritFrom
```

O código a seguir é um tipo de página Razor personalizado:

```
using Microsoft.AspNetCore.Mvc.Razor;

public abstract class CustomRazorPage<TModel> : RazorPage<TModel>
{
    public string CustomText { get; } = "Gardyloo! - A Scottish warning yelled from a window before
dumping a slop bucket on the street below.";
}
```

O `CustomText` é exibido em uma exibição:

```
@inherits CustomRazorPage<TModel>

<div>Custom text: @CustomText</div>
```

O código renderiza o HTML a seguir:

```
<div>Custom text: Gardyloo! - A Scottish warning yelled from a window before dumping a slop bucket on the
street below.</div>
```

`@model` e `@inherits` podem ser usados na mesma exibição. `@inherits` pode estar em um arquivo `_ViewImports.cshtml` que a exibição importa:

```
@inherits CustomRazorPage<TModel>
```

O código a seguir é um exemplo de exibição fortemente tipada:

```
@inherits CustomRazorPage<TModel>

<div>The Login Email: @Model.Email</div>
<div>Custom text: @CustomText</div>
```

Se "rick@contoso.com" for passado no modelo, a exibição gerará a seguinte marcação HTML:

```
<div>The Login Email: rick@contoso.com</div>
<div>Custom text: Gardyloo! - A Scottish warning yelled from a window before dumping a slop bucket on the
street below.</div>
```

## @inject

A diretiva `@inject` permite que a página do Razor injete um serviço do [contêiner de serviço](#) em uma exibição. Para obter mais informações, consulte [Injeção de dependência em exibições](#).

## @functions

A diretiva `@functions` permite que uma página Razor adicione um bloco de código C# a uma exibição:

```
@functions { // C# Code }
```

Por exemplo:

```
@functions {
    public string GetHello()
    {
        return "Hello";
    }
}

<div>From method: @GetHello()</div>
```

O código gera a seguinte marcação HTML:

```
<div>From method: Hello</div>
```

O código a seguir é a classe C# do Razor gerada:

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc.Razor;

public class _Views_Home_Test_cshtml : RazorPage<dynamic>
{
    // Functions placed between here
    public string GetHello()
    {
        return "Hello";
    }
    // And here.
#pragma warning disable 1998
    public override async Task ExecuteAsync()
    {
        WriteLiteral("\r\n<div>From method: ");
        Write(GetHello());
        WriteLiteral("</div>\r\n");
    }
#pragma warning restore 1998
```

## @section

A diretiva `@section` é usada em conjunto com o `layout` para permitir que as exibições renderizem conteúdo em partes diferentes da página HTML. Para obter mais informações, consulte [Seções](#).

## Representantes do Razor com modelo

Os modelos Razor permitem que você defina um snippet da interface do usuário com o seguinte formato:

```
@<tag>...</tag>
```

O exemplo a seguir ilustra como especificar um delegado do Razor com modelo como um `Func<T,TResult>`. O `tipo dinâmico` é especificado para o parâmetro do método encapsulado pelo delegado. Um `tipo de objeto` é especificado como o valor retornado do delegado. O modelo é usado com uma `List<T>` de `Pet` que tem uma propriedade `Name`.

```
public class Pet
{
    public string Name { get; set; }
}
```

```
@{
    Func<dynamic, object> petTemplate = @<p>You have a pet named <strong>@item.Name</strong>.</p>;
}

var pets = new List<Pet>
{
    new Pet { Name = "Rin Tin Tin" },
    new Pet { Name = "Mr. Bigglesworth" },
    new Pet { Name = "K-9" }
};
```

O modelo é renderizado com `pets` fornecido por uma instrução `foreach`:

```
@foreach (var pet in pets)
{
    @petTemplate(pet)
}
```

Saída renderizada:

```
<p>You have a pet named <strong>Rin Tin Tin</strong>.</p>
<p>You have a pet named <strong>Mr. Bigglesworth</strong>.</p>
<p>You have a pet named <strong>K-9</strong>.</p>
```

Você também pode fornecer um modelo Razor embutido como um argumento para um método. No exemplo a seguir, o método `Repeat` recebe um modelo Razor. O método usa o modelo para produzir o conteúdo HTML com repetições de itens fornecidos em uma lista:

```
@using Microsoft.AspNetCore.Html

@functions {
    public static IHtmlContent Repeat(IEnumerable<dynamic> items, int times,
        Func<dynamic, IHtmlContent> template)
    {
        var html = new HtmlContentBuilder();

        foreach (var item in items)
        {
            for (var i = 0; i < times; i++)
            {
                html.AppendHtml(template(item));
            }
        }

        return html;
    }
}
```

Usando a lista de animais de estimação do exemplo anterior, o método `Repeat` é chamado com:

- `List<T>` de `Pet`.
- Número de vezes que deve ser repetido cada animal de estimação.

- Modelo embutido a ser usado para os itens da lista de uma lista não ordenada.

```
<ul>
    @Repeat(pets, 3, @<li>@item.Name</li>)
</ul>
```

Saída renderizada:

```
<ul>
    <li>Rin Tin Tin</li>
    <li>Rin Tin Tin</li>
    <li>Rin Tin Tin</li>
    <li>Mr. Bigglesworth</li>
    <li>Mr. Bigglesworth</li>
    <li>Mr. Bigglesworth</li>
    <li>K-9</li>
    <li>K-9</li>
    <li>K-9</li>
</ul>
```

## Auxiliares de Marca

Há três diretivas que relacionadas aos [Auxiliares de marca](#).

DIRETIVA	FUNÇÃO
<code>@addTagHelper</code>	Disponibiliza os Auxiliares de marca para uma exibição.
<code>@removeTagHelper</code>	Remove os Auxiliares de marca adicionados anteriormente de uma exibição.
<code>@tagHelperPrefix</code>	Especifica um prefixo de marca para habilitar o suporte do Auxiliar de marca e tornar explícito o uso do Auxiliar de marca.

## Palavras-chave reservadas ao Razor

### Palavras-chave do Razor

- page (requer o ASP.NET Core 2.0 e posteriores)
- namespace
- funções
- herda
- modelo
- section
- helper (atualmente sem suporte do ASP.NET Core)

Palavras-chave do Razor têm o escape feito com `@(Razor Keyword)` (por exemplo, `@(functions)`).

### Palavras-chave do Razor em C#

- case
- do
- default
- for

- foreach
- if
- else
- bloqueio
- switch
- try
- catch
- finally
- using
- while

Palavras-chave do Razor em C# precisam ter o escape duplo com `@@@C# Razor Keyword` (por exemplo, `@@@case`). O primeiro `@` faz o escape do analisador Razor. O segundo `@` faz o escape do analisador C#.

#### **Palavras-chave reservadas não usadas pelo Razor**

- classe

## Inspecionar a classe do Razor C# gerada para uma exibição

Com o SDK do .NET Core 2.1 ou posterior, o [SDK do Razor](#) lida com a compilação de arquivos do Razor. Ao compilar um projeto, o SDK do Razor gera um diretório `obj/<configuração_de_build>/<moniker_da_estrutura_de_destino>/Razor` na raiz do projeto. A estrutura de diretórios dentro do diretório do Razor espelha a estrutura de diretórios do projeto.

Considere a seguinte estrutura de diretórios em um projeto do Razor Pages ASP.NET Core 2.1 direcionado ao .NET Core 2.1:

- **Areas/**
  - **Admin/**
  - **Pages/**
    - *Index.cshtml*
    - *Index.cshtml.cs*
- **Pages/**
  - **Shared/**
    - *\_Layout.cshtml*
    - *\_ViewImports.cshtml*
    - *\_ViewStart.cshtml*
    - *Index.cshtml*
    - *Index.cshtml.cs*

A criação do projeto na configuração de *Depuração* produz o seguinte diretório `obj/`:

- **obj/**
  - **Debug/**
  - **netcoreapp2.1/**
    - **Razor/**
      - **Areas/**
      - **Admin/**
      - **Pages/**
        - *Index.g.cshtml.cs*
    - **Pages/**

- o **Shared/**
  - o *\_Layout.g.cshtml.cs*
  - o *\_ViewImports.g.cshtml.cs*
  - o *\_ViewStart.g.cshtml.cs*
  - o *Index.g.cshtml.cs*

Para exibir a classe gerada para *Pages/Index.cshtml*, abra *obj/Debug/netcoreapp2.1/Razor/Pages/Index.g.cshtml.cs*.

Adicione a seguinte classe ao projeto do ASP.NET Core MVC:

```
using Microsoft.AspNetCore.Mvc.Razor.Extensions;
using Microsoft.AspNetCore.Razor.Language;

public class CustomTemplateEngine : MvcRazorTemplateEngine
{
    public CustomTemplateEngine(RazorEngine engine, RazorProject project)
        : base(engine, project)
    {
    }

    public override RazorCSharpDocument GenerateCode(RazorCodeDocument codeDocument)
    {
        var csharpDocument = base.GenerateCode(codeDocument);
        var generatedCode = csharpDocument.GeneratedCode;

        // Look at generatedCode

        return csharpDocument;
    }
}
```

Em `Startup.ConfigureServices`, substitua o `RazorTemplateEngine` adicionado pelo MVC pela classe `CustomTemplateEngine`:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddSingleton<RazorTemplateEngine, CustomTemplateEngine>();
}
```

Defina o ponto de interrupção `CustomTemplateEngine` na instrução `return csharpDocument;`. Quando a execução do programa for interrompida no ponto de interrupção, veja o valor de `generatedCode`.

The screenshot shows the 'Text Visualizer' window from a debugger. The 'Expression:' field is set to 'generatedCode'. The 'Value:' pane displays the generated C# code for a Razor view. The code includes directives like #line, context management (BeginContext, EndContext), and output writing (WriteLiteral). It also shows pragmas for warning restoration.

```
public override async Task ExecuteAsync()
{
#line 1 "/Views/Home/Contact3.cshtml"

    var output = "Hello World";

#line default
#line hidden

    BeginContext(40, 15, true);
    WriteLiteral("\r\n<div>Output: ");
    EndContext();
    BeginContext(56, 6, false);
#line 5 "/Views/Home/Contact3.cshtml"
    Write(output);

#line default
#line hidden
    EndContext();
    BeginContext(62, 6, true);
    WriteLiteral("</div>");
    EndContext();
}
#pragma warning restore 1998
```

## Pesquisas de exibição e diferenciação de maiúsculas e minúsculas

O mecanismo de exibição do Razor executa pesquisas que diferenciam maiúsculas de minúsculas para as exibições. No entanto, a pesquisa real é determinada pelo sistema de arquivos subjacente:

- Origem baseada em arquivo:
  - Em sistemas operacionais com sistemas de arquivos que não diferenciam maiúsculas e minúsculas (por exemplo, Windows), pesquisas no provedor de arquivos físico não diferenciam maiúsculas de minúsculas. Por exemplo, `return View("Test")` resulta em correspondências para `/Views/Home/Test.cshtml`, `/Views/home/test.cshtml` e qualquer outra variação de maiúsculas e minúsculas.
  - Em sistemas de arquivos que diferenciam maiúsculas de minúsculas (por exemplo, Linux, OSX e com `EmbeddedFileProvider`), as pesquisas diferenciam maiúsculas de minúsculas. Por exemplo, `return View("Test")` corresponde especificamente a `/Views/Home/Test.cshtml`.
- Exibições pré-compiladas: Com o ASP.NET Core 2.0 e posteriores, pesquisar em exibições pré-compiladas não diferencia maiúsculas de minúsculas em nenhum sistema operacional. O comportamento é idêntico ao comportamento do provedor de arquivos físico no Windows. Se duas exibições pré-compiladas diferirem apenas quanto ao padrão de maiúsculas e minúsculas, o resultado da pesquisa não será determinístico.

Os desenvolvedores são incentivados a fazer a correspondência entre as maiúsculas e minúsculas dos nomes dos arquivos e de diretórios com o uso de maiúsculas e minúsculas em:

- Nomes de área, controlador e ação.
- Páginas do Razor.

Fazer essa correspondência garante que as implantações encontrem suas exibições, independentemente do sistema de arquivos subjacente.

# Componentes de exibição no ASP.NET Core

06/02/2019 • 20 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Componentes da exibição

Os componentes de exibição são semelhantes às exibições parciais, mas são muito mais eficientes. Os componentes de exibição não usam o model binding e dependem apenas dos dados fornecidos durante uma chamada a eles. Este artigo foi elaborado com controladores e modos de exibição, mas os componentes da exibição também funcionam com o Razor Pages.

Um componente de exibição:

- Renderiza uma parte em vez de uma resposta inteira.
- Inclui os mesmos benefícios de capacidade de teste e separação de interesses e encontrados entre um controlador e uma exibição.
- Pode ter parâmetros e uma lógica de negócios.
- É geralmente invocado em uma página de layout.

Os componentes de exibição destinam-se a qualquer momento em que há uma lógica de renderização reutilizável muito complexa para uma exibição parcial, como:

- Menus de navegação dinâmica
- Nuvem de marcas (na qual ela consulta o banco de dados)
- Painel de logon
- Carrinho de compras
- Artigos publicados recentemente
- Conteúdo da barra lateral em um blog típico
- Um painel de logon que é renderizado em cada página e mostra os links para logoff ou logon, dependendo do estado de logon do usuário

Um componente de exibição consiste em duas partes: a classe (normalmente derivada de [ViewComponent](#)) e o resultado que ele retorna (normalmente, uma exibição). Assim como os controladores, um componente de exibição pode ser um POCO, mas a maioria dos desenvolvedores desejará aproveitar os métodos e as propriedades disponíveis com a derivação de [ViewComponent](#).

## Criando um componente de exibição

Esta seção contém os requisitos de alto nível para a criação de um componente de exibição. Mais adiante neste artigo, examinaremos cada etapa em detalhes e criaremos um componente de exibição.

### A classe de componente de exibição

Uma classe de componente de exibição pode ser criada por um dos seguintes:

- Derivação de [ViewComponent](#)
- Decoração de uma classe com o atributo `[ViewComponent]` ou derivação de uma classe com o atributo `[ViewComponent]`
- Criação de uma classe em que o nome termina com o sufixo [ViewComponent](#)

Assim como os controladores, os componentes de exibição precisam ser classes públicas, não aninhadas e não abstratas. O nome do componente de exibição é o nome da classe com o sufixo "ViewComponent" removido. Também pode ser especificado de forma explícita com a propriedade `ViewComponentAttribute.Name`.

Uma classe de componente de exibição:

- Dá suporte total à [injeção de dependência do construtor](#)
- Não participa do ciclo de vida do controlador, o que significa que não é possível usar [filtros](#) em um componente de exibição

## Métodos de componente de exibição

Um componente de exibição define sua lógica em um método `InvokeAsync` que retorna um `Task<IVViewComponentResult>` ou em um método `Invoke` síncrono que retorna um `IVViewComponentResult`. Os parâmetros são recebidos diretamente da invocação do componente de exibição, não do model binding. Um componente de exibição nunca manipula uma solicitação diretamente. Normalmente, um componente de exibição inicializa um modelo e passa-o para uma exibição chamando o método `View`. Em resumo, os métodos de componente de exibição:

- Definem um método `InvokeAsync` que retorna um `Task<IVViewComponentResult>` ou um método `Invoke` síncrono que retorna um `IVViewComponentResult`.
- Normalmente, inicializam um modelo e o passam para uma exibição chamando o método `ViewComponent View`.
- Os parâmetros são recebidos do método de chamada, não do HTTP. Não há nenhum model binding.
- Não são acessíveis diretamente como um ponto de extremidade HTTP. Eles são invocados no código (normalmente, em uma exibição). Um componente de exibição nunca manipula uma solicitação.
- São sobrecarregados na assinatura, em vez de nos detalhes da solicitação HTTP atual.

## Caminho de pesquisa de exibição

O tempo de execução pesquisa a exibição nos seguintes caminhos:

- `/Views/{Nome do Controlador}/Components/{Nome do Componente da Exibição}/{Nome da Exibição}`
- `/Views/Shared/Components/{Nome do Componente da Exibição}/{Nome da Exibição}`
- `/Pages/Shared/Components/{Nome do Componente da Exibição}/{Nome da Exibição}`

O caminho de pesquisa se aplica a projetos usando controladores + exibições e Razor Pages.

O nome de exibição padrão de um componente de exibição é *Default*, o que significa que o arquivo de exibição geralmente será nomeado `Default.cshtml`. Especifique outro nome de exibição ao criar o resultado do componente de exibição ou ao chamar o método `View`.

Recomendamos que você nomeie o arquivo de exibição `Default.cshtml` e use o caminho `Views/Shared/Components/{Nome do Componente da Exibição}/{Nome da Exibição}`. O componente de exibição `PriorityList` usado nesta amostra usa `Views/Shared/Components/PriorityList/Default.cshtml` como a exibição do componente de exibição.

## Invocando um componente de exibição

Para usar o componente de exibição, chame o seguinte em uma exibição:

```
@await Component.InvokeAsync("Name of view component", {Anonymous Type Containing Parameters})
```

Os parâmetros serão passados para o método `InvokeAsync`. O componente de exibição `PriorityList` desenvolvido no artigo é invocado por meio do arquivo de exibição `Views/ToDo/Index.cshtml`. A seguir, o

método `InvokeAsync` é chamado com dois parâmetros:

```
@await Component.InvokeAsync("PriorityList", new { maxPriority = 4, isDone = true })
```

## Invocando um componente de exibição como um Auxiliar de Marca

Para o ASP.NET Core 1.1 e superior, invoque um componente de exibição como um [Auxiliar de Marca](#):

```
<vc:priority-list max-priority="2" is-done="false">
</vc:priority-list>
```

Os parâmetros de classe e de método na formatação Pascal Case para Auxiliares de Marcas são convertidos em [kebab case](#). Para invocar um componente de exibição, o Auxiliar de Marca usa o elemento `<vc></vc>`. O componente de exibição é especificado da seguinte maneira:

```
<vc:[view-component-name]
  parameter1="parameter1 value"
  parameter2="parameter2 value">
</vc:[view-component-name]>
```

Para usar um componente de exibição como um Auxiliar de Marca, registre o assembly que contém o componente de exibição usando a diretiva `@addTagHelper`. Se seu componente de exibição estiver em um assembly chamado `MyWebApp`, adicione a seguinte diretiva ao arquivo `_ViewImports.cshtml`:

```
@addTagHelper *, MyWebApp
```

É possível registrar um componente de exibição como um Auxiliar de Marca a qualquer arquivo que referece o componente de exibição. Consulte [Gerenciando o escopo do Auxiliar de Marca](#) para obter mais informações sobre como registrar Auxiliares de Marca.

O método `InvokeAsync` usado neste tutorial:

```
@await Component.InvokeAsync("PriorityList", new { maxPriority = 4, isDone = true })
```

Na marcação do Auxiliar de Marca:

```
<vc:priority-list max-priority="2" is-done="false">
</vc:priority-list>
```

Na amostra acima, o componente de exibição `PriorityList` torna-se `priority-list`. Os parâmetros para o componente de exibição são passados como atributos em kebab case.

### Invocando um componente de exibição diretamente em um controlador

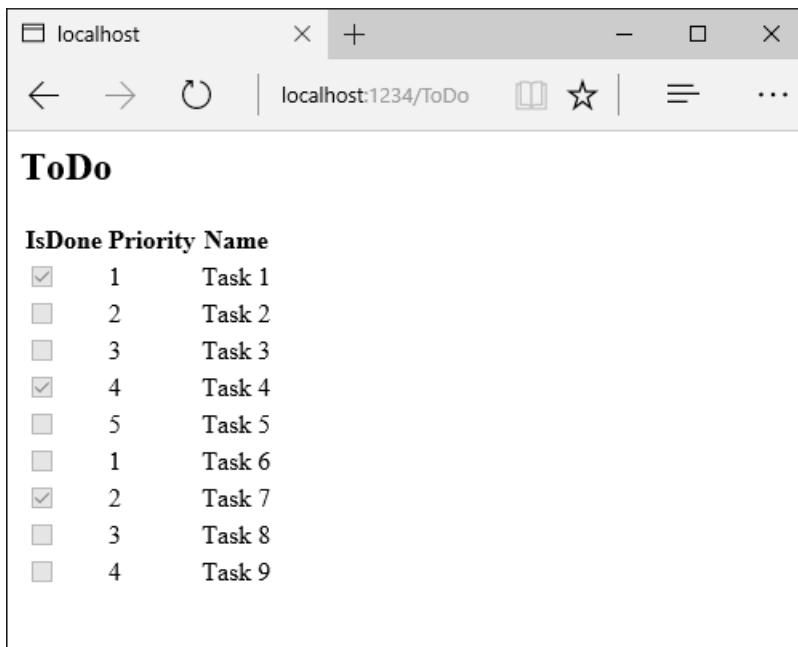
Os componentes de exibição normalmente são invocados em uma exibição, mas você pode invocá-los diretamente em um método do controlador. Embora os componentes de exibição não definam pontos de extremidade como controladores, você pode implementar com facilidade uma ação do controlador que retorna o conteúdo de um `ViewComponentResult`.

Neste exemplo, o componente de exibição é chamado diretamente no controlador:

```
public IActionResult IndexVC()
{
    return ViewComponent("PriorityList", new { maxPriority = 3, isDone = false });
}
```

## Passo a passo: Como criar um componente de exibição simples

Baixe, compile e teste o código inicial. É um projeto simples com um controlador `ToDo` que exibe uma lista de itens `ToDo`.



### Adicionar uma classe ViewComponent

Crie uma pasta `ViewComponents` e adicione a seguinte classe `PriorityListViewComponent`:

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using ViewComponentSample.Models;

namespace ViewComponentSample.ViewComponents
{
    public class PriorityListViewComponent : ViewComponent
    {
        private readonly ToDoContext db;

        public PriorityListViewComponent(ToDoContext context)
        {
            db = context;
        }

        public async Task<IViewComponentResult> InvokeAsync(
            int maxPriority, bool isDone)
        {
            var items = await GetItemsAsync(maxPriority, isDone);
            return View(items);
        }

        private Task<List<TodoItem>> GetItemsAsync(int maxPriority, bool isDone)
        {
            return db.ToDo.Where(x => x.IsDone == isDone &
                x.Priority <= maxPriority).ToListAsync();
        }
    }
}

```

Observações sobre o código:

- As classes de componente de exibição podem ser contidas em **qualquer** pasta do projeto.
- Como o nome da classe **PriorityListViewComponent** termina com o sufixo **ViewComponent**, o tempo de execução usará a cadeia de caracteres "PriorityList" ao referenciar o componente de classe em uma exibição. Explicarei isso mais detalhadamente mais adiante.
- O atributo `[ViewComponent]` pode alterar o nome usado para referenciar um componente de exibição. Por exemplo, poderíamos nomear a classe `xyz` e aplicar o atributo `ViewComponent`:

```

[ViewComponent(Name = "PriorityList")]
public class XYZ : ViewComponent

```

- O atributo `[ViewComponent]` acima instrui o seletor de componente de exibição a usar o nome `PriorityList` ao procurar as exibições associadas ao componente e a usar a cadeia de caracteres "PriorityList" ao referenciar o componente de classe em uma exibição. Explicarei isso mais detalhadamente mais adiante.
- O componente usa a [injeção de dependência](#) para disponibilizar o contexto de dados.
- `InvokeAsync` expõe um método que pode ser chamado em uma exibição e pode usar um número arbitrário de argumentos.
- O método `InvokeAsync` retorna o conjunto de itens `ToDo` que atendem aos parâmetros `isDone` e `maxPriority`.

### Criar a exibição do Razor do componente de exibição

- Crie a pasta `Views/Shared/Components`. Essa pasta **deve** nomeada `Components`.

- Crie a pasta `Views/Shared/Components/PriorityList`. Esse nome de pasta deve corresponder ao nome da classe do componente de exibição ou ao nome da classe menos o sufixo (se seguimos a convenção e usamos o sufixo `ViewComponent` no nome da classe). Se você usou o atributo `ViewComponent`, o nome da classe precisa corresponder à designação de atributo.

- Crie uma exibição do Razor `Views/Shared/Components/PriorityList/Default.cshtml`: [!code-cshtml]

A exibição do Razor usa uma lista de `TodoItem` e exibe-os. Se o método `InvokeAsync` do componente de exibição não passar o nome da exibição (como em nossa amostra), `Default` será usado como o nome da exibição, por convenção. Mais adiante no tutorial, mostrarei como passar o nome da exibição. Para substituir o estilo padrão de um controlador específico, adicione uma exibição à pasta de exibição específica do controlador (por exemplo, `Views/ToDo/Components/PriorityList/Default.cshtml`).

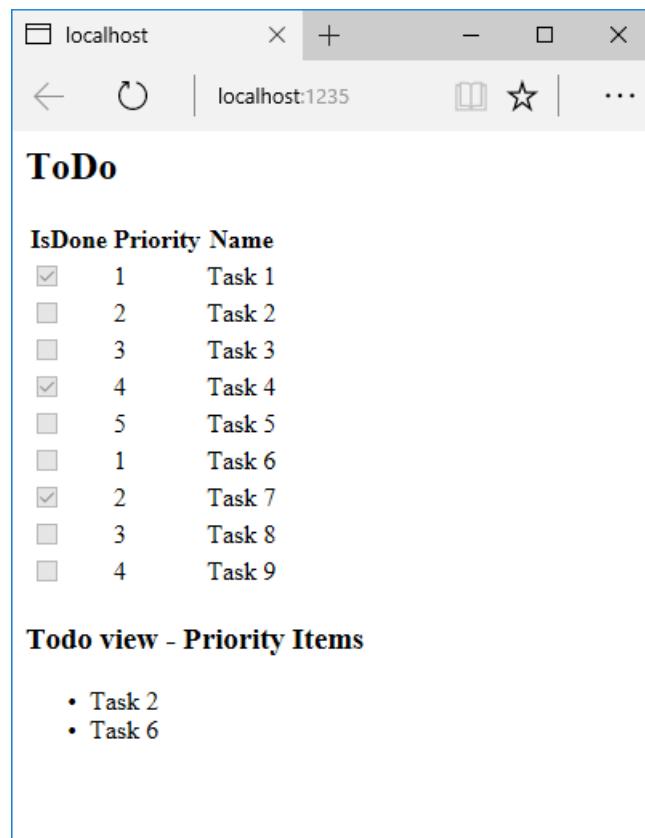
Se o componente de exibição for específico do controlador, adicione-o à pasta específica do controlador (`Views/ToDo/Components/PriorityList/Default.cshtml`).

- Adicione um `div` que contenha uma chamada para o componente da lista de prioridades à parte inferior do arquivo `Views/ToDo/index.cshtml`:

```
</table>
<div>
    @await Component.InvokeAsync("PriorityList", new { maxPriority = 2, isDone = false })
</div>
```

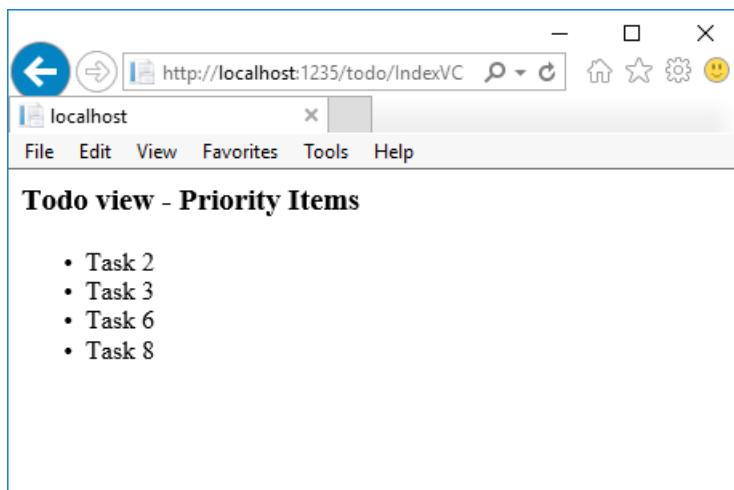
A marcação `@await Component.InvokeAsync` mostra a sintaxe para chamar componentes de exibição. O primeiro argumento é o nome do componente que queremos invocar ou chamar. Os parâmetros seguintes são passados para o componente. `InvokeAsync` pode usar um número arbitrário de argumentos.

Teste o aplicativo. A seguinte imagem mostra a lista ToDo e os itens de prioridade:



Também chame o componente de exibição diretamente no controlador:

```
public IActionResult IndexVC()
{
    return ViewComponent("PriorityList", new { maxPriority = 3, isDone = false });
}
```



### Especificando um nome de exibição

Um componente de exibição complexo pode precisar especificar uma exibição não padrão em algumas condições. O código a seguir mostra como especificar a exibição "PVC" no método `InvokeAsync`. Atualize o método `InvokeAsync` na classe `PriorityListViewComponent`.

```
public async Task<IViewComponentResult> InvokeAsync(
    int maxPriority, bool isDone)
{
    string MyView = "Default";
    // If asking for all completed tasks, render with the "PVC" view.
    if (maxPriority > 3 && isDone == true)
    {
        MyView = "PVC";
    }
    var items = await GetItemsAsync(maxPriority, isDone);
    return View(MyView, items);
}
```

Copie o arquivo `Views/Shared/Components/PriorityList/Default.cshtml` para uma exibição nomeada `Views/Shared/Components/PriorityList/PVC.cshtml`. Adicione um cabeçalho para indicar que a exibição PVC está sendo usada.

```
@model IEnumerable<ViewComponentSample.Models.TodoItem>

<h2> PVC Named Priority Component View</h2>
<h4>@ViewBag.PriorityMessage</h4>
<ul>
    @foreach (var todo in Model)
    {
        <li>@todo.Name</li>
    }
</ul>
```

Atualize `Views/ToDo/Index.cshtml`:

```
@await Component.InvokeAsync("PriorityList", new { maxPriority = 4, isDone = true })
```

Execute o aplicativo e verifique a exibição PVC.

IsDone	Priority	Name
<input checked="" type="checkbox"/>	1	Task 1
<input type="checkbox"/>	2	Task 2
<input type="checkbox"/>	3	Task 3
<input checked="" type="checkbox"/>	4	Task 4
<input type="checkbox"/>	5	Task 5
<input type="checkbox"/>	1	Task 6
<input checked="" type="checkbox"/>	2	Task 7
<input type="checkbox"/>	3	Task 8
<input type="checkbox"/>	4	Task 9

**PVC Named Priority Component View**

- Task 1
- Task 4
- Task 7

Se a exibição PVC não é renderizada, verifique se você está chamando o componente de exibição com uma prioridade igual a 4 ou superior.

#### Examinar o caminho de exibição

- Altere o parâmetro de prioridade para três ou menos para que a exibição de prioridade não seja retornada.
- Renomeie temporariamente `Views/ToDo/Components/PriorityList/Default.cshtml` como `1Default.cshtml`.
- Teste o aplicativo. Você obterá o seguinte erro:

```
An unhandled exception occurred while processing the request.  
InvalidOperationException: The view 'Components/PriorityList/Default' wasn't found. The following  
locations were searched:  
/Views/ToDo/Components/PriorityList/Default.cshtml  
/Views/Shared/Components/PriorityList/Default.cshtml  
EnsureSuccessful
```

- Copie `Views/ToDo/Components/PriorityList/1Default.cshtml` para `Views/Shared/Components/PriorityList/Default.cshtml`.
- Adicione uma marcação ao componente de exibição `ToDo Shared` para indicar que a exibição foi obtida da pasta `Shared`.
- Teste o componente de exibição **Shared**.

IsDone	Priority	Name
<input checked="" type="checkbox"/>	1	Task 1
<input type="checkbox"/>	2	Task 2
<input type="checkbox"/>	3	Task 3
<input checked="" type="checkbox"/>	4	Task 4
<input type="checkbox"/>	5	Task 5
<input type="checkbox"/>	1	Task 6
<input checked="" type="checkbox"/>	2	Task 7
<input type="checkbox"/>	3	Task 8
<input type="checkbox"/>	4	Task 9

**Shared**

- Task 1
- Task 7

### Evitando cadeias de caracteres mágicas

Se deseja obter segurança em tempo de compilação, substitua o nome do componente de exibição embutido em código pelo nome da classe. Crie o componente de exibição sem o sufixo "ViewComponent":

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using ViewComponentSample.Models;

namespace ViewComponentSample.ViewComponents
{
    public class PriorityList : ViewComponent
    {
        private readonly ToDoContext db;

        public PriorityList(ToDoContext context)
        {
            db = context;
        }

        public async Task<IViewComponentResult> InvokeAsync(
            int maxPriority, bool isDone)
        {
            var items = await GetItemsAsync(maxPriority, isDone);
            return View(items);
        }

        private Task<List<TodoItem>> GetItemsAsync(int maxPriority, bool isDone)
        {
            return db.ToDo.Where(x => x.IsDone == isDone &&
                x.Priority <= maxPriority).ToListAsync();
        }
    }
}
```

Adicione uma instrução `using` ao arquivo de exibição do Razor e use o operador `nameof`:

```

@using ViewComponentSample.Models
@using ViewComponentSample.ViewComponents
@model IEnumerable<TodoItem>

<h2>ToDo nameof</h2>
<!-- Markup removed for brevity. --&gt;

&lt;div&gt;

/*
Note:
To use the below line, you need to #define no_suffix in ViewComponents/PriorityList.cs or it
won't compile.
By doing so it will cause a problem to index as there will be multiple viewcomponents
with the same name after the compiler removes the suffix "ViewComponent"
*@</pre>


@*await Component.InvokeAsync(nameof(PriorityList), new { maxPriority = 4, isDone = true })*@
</div>


```

## Executar o trabalho síncrono

A estrutura manipulará a invocação de um método `Invoke` síncrono se não for necessário realizar o trabalho assíncrono. O método a seguir cria um componente de exibição `Invoke` síncrono:

```

public class PriorityList : ViewComponent
{
    public IViewComponentResult Invoke(int maxPriority, bool isDone)
    {
        var items = new List<string> { $"maxPriority: {maxPriority}", $"isDone: {isDone}" };
        return View(items);
    }
}
```

O arquivo do Razor do componente de exibição lista as cadeias de caracteres passadas para o método `Invoke` (*Views/Home/Components/PriorityList/Default.cshtml*):

```

@model List<string>

<h3>Priority Items</h3>
<ul>
    @foreach (var item in Model)
    {
        <li>@item</li>
    }
</ul>
```

O componente de exibição é invocado em um arquivo do Razor (por exemplo, *Views/Home/Index.cshtml*) usando uma das seguintes abordagens:

- [IViewComponentHelper](#)
- [Auxiliar de Marca](#)

Para usar a abordagem [IViewComponentHelper](#), chame `Component.InvokeAsync`:

O componente de exibição é invocado em um arquivo do Razor (por exemplo, *Views/Home/Index.cshtml*) com [IViewComponentHelper](#).

Chame `Component.InvokeAsync`:

```
@await Component.InvokeAsync(nameof(PriorityList), new { maxPriority = 4, isDone = true })
```

Para usar o Auxiliar de Marca, registre o assembly que contém o Componente de exibição que usa a diretiva `@addTagHelper` (o componente de exibição está em um assembly chamado `MyWebApp`):

```
@addTagHelper *, MyWebApp
```

Use o componente de exibição Auxiliar de Marca no arquivo de marcação do Razor:

```
<vc:priority-list max-priority="999" is-done="false">
</vc:priority-list>
```

A assinatura do método de `PriorityList.Invoke` é síncrona, mas o Razor localiza e chama o método com `Component.InvokeAsync` no arquivo de marcação.

## Recursos adicionais

- [Injeção de dependência em exibições](#)

# Compilação de arquivo do Razor no ASP.NET Core

30/01/2019 • 6 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

Um arquivo do Razor é compilado em tempo de execução, quando o modo de exibição do MVC associado é chamado. Não há suporte para a publicação de arquivos do Razor de tempo de build. Como opção, os arquivos do Razor podem ser compilados durante a publicação e implantados com o aplicativo — usando a ferramenta de pré-compilação.

Um arquivo do Razor é compilado em tempo de execução, quando o modo de exibição do MVC ou da Página do Razor associada é chamado. Não há suporte para a publicação de arquivos do Razor de tempo de build. Como opção, os arquivos do Razor podem ser compilados durante a publicação e implantados com o aplicativo — usando a ferramenta de pré-compilação.

Um arquivo do Razor é compilado em tempo de execução, quando o modo de exibição do MVC ou da Página do Razor associada é chamado. Os arquivos do Razor são compilados em tempo de build e de publicação usando o [SDK do Razor](#).

## Considerações sobre a pré-compilação

Estes são os efeitos colaterais da pré-compilação de arquivos do Razor:

- Pacote publicado menor
- Tempo de inicialização mais rápido
- Não é possível editar arquivos do Razor—o conteúdo associado está ausente do pacote publicado.

## Implantar arquivos pré-compilados

A compilação em tempo de build e de publicação de arquivos do Razor está habilitada por padrão pelo SDK do Razor. Há suporte para edição de arquivos do Razor depois que eles são atualizados em tempo de build. Por padrão, nenhum arquivo .cshtml é implantado com o aplicativo; somente a *Views.dll* compilada.

### IMPORTANT

A ferramenta de pré-compilação será removida no ASP.NET Core 3.0. É recomendado migrar para o [SDK do Razor](#).

O SDK do Razor é eficaz somente quando não há propriedades específicas de pré-compilação definidas no arquivo de projeto. Por exemplo, definir a propriedade `MvcRazorCompileOnPublish` do arquivo `.csproj` como `true` desabilita o SDK do Razor.

Se o projeto for direcionado ao .NET Framework, instale o pacote NuGet [Microsoft.AspNetCore.Mvc.Razor.ViewCompilation](#):

```
<PackageReference Include="Microsoft.AspNetCore.Mvc.Razor.ViewCompilation"
    Version="2.0.4"
    PrivateAssets="All" />
```

Se o projeto for direcionado ao .NET Core, nenhuma alteração será necessária.

Os modelos de projeto do ASP.NET Core 2.x definem implicitamente a propriedade `MvcRazorCompileOnPublish`

como `true` por padrão. Consequentemente, esse elemento pode ser removido com segurança do arquivo `.csproj`.

## IMPORTANT

A ferramenta de pré-compilação será removida no ASP.NET Core 3.0. É recomendado migrar para o [SDK do Razor](#).

A pré-compilação do arquivo do Razor não está disponível durante a execução de uma [SCD \(implantação autossuficiente\)](#) no ASP.NET Core 2.0.

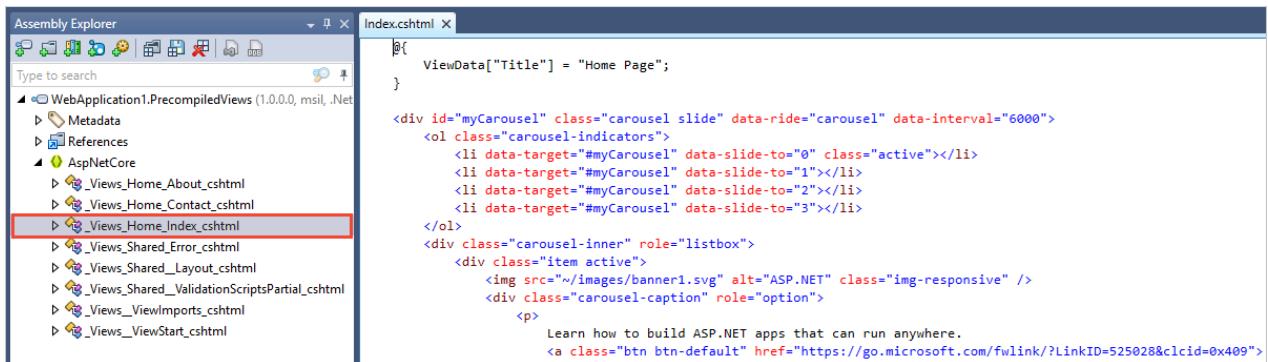
Defina a propriedade `MvcRazorCompileOnPublish` como `true` e instale o pacote NuGet [Microsoft.AspNetCore.Mvc.Razor.ViewCompilation](#). A seguinte amostra `.csproj` realça essas configurações:

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp1.1</TargetFramework>
    <MvcRazorCompileOnPublish>true</MvcRazorCompileOnPublish>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore" Version="1.1.0" />
    <PackageReference Include="Microsoft.AspNetCore.Mvc" Version="1.1.0" />
    <PackageReference Include="Microsoft.AspNetCore.StaticFiles" Version="1.1.0" />
    <PackageReference Include="Microsoft.AspNetCore.Mvc.Razor.ViewCompilation" Version="1.1.0-*" />
  </ItemGroup>
</Project>
```

Prepare o aplicativo para uma [implantação dependente de estrutura](#) com o [comando de publicação da CLI do .NET Core](#). Por exemplo, execute o seguinte comando na raiz do projeto:

```
dotnet publish -c Release
```

Um arquivo `<nome_do_projeto>.PrecompiledViews.dll`, que contém os arquivos do Razor compilados, é produzido quando a pré-compilação é bem-sucedida. Por exemplo, a captura de tela abaixo mostra o conteúdo de `Index.cshtml` dentro de `WebApplication1.PrecompiledViews.dll`:



## Recompilar arquivos do Razor em alteração

O [RazorViewEngineOptions](#) `AllowRecompilingViewsOnChange` obtém ou define um valor que determina se os arquivos do Razor (Exibições Razor e Razor Pages) são recompilados e atualizados, quando alterados em disco.

Quando definido como `true`, [IFileProvider.Watch](#) busca alterações nos arquivos do Razor, em instâncias configuradas de [IFileProvider](#).

O valor padrão é `true` para:

- ASP.NET Core 2.1 ou aplicativos anteriores.
- ASP.NET Core 2.2 ou aplicativos posteriores no ambiente de desenvolvimento.

`AllowRecompilingViewsOnFileChange` está associado a uma opção de compatibilidade e pode fornecer um comportamento diferente, dependendo da versão de compatibilidade configurada do aplicativo. A configuração do aplicativo definindo `AllowRecompilingViewsOnFileChange` tem precedência sobre o valor indicado pela versão de compatibilidade do aplicativo.

Se a versão de compatibilidade do aplicativo for definida como `Version_2_1` ou anterior,

`AllowRecompilingViewsOnFileChange` será definida como `true`, a menos que seja explicitamente configurada.

Se a versão de compatibilidade do aplicativo for definida como `CompatibilityVersion.Version_2_2` ou posterior,

`AllowRecompilingViewsOnFileChange` será definida como `false`, a menos que é o ambiente seja de desenvolvimento ou o valor seja explicitamente configurado.

Para ver exemplos e obter orientação sobre como definir a versão de compatibilidade do aplicativo, confira [Versão de compatibilidade do ASP.NET Core MVC](#).

## Recursos adicionais

- [Exibições no ASP.NET Core MVC](#)
- [Introdução a Páginas do Razor no ASP.NET Core](#)
- [Exibições no ASP.NET Core MVC](#)
- [Introdução a Páginas do Razor no ASP.NET Core](#)
- [Exibições no ASP.NET Core MVC](#)
- [SDK do Razor do ASP.NET Core](#)

# Trabalhar com o modelo de aplicativo no ASP.NET Core

13/11/2018 • 18 minutes to read • [Edit Online](#)

Por [Steve Smith](#)

O ASP.NET Core MVC define um *modelo de aplicativo* que representa os componentes de um aplicativo MVC. Leia e manipule esse modelo para modificar o comportamento de elementos MVC. Por padrão, o MVC segue algumas convenções para determinar quais classes são consideradas controladores, quais métodos nessas classes são ações e qual o comportamento de parâmetros e do roteamento. Personalize esse comportamento de acordo com as necessidades do aplicativo criando suas próprias convenções e aplicando-as globalmente ou como atributos.

## Modelos e provedores

O modelo de aplicativo ASP.NET Core MVC inclui interfaces abstratas e classes de implementação concreta que descrevem um aplicativo MVC. Esse modelo é o resultado da descoberta do MVC de controladores, ações, parâmetros de ação, rotas e filtros do aplicativo de acordo com as convenções padrão. Trabalhando com o modelo de aplicativo, você pode modificar o aplicativo para seguir convenções diferentes do comportamento padrão do MVC. Os parâmetros, os nomes, as rotas e os filtros são todos usados como dados de configuração para ações e controladores.

O Modelo de Aplicativo ASP.NET Core MVC tem a seguinte estrutura:

- ApplicationModel
  - Controladores (ControllerModel)
  - Ações (ActionModel)
  - Parâmetros (ParameterModel)

Cada nível do modelo tem acesso a uma coleção `Properties` comum e níveis inferiores podem acessar e substituir valores de propriedade definidos por níveis mais altos na hierarquia. As propriedades são persistidas nas `ActionDescriptor.Properties` quando as ações são criadas. Em seguida, quando uma solicitação está sendo manipulada, as propriedades que uma convenção adicionou ou modificou podem ser acessadas por meio de `ActionContext.ActionDescriptor.Properties`. O uso de propriedades é uma ótima maneira de configurar filtros, associadores de modelos, etc., por ação.

### NOTE

A coleção `ActionDescriptor.Properties` não é thread-safe (para gravações) após a conclusão da inicialização do aplicativo. Convenções são a melhor maneira de adicionar dados com segurança a essa coleção.

## IApplicationModelProvider

O ASP.NET Core MVC carrega o modelo de aplicativo usando um padrão de provedor, definido pela interface [IApplicationModelProvider](#). Esta seção aborda alguns dos detalhes de implementação interna de como funciona esse provedor. Este é um tópico avançado – a maioria dos aplicativos que utiliza o modelo de aplicativo deve fazer isso trabalhando com convenções.

Implementações da interface `IApplicationModelProvider` "encapsulam" umas às outras, com cada implementação chamando `OnProvidersExecuting` em ordem crescente com base em sua propriedade `order`. O método

`OnProvidersExecuted` é então chamado em ordem inversa. A estrutura define vários provedores:

Primeiro (`Order=-1000`):

- `DefaultApplicationModelProvider`

Em seguida (`Order=-990`):

- `AuthorizationApplicationModelProvider`
- `CorsApplicationModelProvider`

#### NOTE

A ordem na qual dois provedores com o mesmo valor para `Order` são chamados não é definida e, portanto, não se deve depender dela.

#### NOTE

`IApplicationModelProvider` é um conceito avançado a ser estendido pelos autores da estrutura. Em geral, aplicativos devem usar convenções e estruturas devem usar provedores. A principal diferença é que os provedores sempre são executados antes das convenções.

O `DefaultApplicationModelProvider` estabelece muitos dos comportamentos padrão usados pelo ASP.NET Core MVC. Suas responsabilidades incluem:

- Adicionar filtros globais ao contexto
- Adicionar controladores ao contexto
- Adicionar métodos de controlador públicos como ações
- Adicionar parâmetros de método de ação ao contexto
- Aplicar a rota e outros atributos

Alguns comportamentos internos são implementados pelo `DefaultApplicationModelProvider`. Esse provedor é responsável pela construção do `ControllerModel`, que, por sua vez, referencia instâncias `ActionModel`, `PropertyModel` e `ParameterModel`. A classe `DefaultApplicationModelProvider` é um detalhe de implementação de estrutura interna que pode e será alterado no futuro.

O `AuthorizationApplicationModelProvider` é responsável por aplicar o comportamento associado aos atributos `AuthorizeFilter` e `AllowAnonymousFilter`. [Saiba mais sobre esses atributos.](#)

O `CorsApplicationModelProvider` implementa o comportamento associado ao `IEnableCorsAttribute`, ao `IDisableCorsAttribute` e ao `DisableCorsAuthorizationFilter`. [Saiba mais sobre o CORS.](#)

## Convenções

O modelo de aplicativo define abstrações de convenção que fornecem uma maneira simples de personalizar o comportamento dos modelos em vez de substituir o modelo ou o provedor inteiro. Essas abstrações são a maneira recomendada para modificar o comportamento do aplicativo. As convenções fornecem uma maneira de escrever código que aplicará personalizações de forma dinâmica. Enquanto os `filtros` fornecem um meio de modificar o comportamento da estrutura, as personalizações permitem que você controle como todo o aplicativo está conectado.

As seguintes convenções estão disponíveis:

- `IApplicationModelConvention`

- [IControllerModelConvention](#)
- [IActionModelConvention](#)
- [IParameterModelConvention](#)

As convenções são aplicadas por sua adição às opções do MVC ou pela implementação de `Attributes` e sua aplicação a controladores, ações ou parâmetros de ação (semelhante a [Filters](#)). Ao contrário dos filtros, as convenções são executadas apenas quando o aplicativo é iniciado, não como parte de cada solicitação.

### **Amostra: modificando o ApplicationModel**

A convenção a seguir é usada para adicionar uma propriedade ao modelo de aplicativo.

```
using Microsoft.AspNetCore.Mvc.ApplicationModels;

namespace AppModelSample.Conventions
{
    public class ApplicationDescription : IApplicationModelConvention
    {
        private readonly string _description;

        public ApplicationDescription(string description)
        {
            _description = description;
        }

        public void Apply(ApplicationModel application)
        {
            application.Properties["description"] = _description;
        }
    }
}
```

As convenções de modelo de aplicativo são aplicadas como opções quando o MVC é adicionado em `ConfigureServices` em `Startup`.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(options =>
    {
        options.Conventions.Add(new ApplicationDescription("My Application Description"));
        options.Conventions.Add(new NamespaceRoutingConvention());
        //options.Conventions.Add(new IdsMustBeInRouteParameterModelConvention());
    });
}
```

As propriedades são acessíveis na coleção de propriedades `ActionDescriptor` nas ações do controlador:

```
public class AppModelController : Controller
{
    public string Description()
    {
        return "Description: " + ControllerContext.ActionDescriptor.Properties["description"];
    }
}
```

### **Amostra: modificando a descrição de ControllerModel**

Como no exemplo anterior, o modelo de controlador também pode ser modificado para incluir propriedades personalizadas. Elas substituirão as propriedades existentes com o mesmo nome especificado no modelo de aplicativo. O seguinte atributo de convenção adiciona uma descrição no nível do controlador:

```

using System;
using Microsoft.AspNetCore.Mvc.ApplicationModels;

namespace AppModelSample.Conventions
{
    public class ControllerDescriptionAttribute : Attribute, IControllerModelConvention
    {
        private readonly string _description;

        public ControllerDescriptionAttribute(string description)
        {
            _description = description;
        }

        public void Apply(ControllerModel controllerModel)
        {
            controllerModel.Properties["description"] = _description;
        }
    }
}

```

Essa convenção é aplicada como um atributo em um controlador.

```

[ControllerDescription("Controller Description")]
public class DescriptionAttributesController : Controller
{
    public string Index()
    {
        return "Description: " + ControllerContext.ActionDescriptor.Properties["description"];
    }
}

```

A propriedade "description" é acessada da mesma maneira como nos exemplos anteriores.

### **Amostra: modificando a descrição de ActionModel**

Uma convenção de atributo separada pode ser aplicada a ações individuais, substituindo o comportamento já aplicado no nível do aplicativo ou do controlador.

```

using System;
using Microsoft.AspNetCore.Mvc.ApplicationModels;

namespace AppModelSample.Conventions
{
    public class ActionDescriptionAttribute : Attribute, IActionModelConvention
    {
        private readonly string _description;

        public ActionDescriptionAttribute(string description)
        {
            _description = description;
        }

        public void Apply(ActionModel actionModel)
        {
            actionModel.Properties["description"] = _description;
        }
    }
}

```

A aplicação disso a uma ação no controlador do exemplo anterior demonstra como ela substitui a convenção no nível do controlador:

```
[ControllerDescription("Controller Description")]
public class DescriptionAttributesController : Controller
{
    public string Index()
    {
        return "Description: " + ControllerContext.ActionDescriptor.Properties["description"];
    }

    [ActionDescription("Action Description")]
    public string UseActionDescriptionAttribute()
    {
        return "Description: " + ControllerContext.ActionDescriptor.Properties["description"];
    }
}
```

### Amostra: modificando o ParameterModel

A convenção a seguir pode ser aplicada a parâmetros de ação para modificar seu `BindingInfo`. A convenção a seguir exige que o parâmetro seja um parâmetro de rota; outras possíveis origens da associação (como valores de cadeia de caracteres de consulta) são ignoradas.

```
using System;
using Microsoft.AspNetCore.Mvc.ApplicationModels;
using Microsoft.AspNetCore.Mvc.ModelBinding;

namespace AppModelSample.Conventions
{
    public class MustBeInRouteParameterModelConvention : Attribute, IParameterModelConvention
    {
        public void Apply(ParameterModel model)
        {
            if (model.BindingInfo == null)
            {
                model.BindingInfo = new BindingInfo();
            }
            model.BindingInfo.BindingSource = BindingSource.Path;
        }
    }
}
```

O atributo pode ser aplicado a qualquer parâmetro de ação:

```
public class ParameterModelController : Controller
{
    // Will bind: /ParameterModel/.GetById/123
    // WON'T bind: /ParameterModel/.GetById?id=123
    public string GetById([MustBeInRouteParameterModelConvention]int id)
    {
        return $"Bound to id: {id}";
    }
}
```

### Amostra: modificando o nome de ActionModel

A convenção a seguir modifica o `ActionModel` para atualizar o *nome* da ação ao qual ele é aplicado. O novo nome é fornecido como um parâmetro para o atributo. Esse novo nome é usado pelo roteamento e, portanto, isso afetará a rota usada para acessar esse método de ação.

```

using System;
using Microsoft.AspNetCore.Mvc.ApplicationModels;

namespace AppModelSample.Conventions
{
    public class CustomActionNameAttribute : Attribute, IActionModelConvention
    {
        private readonly string _actionName;

        public CustomActionNameAttribute(string actionPerformed)
        {
            _actionName = actionPerformed;
        }

        public void Apply(ActionModel actionModel)
        {
            // this name will be used by routing
            actionModel.ActionName = _actionName;
        }
    }
}

```

Esse atributo é aplicado a um método de ação no `HomeController`:

```

// Route: /Home/MyCoolAction
[CustomActionName("MyCoolAction")]
public string SomeName()
{
    return ControllerContext.ActionDescriptor.ActionName;
}

```

Mesmo que o nome do método seja `SomeName`, o atributo substitui a convenção do MVC de uso do nome do método e substitui o nome da ação por `MyCoolAction`. Portanto, a rota usada para acessar essa ação é `/Home/MyCoolAction`.

#### NOTE

Esse exemplo é basicamente o mesmo que usar o atributo `ActionName` interno.

#### **Amostra: convenção de roteamento personalizada**

Use uma `IApplicationModelConvention` para personalizar como funciona o roteamento. Por exemplo, a seguinte convenção incorporará namespaces dos Controladores em suas rotas, substituindo `.` no namespace por `/` na rota:

```

using Microsoft.AspNetCore.Mvc.ApplicationModels;
using System.Linq;

namespace AppModelSample.Conventions
{
    public class NamespaceRoutingConvention : IApplicationModelConvention
    {
        public void Apply(ApplicationModel application)
        {
            foreach (var controller in application.Controllers)
            {
                var hasAttributeRouteModels = controller.Selectors
                    .Any(selector => selector.AttributeRouteModel != null);

                if (!hasAttributeRouteModels
                    && controller.ControllerName.Contains("Namespace")) // affect one controller in this sample
                {
                    // Replace the . in the namespace with a / to create the attribute route
                    // Ex: MySite.Admin namespace will correspond to MySite/Admin attribute route
                    // Then attach [controller], [action] and optional {id?} token.
                    // [Controller] and [action] is replaced with the controller and action
                    // name to generate the final template
                    controller.Selectors[0].AttributeRouteModel = new AttributeRouteModel()
                    {
                        Template = controller.ControllerType.Namespace.Replace('.', '/') +
                        "/{controller}/{action}/{id?}"
                    };
                }
            }

            // You can continue to put attribute route templates for the controller actions depending on the
            // way you want them to behave
        }
    }
}

```

A convenção é adicionada como uma opção na classe Startup.

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(options =>
    {
        options.Conventions.Add(new ApplicationDescription("My Application Description"));
        options.Conventions.Add(new NamespaceRoutingConvention());
        //options.Conventions.Add(new IdsMustBeInRouteParameterModelConvention());
    });
}

```

#### TIP

Adicione convenções ao middleware acessando `MvcOptions` com

```
services.Configure<MvcOptions>(c => c.Conventions.Add(YOURCONVENTION));
```

Esta amostra aplica essa convenção às rotas que não estão usando o roteamento de atributo, nas quais o controlador tem "Namespace" em seu nome. O seguinte controlador demonstra essa convenção:

```
using Microsoft.AspNetCore.Mvc;

namespace AppModelSample.Controllers
{
    public class NamespaceRoutingController : Controller
    {
        // using NamespaceRoutingConvention
        // route: /AppModelSample/Controllers/NamespaceRouting/Index
        public string Index()
        {
            return "This demonstrates namespace routing.";
        }
    }
}
```

## Uso do modelo de aplicativo em WebApiCompatShim

O ASP.NET Core MVC usa um conjunto diferente de convenções do ASP.NET Web API 2. Usando convenções personalizadas, você pode modificar o comportamento de um aplicativo ASP.NET Core MVC para que ele seja consistente com o comportamento de um aplicativo de API Web. A Microsoft fornece o [WebApiCompatShim](#) especificamente para essa finalidade.

### NOTE

Saiba mais sobre [migração do ASP.NET Web API](#).

Para usar o Shim de Compatibilidade de API Web, você precisa adicionar o pacote ao projeto e, em seguida, adicionar as convenções ao MVC chamando `AddWebApiConventions` em `Startup`:

```
services.AddMvc().AddWebApiConventions();
```

As convenções fornecidas pelo shim são aplicadas apenas às partes do aplicativo que tiveram determinados atributos aplicados a elas. Os seguintes quatro atributos são usados para controlar quais controladores devem ter suas convenções modificadas pelas convenções do shim:

- [UseWebApiActionConventionsAttribute](#)
- [UseWebApiOverloadingAttribute](#)
- [UseWebApiParameterConventionsAttribute](#)
- [UseWebApiRoutesAttribute](#)

### Convenções de ação

O `UseWebApiActionConventionsAttribute` é usado para mapear o método HTTP para ações com base em seu nome (por exemplo, `Get` será mapeado para `HttpGet`). Ele se aplica somente a ações que não usam o roteamento de atributo.

### Sobrecarga

O `UseWebApiOverloadingAttribute` é usado para aplicar a convenção `WebApiOverloadingApplicationModelConvention`. Essa convenção adiciona uma `OverloadActionConstraint` ao processo de seleção de ação, o que limita as ações de candidato àquelas para as quais a solicitação atende a todos os parâmetros não opcionais.

### Convenções de parâmetro

O `UseWebApiParameterConventionsAttribute` é usado para aplicar a convenção de ação `WebApiParameterConventionsApplicationModelConvention`. Essa convenção especifica que tipos simples usados como parâmetros de ação são associados por meio do URI por padrão, enquanto tipos complexos são associados por

meio do corpo da solicitação.

## Rotas

O `UseWebApiRoutesAttribute` controla se a convenção de controlador `WebApiApplicationModelConvention` é aplicada. Quando habilitada, essa convenção é usada para adicionar suporte de [áreas](#) à rota.

Além de um conjunto de convenções, o pacote de compatibilidade inclui uma classe base `System.Web.Http.ApiController` que substitui aquela fornecida pela API Web. Isso permite que os controladores escritos para a API Web e que herdam de seu `ApiController` funcionem como foram criados, enquanto são executados no ASP.NET Core MVC. Essa classe base de controlador é decorada com todos os atributos `UseWebApi*` listados acima. O `ApiController` expõe propriedades, métodos e tipos de resultado compatíveis com aqueles encontrados na API Web.

## Usando o ApiExplorer para documentar o aplicativo

O modelo de aplicativo expõe uma propriedade `ApiExplorer` em cada nível, que pode ser usada para percorrer a estrutura do aplicativo. Isso pode ser usado para [gerar páginas da Ajuda para as APIs Web usando ferramentas como o Swagger](#). A propriedade `ApiExplorer` expõe uma propriedade `IsVisible` que pode ser definida para especificar quais partes do modelo do aplicativo devem ser expostas. Defina essa configuração usando uma convenção:

```
using Microsoft.AspNetCore.Mvc.ApplicationModels;

namespace AppModelSample.Conventions
{
    public class EnableApiExplorerApplicationConvention : IApplicationModelConvention
    {
        public void Apply(ApplicationModel application)
        {
            application.ApiExplorer.IsVisible = true;
        }
    }
}
```

Usando essa abordagem (e convenções adicionais, se necessário), você pode habilitar ou desabilitar a visibilidade da API em qualquer nível no aplicativo.

# Filtros no ASP.NET Core

30/01/2019 • 38 minutes to read • [Edit Online](#)

Por [Rick Anderson](#), [Tom Dykstra](#) e [Steve Smith](#)

Os *Filtros* no ASP.NET Core MVC permitem executar código antes ou depois de determinados estágios do pipeline de processamento de solicitações.

O filtros internos lidam com tarefas como:

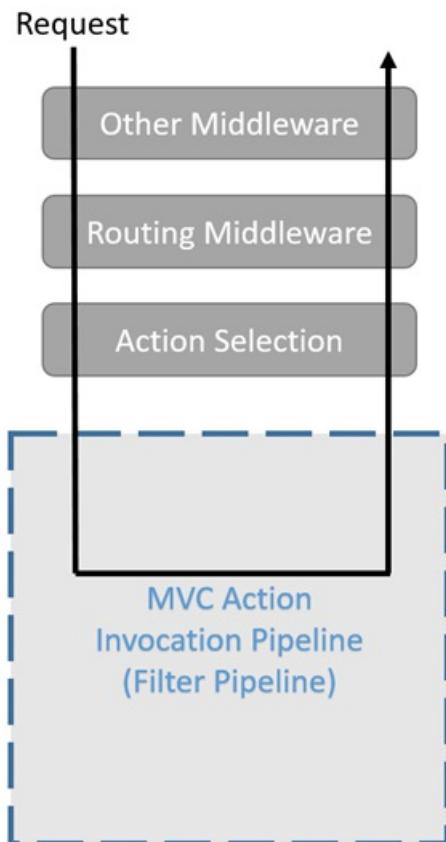
- Autorização (impedir o acesso a recursos aos quais o usuário não está autorizado).
- Garantir que todas as solicitações usem HTTPS.
- Cache de resposta (causar um curto-circuito do pipeline de solicitação para retornar uma resposta armazenada em cache).

É possível criar filtros personalizados para lidar com interesses paralelos. Os filtros podem evitar a duplicação de código entre as ações. Por exemplo, um filtro de exceção de tratamento de erro poderia consolidar o tratamento de erro.

[Exibir ou baixar amostra do GitHub](#).

## Como os filtros funcionam

Os filtros são executados dentro do *pipeline de invocação de ações do MVC*, às vezes chamado de *pipeline de filtros*. O pipeline de filtros é executado após o MVC selecionar a ação a ser executada.

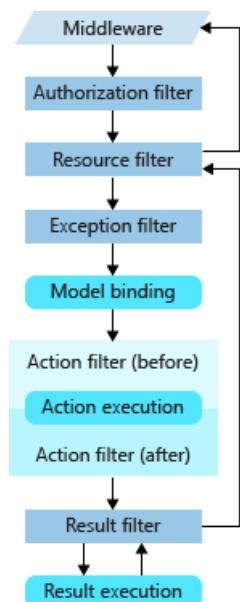


## Tipos de filtro

Cada tipo de filtro é executado em um estágio diferente no pipeline de filtros.

- **Filtros de autorização** são executados primeiro e são usados para determinar se o usuário atual tem autorização para a solicitação atual. Eles podem fazer um curto-circuito do pipeline quando uma solicitação não é autorizada.
- **Filtros de recurso** são os primeiros a lidar com uma solicitação após a autorização. Eles podem executar código antes do restante do pipeline de filtros e após o restante do pipeline ser concluído. Esses filtros são úteis para implementar o cache ou para, de alguma forma, fazer o curto-circuito do pipeline de filtros por motivos de desempenho. Eles são executados antes do model binding, portanto, podem influenciar o model binding.
- **Filtros de ação** podem executar código imediatamente antes e depois de um método de ação individual ser chamado. Eles podem ser usados para manipular os argumentos passados para uma ação, bem como o resultado da ação. Os filtros de ação não são compatíveis com o Razor Pages.
- **Filtros de exceção** são usados para aplicar políticas globais para exceções sem tratamento que ocorrem antes que qualquer coisa tenha sido gravada no corpo da resposta.
- **Filtros de resposta** podem executar código imediatamente antes e depois da execução de resultados de ações individuais. Eles são executados somente quando o método de ação é executado com êxito. Eles são úteis para a lógica que precisa envolver a execução da exibição ou do formatador.

O diagrama a seguir mostra como esses tipos de filtro interagem no pipeline de filtros.



## Implementação

Os filtros dão suporte a implementações síncronas e assíncronas por meio de diferentes definições de interface.

Filtros síncronos que podem executar código antes e depois do estágio do pipeline definem os métodos `OnStageExecuting` e `OnStageExecuted`. Por exemplo, `OnActionExecuting` é chamado antes que o método de ação seja chamado e `OnActionExecuted` é chamado após o método de ação retornar.

```

using FiltersSample.Helper;
using Microsoft.AspNetCore.Mvc.Filters;

namespace FiltersSample.Filters
{
    public class SampleActionFilter : IActionFilter
    {
        public void OnActionExecuting(ActionExecutingContext context)
        {
            // do something before the action executes
        }

        public void OnActionExecuted(ActionExecutedContext context)
        {
            // do something after the action executes
        }
    }
}

```

Filtros assíncronos definem um único método `OnStageExecutionAsync`. Esse método usa um delegado `FilterTypeExecutionDelegate`, que executa o estágio de pipeline do filtro. Por exemplo, `ActionExecutionDelegate` chama o método de ação ou o próximo filtro de ação, e você pode executar código antes e depois de chamá-lo.

```

using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc.Filters;

namespace FiltersSample.Filters
{
    public class SampleAsyncActionFilter : IAsyncActionFilter
    {
        public async Task OnActionExecutionAsync(
            ActionExecutingContext context,
            ActionExecutionDelegate next)
        {
            // do something before the action executes
            var resultContext = await next();
            // do something after the action executes; resultContext.Result will be set
        }
    }
}

```

É possível implementar interfaces para vários estágios do filtro em uma única classe. Por exemplo, a classe `ActionFilterAttribute` implementa `IActionFilter`, `IResultFilter` e os respectivos equivalentes assíncronos.

#### NOTE

Implemente **ou** a versão assíncrona ou a versão síncrona de uma interface de filtro, não ambas. Primeiro, a estrutura verifica se o filtro implementa a interface assíncrona e, se for esse o caso, a chama. Caso contrário, ela chama os métodos da interface síncrona. Se você implementasse as duas interfaces em uma classe, somente o método assíncrono seria chamado. Ao usar classes abstratas como `ActionFilterAttribute`, você substituiria apenas os métodos síncronos ou o método assíncrono para cada tipo de filtro.

## IFilterFactory

`IFilterFactory` implementa `IFilterMetadata`. Portanto, uma instância `IFilterFactory` pode ser usada como uma instância `IFilterMetadata` em qualquer parte do pipeline de filtro. Quando se prepara para invocar o filtro, a estrutura tenta convertê-lo em um `IFilterFactory`. Se essa conversão for bem-sucedida, o método `CreateInstance` será chamado para criar a instância `IFilterMetadata` que será invocada. Isso fornece um

design flexível, porque o pipeline de filtro preciso não precisa ser definido explicitamente quando o aplicativo é iniciado.

Você pode implementar `IFilterFactory` em suas próprias implementações de atributo como outra abordagem à criação de filtros:

```
public class AddHeaderWithFactoryAttribute : Attribute, IFilterFactory
{
    // Implement IFilterFactory
    public IFilterMetadata CreateInstance(IServiceProvider serviceProvider)
    {
        return new InternalAddHeaderFilter();
    }

    private class InternalAddHeaderFilter : IResultFilter
    {
        public void OnResultExecuting(ResultExecutingContext context)
        {
            context.HttpContext.Response.Headers.Add(
                "Internal", new string[] { "Header Added" });
        }

        public void OnResultExecuted(ResultExecutedContext context)
        {
        }
    }

    public bool IsReusable
    {
        get
        {
            return false;
        }
    }
}
```

## Atributos de filtro internos

A estrutura inclui filtros internos baseados em atributos que você pode organizar em subclasses e personalizar. Por exemplo, o seguinte Filtro de resultado adiciona um cabeçalho à resposta.

```
using Microsoft.AspNetCore.Mvc.Filters;

namespace FiltersSample.Filters
{
    public class AddHeaderAttribute : ResultFilterAttribute
    {
        private readonly string _name;
        private readonly string _value;

        public AddHeaderAttribute(string name, string value)
        {
            _name = name;
            _value = value;
        }

        public override void OnResultExecuting(ResultExecutingContext context)
        {
            context.HttpContext.Response.Headers.Add(
                _name, new string[] { _value });
            base.OnResultExecuting(context);
        }
    }
}
```

Os atributos permitem que os filtros aceitem argumentos, conforme mostrado no exemplo acima. Você adicionaria esse atributo a um método de ação ou controlador e especificaria o nome e o valor do cabeçalho HTTP:

```
[AddHeader("Author", "Steve Smith @ardalis")]
public class SampleController : Controller
{
    public IActionResult Index()
    {
        return Content("Examine the headers using developer tools.");
    }

    [ShortCircuitingResourceFilter]
    public IActionResult SomeResource()
    {
        return Content("Successful access to resource - header should be set.");
    }
}
```

O resultado da ação `Index` é mostrado abaixo – os cabeçalhos de resposta são exibidos na parte inferior direita.

The screenshot shows a browser window with the URL `localhost:5000/sample`. The page content is "Examine the headers using developer tools.". Below the page, the browser's developer tools Network tab is open, showing a single request for the URL. In the Headers section of the Network tab, there are two entries: "Author: Steve Smith @ardalis" and "Content-Type: text/plain; charset=utf-8".

Várias interfaces de filtro têm atributos correspondentes que podem ser usados como classes base para implementações personalizadas.

Atributos de filtro:

- `ActionFilterAttribute`
- `ExceptionFilterAttribute`
- `ResultFilterAttribute`
- `FormatFilterAttribute`
- `ServiceFilterAttribute`
- `TypeFilterAttribute`

`TypeFilterAttribute` e `ServiceFilterAttribute` são explicados [posteriormente neste artigo](#).

## Escopos e ordem de execução dos filtros

Um filtro pode ser adicionado ao pipeline com um de três escopos. É possível adicionar um filtro a um

método de ação específico ou a uma classe de controlador usando um atributo. Ou você pode registrar um filtro globalmente para todos os controladores e ações. Os filtros são adicionados globalmente quando são adicionados à coleção `MvcOptions.Filters` em `ConfigureServices`:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(options =>
    {
        options.Filters.Add(new AddHeaderAttribute("GlobalAddHeader",
            "Result filter added to MvcOptions.Filters")); // an instance
        options.Filters.Add(typeof(SampleActionFilter)); // by type
        options.Filters.Add(new SampleGlobalActionFilter()); // an instance
    });

    services.AddScoped<AddHeaderFilterWithDi>();
}
```

## Ordem padrão de execução

Quando há vários filtros para um determinado estágio do pipeline, o escopo determina a ordem padrão de execução dos filtros. Filtros globais circundam filtros de classe, que, por sua vez, circundam filtros de método. Isso é, às vezes, chamado de aninhamento de "bonecas russas", pois cada aumento de escopo é encapsulado em torno do escopo anterior, como uma [matriosca](#). Normalmente, você obtém o comportamento de substituição desejado sem precisar determinar uma ordem explicitamente.

Como resultado desse aninhamento, o código *posterior* dos filtros é executado na ordem inversa do código *anterior*. A sequência é semelhante a esta:

- O código *anterior* de filtros aplicados globalmente
  - O código *anterior* de filtros aplicados a controladores
    - O código *anterior* de filtros aplicados a métodos de ação
    - O código *posterior* de filtros aplicados a métodos de ação
    - O código *posterior* de filtros aplicados a controladores
  - O código *posterior* de filtros aplicados globalmente

Este é um exemplo que ilustra a ordem na qual métodos de filtro são chamados para filtros de ação síncronos.

SEQUÊNCIA	ESCOPO DO FILTRO	MÉTODO DO FILTRO
1	Global	OnActionExecuting
2	Controlador	OnActionExecuting
3	Método	OnActionExecuting
4	Método	OnActionExecuted
5	Controlador	OnActionExecuted
6	Global	OnActionExecuted

Esta sequência mostra:

- O filtro de método está aninhado no filtro de controlador.
- O filtro de controlador está aninhado no filtro global.

Para colocar de outra forma, se você estiver dentro do método `OnStageExecutionAsync` de um filtro assíncrono, todos os filtros com escopo mais estreito serão executados enquanto seu código estiver na pilha.

#### NOTE

Cada controlador que herda da classe base `Controller` inclui os métodos `onActionExecuting` e `OnActionExecuted`. Esses métodos encapsulam os filtros que são executados para uma determinada ação: `OnActionExecuting` é chamado antes de qualquer um dos filtros e `OnActionExecuted` é chamado após todos os filtros.

### Substituindo a ordem padrão

É possível substituir a sequência padrão de execução implementando `IOrderedFilter`. Essa interface expõe uma propriedade `Order` que tem precedência sobre o escopo para determinar a ordem de execução. Um filtro com um valor mais baixo de `Order` terá seu código *anterior* executado antes que um filtro com um valor mais alto de `Order`. Um filtro com um valor mais baixo de `Order` terá seu código *posterior* executado depois que um filtro com um valor mais alto de `Order`. Você pode definir a propriedade `Order` usando um parâmetro de construtor:

```
[MyFilter(Name = "Controller Level Attribute", Order=1)]
```

Se você tiver os mesmos três filtros de ação mostrados no exemplo anterior, mas definir a propriedade `Order` dos filtros de controlador e global como 1 e 2 respectivamente, a ordem de execução será invertida.

SEQUÊNCIA	ESCOPO DO FILTRO	PROPRIEDADE ORDER	MÉTODO DO FILTRO
1	Método	0	<code>OnActionExecuting</code>
2	Controlador	1	<code>OnActionExecuting</code>
3	Global	2	<code>OnActionExecuting</code>
4	Global	2	<code>OnActionExecuted</code>
5	Controlador	1	<code>OnActionExecuted</code>
6	Método	0	<code>OnActionExecuted</code>

A propriedade `Order` tem precedência sobre o escopo ao determinar a ordem na qual os filtros serão executados. Os filtros são classificados primeiro pela ordem e o escopo é usado para desempatar. Todos os filtros internos implementam `IOrderedFilter` e definem o valor de `Order` padrão como 0. Para os filtros internos, o escopo determina a ordem, a menos que você defina `Order` com um valor diferente de zero.

### Cancelamento e curto-circuito

Você pode fazer um curto-circuito no pipeline de filtros a qualquer momento, definindo a propriedade `Result` no parâmetro `context` fornecido ao método do filtro. Por exemplo, o filtro de recurso a seguir impede que o resto do pipeline seja executado.

```

using System;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;

namespace FiltersSample.Filters
{
    public class ShortCircuitingResourceFilterAttribute : Attribute,
        IResourceFilter
    {
        public void OnResourceExecuting(ResourceExecutingContext context)
        {
            context.Result = new ContentResult()
            {
                Content = "Resource unavailable - header should not be set"
            };
        }

        public void OnResourceExecuted(ResourceExecutedContext context)
        {
        }
    }
}

```

No código a seguir, os filtros `ShortCircuitingResourceFilter` e `AddHeader` têm como destino o método de ação `SomeResource`. O `ShortCircuitingResourceFilter`:

- É executado primeiro, porque ele é um filtro de recurso e `AddHeader` é um filtro de ação.
- Causa um curto-círcuito no restante do pipeline.

Portanto, o filtro `AddHeader` nunca é executado para a ação `SomeResource`. Esse comportamento seria o mesmo se os dois filtros fossem aplicados no nível do método de ação, desde que `ShortCircuitingResourceFilter` fosse executado primeiro. O `ShortCircuitingResourceFilter` é executado primeiro, devido ao seu tipo de filtro ou pelo uso explícito da propriedade `Order`.

```

[AddHeader("Author", "Steve Smith @ardalis")]
public class SampleController : Controller
{
    public IActionResult Index()
    {
        return Content("Examine the headers using developer tools.");
    }

    [ShortCircuitingResourceFilter]
    public IActionResult SomeResource()
    {
        return Content("Successful access to resource - header should be set.");
    }
}

```

## Injeção de dependência

Filtros podem ser adicionados por tipo ou por instância. Se você adicionar uma instância, ela será usada para cada solicitação. Se você adicionar um tipo, ele será ativado pelo tipo, o que significa que uma instância será criada para cada solicitação e as dependências de construtor serão populadas pela DI ([injeção de dependência](#)). Adicionar um filtro por tipo é equivalente a

```
filters.Add(new TypeFilterAttribute(typeof(MyFilter))).
```

Filtros que são implementados como atributos e adicionados diretamente a classes de controlador ou métodos de ação não podem ter dependências de construtor fornecidas pela DI ([injeção de dependência](#)). Isso ocorre porque os atributos precisam ter os parâmetros de construtor fornecidos quando eles são

aplicados. Essa é uma limitação do funcionamento dos atributos.

Se seus filtros tiverem dependências que você precisa acessar da DI, há várias abordagens com suporte. É possível aplicar o filtro a um método de ação ou classe usando uma das opções a seguir:

- `ServiceFilterAttribute`
- `TypeFilterAttribute`
- `IFilterFactory` implementado em seu atributo

#### NOTE

Uma dependência que talvez você queira obter da DI é um agente. No entanto, evite criar e usar filtros apenas para fins de registro em log, pois é possível que os [recursos de registro em log da estrutura interna](#) já forneçam o que você precisa. Se você for adicionar o registro em log a seus filtros, ele deve se concentrar em questões referentes ao domínio de negócios ou ao comportamento específico de seu filtro, em vez de ações do MVC ou outros eventos da estrutura.

### ServiceFilterAttribute

Tipos de implementação do filtro de serviço são registrados em DI. Um `ServiceFilterAttribute` recupera uma instância do filtro da DI. Adicione o `ServiceFilterAttribute` ao contêiner em `Startup.ConfigureServices` e faça uma referência a ele em um atributo `[ServiceFilter]`:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(options =>
    {
        options.Filters.Add(new AddHeaderAttribute("GlobalAddHeader",
            "Result filter added to MvcOptions.Filters")); // an instance
        options.Filters.Add(typeof(SampleActionFilter)); // by type
        options.Filters.Add(new SampleGlobalActionFilter()); // an instance
    });

    services.AddScoped<AddHeaderFilterWithDi>();
}
```

```
[ServiceFilter(typeof(AddHeaderFilterWithDi))]
public IActionResult Index()
{
    return View();
}
```

Ao usar `ServiceFilterAttribute`, configurar `IsReusable` é uma dica de que a instância de filtro *pode* ser reutilizada fora do escopo de solicitação em que ele foi criada. A estrutura não fornece garantias de que uma única instância do filtro vá ser criada ou que o filtro não será solicitado novamente do contêiner de DI em algum momento posterior. Evite usar `IsReusable` ao usar um filtro que dependa dos serviços com um tempo de vida diferente de singleton.

Usar `ServiceFilterAttribute` sem registrar o tipo de filtro gera uma exceção:

```
System.InvalidOperationException: No service for type
'FiltersSample.Filters.AddHeaderFilterWithDI' has been registered.
```

`ServiceFilterAttribute` implementa `IFilterFactory`. `IFilterFactory` expõe o método `CreateInstance` para criar uma instância de `IFilterMetadata`. O método `CreateInstance` carrega o tipo especificado do contêiner de serviços (DI).

## TypeFilterAttribute

O `TypeFilterAttribute` é semelhante ao `ServiceFilterAttribute`, mas seu tipo não é resolvido diretamente por meio do contêiner de DI. Ele cria uma instância do tipo usando `Microsoft.Extensions.DependencyInjection.ObjectFactory`.

Por causa dessa diferença:

- Os tipos que são referenciados usando o `TypeFilterAttribute` não precisam ser registrados no contêiner primeiro. As dependências deles são atendidas pelo contêiner.
- Opcionalmente, o `TypeFilterAttribute` pode aceitar argumentos de construtor para o tipo.

Ao usar `TypeFilterAttribute`, configurar `IsReusable` é uma dica de que a instância de filtro *pode* ser reutilizada fora do escopo de solicitação em que ele foi criada. A estrutura não fornece nenhuma garantia de que uma única instância do filtro será criada. Evite usar `IsReusable` ao usar um filtro que dependa dos serviços com um tempo de vida diferente de singleton.

O exemplo a seguir demonstra como passar argumentos para um tipo usando `TypeFilterAttribute`:

```
[TypeFilter(typeof(LogConstantFilter),
    Arguments = new object[] { "Method 'Hi' called" })]
public IActionResult Hi(string name)
{
    return Content($"Hi {name}");
}
```

```
public class LogConstantFilter : IActionFilter
{
    private readonly string _value;
    private readonly ILogger<LogConstantFilter> _logger;

    public LogConstantFilter(string value, ILogger<LogConstantFilter> logger)
    {
        _logger = logger;
        _value = value;
    }

    public void OnActionExecuting(ActionExecutingContext context)
    {
        _logger.LogInformation(_value);
    }

    public void OnActionExecuted(ActionExecutedContext context)
    { }
}
```

## IFilterFactory implementado em seu atributo

Se você tiver um filtro que:

- Não exija nenhum argumento.
- Tenha dependências de construtor que precisem ser atendidas pela DI.

Você pode usar seu próprio atributo nomeado em classes e métodos em vez de `[TypeFilter(typeof(FilterType))]`. O filtro a seguir mostra como isso pode ser implementado:

```

public class SampleActionFilterAttribute : TypeFilterAttribute
{
    public SampleActionFilterAttribute():base(typeof(SampleActionFilterImpl))
    {
    }

    private class SampleActionFilterImpl : IActionFilter
    {
        private readonly ILogger _logger;
        public SampleActionFilterImpl(ILoggerFactory loggerFactory)
        {
            _logger = loggerFactory.CreateLogger<SampleActionFilterAttribute>();
        }

        public void OnActionExecuting(ActionExecutingContext context)
        {
            _logger.LogInformation("Business action starting...");
            // perform some business logic work
        }

        public void OnActionExecuted(ActionExecutedContext context)
        {
            // perform some business logic work
            _logger.LogInformation("Business action completed.");
        }
    }
}

```

Esse filtro pode ser aplicado a classes ou métodos usando a sintaxe `[SampleActionFilter]`, em vez de precisar usar `[TypeFilter]` ou `[ServiceFilter]`.

## Filtros de autorização

*Filtros de autorização:*

- Controlam o acesso aos métodos de ação.
- São os primeiros filtros a serem executados no pipeline de filtro.
- Têm um método anterior, mas não têm um método posterior.

Você deve escrever somente um filtro de autorização personalizado se estiver escrevendo sua própria estrutura de autorização. Prefira configurar suas políticas de autorização ou escrever uma política de autorização personalizada em vez de escrever um filtro personalizado. A implementação do filtro interno é responsável somente por chamar o sistema de autorização.

Você não deve gerar exceções dentro de filtros de autorização, pois nada tratará a exceção (os filtros de exceção não as tratarão). Considere a possibilidade de emitir um desafio quando ocorrer uma exceção.

Saiba mais sobre [Autorização](#).

## Filtros de recurso

- Implementam a interface `IResourceFilter` ou `IAsyncResourceFilter`.
- Suas execuções encapsulam a maior parte do pipeline de filtro.
- Somente os [Filtros de autorização](#) são executados antes dos Filtros de recurso.

Os filtros de recursos são úteis para causar um curto-circuito na maior parte do trabalho que uma solicitação faz. Por exemplo, um filtro de cache poderá evitar o restante do pipeline se a resposta estiver no cache.

O [filtro de recurso com curto-circuito](#) mostrado anteriormente é um exemplo de filtro de recurso. Outro exemplo é [DisableFormValueModelBindingAttribute](#):

- Essa opção impedirá o model binding de acessar os dados do formulário.
- Ela é útil para uploads de arquivos grandes e para impedir que o formulário seja lido na memória.

## Filtros de ação

### IMPORTANT

Os filtros de ação **não** se aplicam ao Razor Pages. O Razor Pages é compatível com [IPageFilter](#) e [IAsyncPageFilter](#). Para obter mais informações, confira [Métodos de filtro para Páginas Razor](#).

*Filtros de ação:*

- Implementam a interface `IActionFilter` ou `IAsyncActionFilter`.
- A execução deles envolve a execução de métodos de ação.

Veja um exemplo de filtro de ação:

```
public class SampleActionFilter : IActionFilter
{
    public void OnActionExecuting(ActionExecutingContext context)
    {
        // do something before the action executes
    }

    public void OnActionExecuted(ActionExecutedContext context)
    {
        // do something after the action executes
    }
}
```

A classe [ActionExecutingContext](#) fornece as seguintes propriedades:

- `ActionArguments` – permite manipular as entradas para a ação.
- `Controller` – permite manipular a instância do controlador.
- `Result` – defini-lo faz um curto-circuito da execução do método de ação e dos filtros de ação posteriores. Apresentar uma exceção também impede a execução do método de ação e dos filtros posteriores, mas isso é tratado como uma falha, e não como um resultado bem-sucedido.

A classe [ActionExecutedContext](#) fornece `Controller` e `Result`, além das seguintes propriedades:

- `Canceled` – será verdadeiro se a execução da ação tiver sofrido curto-circuito por outro filtro.
- `Exception` – não será nulo se a ação ou um filtro de ação posterior tiver apresentado uma exceção. Definir essa propriedade como nula “manipula” uma exceção efetivamente e `Result` será executado como se tivesse sido retornado do método de ação normalmente.

Para um `IAsyncActionFilter`, uma chamada para o `ActionExecutionDelegate`:

- Executa todos os próximos filtros de ação e o método de ação.
- Retorna `ActionExecutedContext`.

Para fazer um curto-circuito, atribua `ActionExecutingContext.Result` a uma instância de resultado e não chame o `ActionExecutionDelegate`.

A estrutura fornece um `ActionFilterAttribute` abstrato que você pode colocar em uma subclasse.

Você poderá usar um filtro de ação para validar o estado do modelo e retornar erros se o estado for inválido:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;

namespace FiltersSample.Filters
{
    public class ValidateModelAttribute : ActionFilterAttribute
    {
        public override void OnActionExecuting(ActionExecutingContext context)
        {
            if (!context.ModelState.IsValid)
            {
                context.Result = new BadRequestObjectResult(context.ModelState);
            }
        }
    }
}
```

O método `OnActionExecuted` é executado depois do método de ação e pode ver e manipular os resultados da ação por meio da propriedade `ActionExecutedContext.Result`. `ActionExecutedContext.Canceled` será definido como verdadeiro se a execução da ação tiver sofrido curto-circuito por outro filtro.

`ActionExecutedContext.Exception` será definido como um valor não nulo se a ação ou um filtro de ação posterior tiver apresentado uma exceção. Definindo `ActionExecutedContext.Exception` como nulo:

- 'Trata' efetivamente uma exceção.
- `ActionExecutedContext.Result` é executado como se fosse retornado normalmente do método de ação.

## Filtros de exceção

*Filtros de exceção* implementam a interface `IExceptionFilter` ou `IAsyncExceptionFilter`. Eles podem ser usados para implementar políticas de tratamento de erro comuns para um aplicativo.

O exemplo de filtro de exceção a seguir usa uma exibição de erro do desenvolvedor personalizada para exibir detalhes sobre exceções que ocorrem quando o aplicativo está em desenvolvimento:

```

public class CustomExceptionFilterAttribute : ExceptionFilterAttribute
{
    private readonly IHostingEnvironment _hostingEnvironment;
    private readonly IModelMetadataProvider _modelMetadataProvider;

    public CustomExceptionFilterAttribute(
        IHostingEnvironment hostingEnvironment,
        IModelMetadataProvider modelMetadataProvider)
    {
        _hostingEnvironment = hostingEnvironment;
        _modelMetadataProvider = modelMetadataProvider;
    }

    public override void OnException(ExceptionContext context)
    {
        if (!_hostingEnvironment.IsDevelopment())
        {
            // do nothing
            return;
        }
        var result = new ViewResult {ViewName = "CustomError"};
        result.ViewData = new ViewDataDictionary(_modelMetadataProvider, context.ModelState);
        result.ViewData.Add("Exception", context.Exception);
        // TODO: Pass additional detailed data via ViewData
        context.Result = result;
    }
}

```

#### Filtros de exceção:

- Não têm eventos anteriores nem posteriores.
- Implementam `OnException` ou `OnExceptionAsync`.
- Tratam as exceções sem tratamento que ocorrem na criação do controlador, no **model binding**, nos filtros de ação ou nos métodos de ação.
- Não capturam as exceções que ocorrem em Filtros de recurso, em Filtros de resultado ou na execução do Resultado de MVC.

Para tratar uma exceção, defina a propriedade `ExceptionContext.ExceptionHandled` como verdadeiro ou grave uma resposta. Isso interrompe a propagação da exceção. Um Filtro de exceção não pode transformar uma exceção em "êxito". Somente um filtro de ação pode fazer isso.

#### NOTE

No ASP.NET Core 1.1, a resposta não será enviada se você definir `ExceptionHandled` como true e gravar uma resposta. Nesse cenário, o ASP.NET Core 1.0 envia a resposta e o ASP.NET Core 1.1.2 retorna ao comportamento do 1.0. Para obter mais informações, consulte [problema #5594](#) no repositório do GitHub.

#### Filtros de exceção:

- São bons para interceptar as exceções que ocorrem nas ações de MVC.
- Não são tão flexíveis quanto o middleware de tratamento de erro.

Prefira o middleware para tratamento de exceção. Use filtros de exceção somente quando você precisar fazer o tratamento de erro *de forma diferente* com base na ação de MVC escolhida. Por exemplo, seu aplicativo pode ter métodos de ação para os pontos de extremidade da API e para modos de exibição/HTML. Os pontos de extremidade da API podem retornar informações de erro como JSON, enquanto as ações baseadas em modo de exibição podem retornar uma página de erro como HTML.

O `ExceptionFilterAttribute` pode tornar-se uma subclasse.

## Filtros de resultado

- Implementam a interface `IResultFilter` ou `IAsyncResultFilter`.
- A execução deles envolve a execução de resultados de ação.

Veja um exemplo de um filtro de resultado que adiciona um cabeçalho HTTP.

```
public class AddHeaderFilterWithDi : IResultFilter
{
    private ILogger _logger;
    public AddHeaderFilterWithDi	ILoggerFactory loggerFactory)
    {
        _logger = loggerFactory.CreateLogger<AddHeaderFilterWithDi>();
    }

    public void OnResultExecuting(ResultExecutingContext context)
    {
        var headerName = "OnResultExecuting";
        context.HttpContext.Response.Headers.Add(
            headerName, new string[] { "ResultExecutingSuccessfully" });
        _logger.LogInformation($"Header added: {headerName}");
    }

    public void OnResultExecuted(ResultExecutedContext context)
    {
        // Can't add to headers here because response has already begun.
    }
}
```

O tipo de resultado que está sendo executado depende da ação em questão. Uma ação de MVC que retorna um modo de exibição incluiria todo o processamento de Razor como parte do `ViewResult` em execução. Um método de API pode executar alguma serialização como parte da execução do resultado. Saiba mais sobre [resultados de ação](#)

Filtros de resultado são executados somente para resultados bem-sucedidos – quando a ação ou filtros da ação produzem um resultado de ação. Filtros de resultado não são executados quando filtros de exceção tratam uma exceção.

O método `OnResultExecuting` pode fazer o curto-circuito da execução do resultado da ação e dos filtros de resultados posteriores definindo `ResultExecutingContext.Cancel` como verdadeiro. De modo geral, você deve gravar no objeto de resposta ao fazer um curto-círculo para evitar gerar uma resposta vazia. A geração de uma exceção vai:

- Impedir a execução do resultado da ação e dos próximos filtros.
- Ser tratada como uma falha e não como um resultado com êxito.

Quando o método `OnResultExecuted` é executado, a resposta provavelmente foi enviada ao cliente e não pode mais ser alterada (a menos que uma exceção tenha sido apresentada). `ResultExecutedContext.Canceled` será definido como verdadeiro se a execução do resultado da ação tiver sofrido curto-círcuito por outro filtro.

`ResultExecutedContext.Exception` será definido como um valor não nulo se o resultado da ação ou um filtro de resultado posterior tiver apresentado uma exceção. Definir `Exception` para como nulo “trata” uma exceção com eficiência e impede que a exceção seja apresentada novamente pelo MVC posteriormente no pipeline. Quando está tratando uma exceção em um filtro de resultado, talvez você não possa gravar dados na resposta. Se o resultado da ação for apresentado durante sua execução e os cabeçalhos já tiverem sido liberados para o cliente, não haverá nenhum mecanismo confiável para enviar um código de falha.

Para um `IAsyncResultFilter`, uma chamada para `await next` no `ResultExecutionDelegate` executa qualquer

filtro de resultado posterior e o resultado da ação. Para fazer um curto-circuito, defina `ResultExecutingContext.Cancel` para verdadeiro e não chame `ResultExecutionDelegate`.

A estrutura fornece um `ResultFilterAttribute` abstrato que você pode colocar em uma subclasse. A classe `AddHeaderAttribute` mostrada anteriormente é um exemplo de atributo de filtro de resultado.

## Usando middleware no pipeline de filtros

Filtros de recursos funcionam como [middleware](#), no sentido em que envolvem a execução de tudo o que vem depois no pipeline. Mas os filtros diferem do middleware porque fazem parte do MVC, o que significa que eles têm acesso ao contexto e a constructos do MVC.

No ASP.NET Core 1.1, você pode usar o middleware no pipeline de filtros. Talvez você queira fazer isso se tiver um componente de middleware que precisa acessar dados de rota do MVC ou que precisa ser executado somente para determinados controladores ou ações.

Para usar o middleware como um filtro, crie um tipo com um método `Configure` que especifica o middleware que você deseja injetar no pipeline de filtros. Veja um exemplo que usa o middleware de localização para estabelecer a cultura atual para uma solicitação:

```
public class LocalizationPipeline
{
    public void Configure(IApplicationBuilder applicationBuilder)
    {
        var supportedCultures = new[]
        {
            new CultureInfo("en-US"),
            new CultureInfo("fr")
        };

        var options = new RequestLocalizationOptions
        {

            DefaultRequestCulture = new RequestCulture(culture: "en-US", uiCulture: "en-US"),
            SupportedCultures = supportedCultures,
            SupportedUICultures = supportedCultures
        };
        options.RequestCultureProviders = new[]
        {
            new RouteDataRequestCultureProvider() { Options = options }
        };

        applicationBuilder.UseRequestLocalization(options);
    }
}
```

Depois, você pode usar o `MiddlewareFilterAttribute` para executar o middleware para um controlador ou ação selecionada ou para executá-lo globalmente:

```
[Route("{culture}/[controller]/[action]")]
[MiddlewareFilter(typeof(LocalizationPipeline))]
public IActionResult CultureFromRouteData()
{
    return Content($"CurrentCulture:{CultureInfo.CurrentCulture.Name},"
        + $"CurrentUICulture:{CultureInfo.CurrentUICulture.Name}");
}
```

Filtros de middleware são executados no mesmo estágio do pipeline de filtros que filtros de recurso, antes do model binding e depois do restante do pipeline.

## Próximas ações

- Confira Métodos de filtro do Razor Pages
- Para fazer experiências com filtros, [baixe, teste e modifique o exemplo do Github](#).

# Áreas no ASP.NET Core

10/09/2018 • 9 minutes to read • [Edit Online](#)

Por [Dhananjay Kumar](#) e [Rick Anderson](#)

Áreas são um recurso do ASP.NET MVC usado para organizar funcionalidades relacionadas em um grupo como um namespace separado (para roteamento) e uma estrutura de pastas (para exibições). O uso de áreas cria uma hierarquia para fins de roteamento, adicionando outro parâmetro de rota, `area`, a `controller` e `action`.

As áreas fornecem uma maneira de particionar um aplicativo Web ASP.NET Core grande em agrupamentos funcionais menores. Uma área é efetivamente uma estrutura MVC dentro de um aplicativo. Em um projeto MVC, componentes lógicos como Modelo, Controlador e Exibição são mantidos em pastas diferentes e o MVC usa convenções de nomenclatura para criar a relação entre esses componentes. Para um aplicativo grande, pode ser vantajoso particionar o aplicativo em áreas de nível alto separadas de funcionalidade. Por exemplo, um aplicativo de comércio eletrônico com várias unidades de negócios, como check-out, cobrança e pesquisa, etc. Cada uma dessas unidades têm suas próprias exibições de componente lógico, controladores e modelos. Nesse cenário, você pode usar Áreas para partitionar fisicamente os componentes de negócios no mesmo projeto.

Uma área pode ser definida como as menores unidades funcionais em um projeto do ASP.NET Core MVC com seu próprio conjunto de controladores, exibições e modelos.

Considere o uso de Áreas em um projeto MVC quando:

- O aplicativo é composto por vários componentes funcionais de alto nível que devem ser separados logicamente
- Você deseja particionar o projeto MVC para que cada área funcional possa ser trabalhada de forma independente

Recursos de área:

- Um aplicativo ASP.NET Core MVC pode ter qualquer quantidade de áreas.
- Cada área tem seus próprios controladores, modelos e exibições.
- As áreas permitem organizar projetos MVC grandes em vários componentes de alto nível que podem ser trabalhados de forma independente.
- As áreas dão suporte a vários controladores com o mesmo nome, desde que eles tenham `áreas` diferentes.

Vamos dar uma olhada em um exemplo para ilustrar como as Áreas são criadas e usadas. Digamos que você tenha um aplicativo de loja que tem dois agrupamentos distintos de controladores e exibições: Produtos e Serviços. Uma estrutura de pastas comum para isso usando áreas do MVC tem a aparência mostrada abaixo:

- Nome do projeto
  - Áreas
    - Produtos
      - Controladores
        - HomeController.cs

- ManageController.cs
- Exibições
  - Home
    - Index.cshtml
  - Gerenciar
    - Index.cshtml
- Serviços
  - Controladores
    - HomeController.cs
  - Exibições
    - Home
      - Index.cshtml

Quando o MVC tenta renderizar uma exibição em uma Área, por padrão, ele tenta procurar nos seguintes locais:

```
/Areas/<Area-Name>/Views/<Controller-Name>/<Action-Name>.cshtml
/Areas/<Area-Name>/Views/Shared/<Action-Name>.cshtml
/Views/Shared/<Action-Name>.cshtml
```

Esses são os locais padrão que podem ser alterados por meio do `AreaViewLocationFormats` no `Microsoft.AspNetCore.Mvc.Razor.RazorViewEngineOptions`.

Por exemplo, no código abaixo, em vez de ter o nome da pasta como 'Areas', ele foi alterado para 'Categories'.

```
services.Configure<RazorViewEngineOptions>(options =>
{
    options.AreaViewLocationFormats.Clear();
    options.AreaViewLocationFormats.Add("/Categories/{2}/Views/{1}/{0}.cshtml");
    options.AreaViewLocationFormats.Add("/Categories/{2}/Views/Shared/{0}.cshtml");
    options.AreaViewLocationFormats.Add("/Views/Shared/{0}.cshtml");
});
```

Uma coisa importante a ser observada é que a estrutura da pasta `Views` é a única que é considerada importante aqui e, o conteúdo do restante das pastas como `Controllers` e `Models` **não** importa. Por exemplo, você não precisa ter uma pasta `Controllers` e `Models`. Isso funciona porque o conteúdo de `Controllers` e `Models` é apenas um código que é compilado em uma .dll, enquanto o conteúdo de `Views` não é, até que uma solicitação para essa exibição seja feita.

Depois de definir a hierarquia de pastas, você precisa informar ao MVC que cada controlador está associado a uma área. Faça isso decorando o nome do controlador com o atributo `[Area]`.

```

...
namespace MyStore.Areas.Products.Controllers
{
    [Area("Products")]
    public class HomeController : Controller
    {
        // GET: /Products/Home/Index
        public IActionResult Index()
        {
            return View();
        }

        // GET: /Products/Home/Create
        public IActionResult Create()
        {
            return View();
        }
    }
}

```

Configure uma definição de rota que funciona com as áreas recém-criadas. O artigo [Rota para ações do controlador](#) apresenta detalhes de como criar definições de rota, incluindo o uso de rotas convencionais versus rotas de atributo. Neste exemplo, usaremos uma rota convencional. Para fazer isso, abra o arquivo `Startup.cs` e modifique-o adicionando a definição de rota nomeada `areaRoute` abaixo.

```

...
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "areaRoute",
        template: "{area:exists}/{controller=Home}/{action=Index}/{id?}");

    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});

```

Navegando para `http://<yourApp>/products`, o método de ação `Index` do `HomeController` na área `Products` será invocado.

## Geração de link

- Geração de links em uma ação dentro de um controlador baseado em área para outra ação no mesmo controlador.

Digamos que o caminho da solicitação atual seja semelhante a `/Products/Home/Create`

Sintaxe de `HtmlHelper`: `@Html.ActionLink("Go to Product's Home Page", "Index")`

Sintaxe de `TagHelper`: `<a asp-action="Index">Go to Product's Home Page</a>`

Observe que não é necessário fornecer os valores 'area' e 'controller' aqui, pois já estão disponíveis no contexto da solicitação atual. Esses tipos de valores são chamados valores `ambient`.

- Geração de links em uma ação dentro de um controlador baseado em área para outra ação em outro controlador

Digamos que o caminho da solicitação atual seja semelhante a `/Products/Home/Create`

Sintaxe de `HtmlHelper`: `@Html.ActionLink("Go to Manage Products Home Page", "Index", "Manage")`

Sintaxe de TagHelper:

```
<a asp-controller="Manage" asp-action="Index">Go to Manage Products Home Page</a>
```

Observe que aqui o valor de ambiente de uma "area" é usado, mas o valor de 'controller' é especificado de forma explícita acima.

- Geração de links em uma ação dentro de um controlador baseado em área para outra ação em outro controlador e outra área.

Digamos que o caminho da solicitação atual seja semelhante a /Products/Home/Create

Sintaxe de HtmlHelper:

```
@Html.ActionLink("Go to Services Home Page", "Index", "Home", new { area = "Services" })
```

Sintaxe de TagHelper:

```
<a asp-area="Services" asp-controller="Home" asp-action="Index">Go to Services Home Page</a>
```

Observe que aqui nenhum valor de ambiente é usado.

- Geração de links em uma ação dentro de um controlador baseado em área para outra ação em outro controlador e **não** em uma área.

Sintaxe de HtmlHelper:

```
@Html.ActionLink("Go to Manage Products Home Page", "Index", "Home", new { area = "" })
```

Sintaxe de TagHelper:

```
<a asp-area="" asp-controller="Manage" asp-action="Index">Go to Manage Products Home Page</a>
```

Como queremos gerar links para uma ação do controlador não baseada em área, esvaziamos o valor de ambiente para 'area' aqui.

## Publicando áreas

Todos os arquivos `*.cshtml` e `wwwroot/**` são publicados na saída quando

```
<Project Sdk="Microsoft.NET.Sdk.Web">
```

é incluído no arquivo `.csproj`.

# Partes do aplicativo no ASP.NET Core

30/10/2018 • 7 minutes to read • [Edit Online](#)

## Exibir ou baixar código de exemplo (como baixar)

Uma *Parte do aplicativo* é uma abstração dos recursos de um aplicativo, da qual recursos de MVC como controladores, componentes de exibição ou auxiliares de marca podem ser descobertos. Um exemplo de uma parte do aplicativo é um `AssemblyPart`, que encapsula uma referência de assembly e expõe tipos e referências de compilação. *Provedores de recursos* funcionam com as partes do aplicativo para preencher os recursos de um aplicativo do ASP.NET Core MVC. O caso de uso principal das partes do aplicativo é permitir que você configure seu aplicativo para descobrir (ou evitar o carregamento) recursos de MVC de um assembly.

## Introdução às partes do aplicativo

Aplicativos de MVC carregam seus recursos das [partes do aplicativo](#). Em particular, a classe `AssemblyPart` representa uma parte de aplicativo que é apoiada por um assembly. Você pode usar essas classes para descobrir e carregar recursos de MVC, como controladores, componentes de exibição, auxiliares de marca e fontes de compilação do Razor. O `ApplicationPartManager` é responsável por controlar as partes do aplicativo e os provedores de recursos disponíveis para o aplicativo MVC. Você pode interagir com o `ApplicationPartManager` em `Startup` quando configura o MVC:

```
// create an assembly part from a class's assembly
var assembly = typeof(Startup).GetTypeInfo().Assembly;
services.AddMvc()
    .AddApplicationPart(assembly);

// OR
var assembly = typeof(Startup).GetTypeInfo().Assembly;
var part = new AssemblyPart(assembly);
services.AddMvc()
    .ConfigureApplicationPartManager(apm => apm.ApplicationParts.Add(part));
```

Por padrão, o MVC pesquisa a árvore de dependência e localiza controladores (mesmo em outros assemblies). Para carregar um assembly arbitrário (por exemplo, de um plug-in que não é referenciado em tempo de compilação), você pode usar uma parte do aplicativo.

Você pode usar as partes do aplicativo para *evitar* pesquisar controladores em um determinado assembly ou local. Você pode controlar quais partes (ou assemblies) ficam disponíveis para o aplicativo modificando a coleção `ApplicationParts` do `ApplicationPartManager`. A ordem das entradas na coleção `ApplicationParts` não é importante. É importante configurar totalmente o `ApplicationPartManager` antes de usá-lo para configurar serviços no contêiner. Por exemplo, você deve configurar totalmente o `ApplicationPartManager` antes de invocar `AddControllersAsServices`. Se isso não for feito, os controladores em partes do aplicativo adicionadas depois da chamada de método não serão afetados (não serão registrados como serviços), o que poderá resultar em um comportamento incorreto do aplicativo.

Se tiver um assembly que contém controladores que você não deseja usar, remova-o do `ApplicationPartManager`:

```

services.AddMvc()
    .ConfigureApplicationPartManager(apm =>
{
    var dependentLibrary = apm.ApplicationParts
        .FirstOrDefault(part => part.Name == "DependentLibrary");

    if (dependentLibrary != null)
    {
        apm.ApplicationParts.Remove(dependentLibrary);
    }
})

```

Além do assembly de seu projeto e de seus assemblies dependentes, o `ApplicationPartManager` incluirá partes para `Microsoft.AspNetCore.Mvc.TagHelpers` e `Microsoft.AspNetCore.Mvc.Razor` por padrão.

## Provedores de recursos do aplicativo

Os Provedores de recursos do aplicativo examinam as partes do aplicativo e fornece recursos para essas partes. Há provedores de recursos internos para os seguintes recursos de MVC:

- [Controladores](#)
- [Referência de metadados](#)
- [Auxiliares de marcação](#)
- [Componentes da exibição](#)

Provedores de recursos herdam de `IApplicationFeatureProvider<T>`, em que `T` é o tipo do recurso. Você pode implementar seus próprios provedores de recursos para qualquer um dos tipos de recurso do MVC listados acima. A ordem dos provedores de recursos na coleção `ApplicationPartManager.FeatureProviders` pode ser importante, pois provedores posteriores podem reagir às ações tomadas por provedores anteriores.

### Exemplo: recurso de controlador genérico

Por padrão, o ASP.NET Core MVC ignora controladores genéricos (por exemplo, `SomeController<T>`). Este exemplo usa um provedor de recursos de controlador que é executado depois do provedor padrão e adiciona instâncias de controlador genérico a uma lista de tipos especificados (definidos em `EntityTypes.Types`):

```

public class GenericControllerFeatureProvider : IApplicationFeatureProvider<ControllerFeature>
{
    public void PopulateFeature(IEnumerable<ApplicationPart> parts, ControllerFeature feature)
    {
        // This is designed to run after the default ControllerTypeProvider,
        // so the list of 'real' controllers has already been populated.
        foreach (var entityType in EntityTypes.Types)
        {
            var typeName = entityType.Name + "Controller";
            if (!feature.Controllers.Any(t => t.Name == typeName))
            {
                // There's no 'real' controller for this entity, so add the generic version.
                var controllerType = typeof(GenericController<>)
                    .MakeGenericType(entityType.AsType()).GetTypeInfo();
                feature.Controllers.Add(controllerType);
            }
        }
    }
}

```

Os tipos de entidade:

```

public static class EntityTypes
{
    public static IReadOnlyList<TypeInfo> Types => new List<TypeInfo>()
    {
        typeof(Sprocket).GetTypeInfo(),
        typeof(Widget).GetTypeInfo(),
    };

    public class Sprocket { }
    public class Widget { }
}

```

O provedor de recursos é adicionado em `Startup`:

```

services.AddMvc()
    .ConfigureApplicationPartManager(apm =>
        apm.FeatureProviders.Add(new GenericControllerFeatureProvider()));

```

Por padrão, os nomes do controlador genérico usados para roteamento seriam no formato `GenericController`1[Widget]` em vez de `Widget`. O atributo a seguir é usado para modificar o nome para que ele corresponda ao tipo genérico usado pelo controlador:

```

using Microsoft.AspNetCore.Mvc.ApplicationModels;
using System;

namespace AppPartsSample
{
    // Used to set the controller name for routing purposes. Without this convention the
    // names would be like 'GenericController`1[Widget]' instead of 'Widget'.
    //
    // Conventions can be applied as attributes or added to MvcOptions.Conventions.
    [AttributeUsage(AttributeTargets.Class, AllowMultiple = false, Inherited = true)]
    public class GenericControllerNameConvention : Attribute, IControllerModelConvention
    {
        public void Apply(ControllerModel controller)
        {
            if (controller.ControllerType.GetGenericTypeDefinition() !=
                typeof(GenericController<>))
            {
                // Not a GenericController, ignore.
                return;
            }

            var entityType = controller.ControllerType.GenericTypeArguments[0];
            controller.ControllerName = entityType.Name;
        }
    }
}

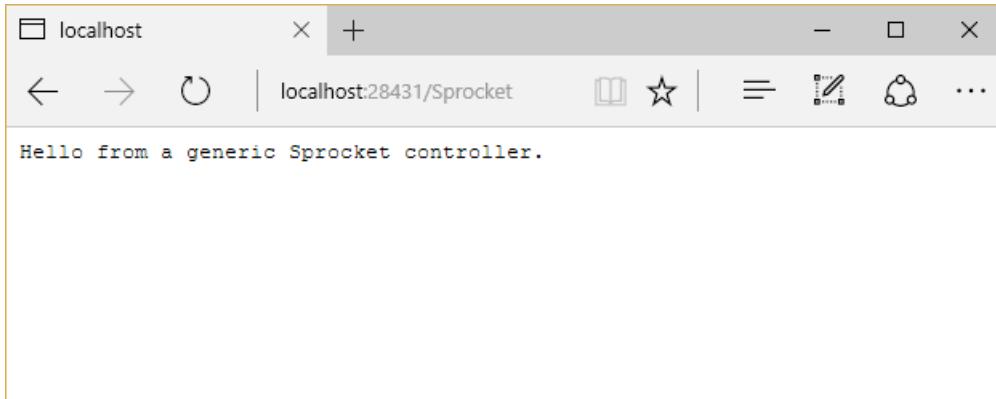
```

A classe `GenericController`:

```
using Microsoft.AspNetCore.Mvc;

namespace AppPartsSample
{
    [GenericControllerNameConvention] // Sets the controller name based on typeof(T).Name
    public class GenericController<T> : Controller
    {
        public IActionResult Index()
        {
            return Content($"Hello from a generic {typeof(T).Name} controller.");
        }
    }
}
```

O resultado, quando uma rota correspondente é solicitada:



### Exemplo: exibir recursos disponíveis

Você pode iterar nos recursos preenchidos disponíveis para seu aplicativo solicitando um `ApplicationPartManager` por meio da [injeção de dependência](#) e usando-o para preencher as instâncias dos recursos apropriados:

```

using AppPartsSample.ViewModels;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.ApplicationParts;
using Microsoft.AspNetCore.Mvc.Controllers;
using System.Linq;
using Microsoft.AspNetCore.Mvc.Razor.Compilation;
using Microsoft.AspNetCore.Mvc.Razor.TagHelpers;
using Microsoft.AspNetCore.Mvc.ViewComponents;

namespace AppPartsSample.Controllers
{
    public class FeaturesController : Controller
    {
        private readonly ApplicationPartManager _partManager;

        public FeaturesController(ApplicationPartManager partManager)
        {
            _partManager = partManager;
        }

        public IActionResult Index()
        {
            var viewModel = new FeaturesViewModel();

            var controllerFeature = new ControllerFeature();
            _partManager.PopulateFeature(controllerFeature);
            viewModel.Controllers = controllerFeature.Controllers.ToList();

            var metaDataReferenceFeature = new MetadataReferenceFeature();
            _partManager.PopulateFeature(metaDataReferenceFeature);
            viewModel.MetadataReferences = metaDataReferenceFeature.MetadataReferences
                .ToList();

            var tagHelperFeature = new TagHelperFeature();
            _partManager.PopulateFeature(tagHelperFeature);
            viewModel.TagHelpers = tagHelperFeature.TagHelpers.ToList();

            var viewComponentFeature = new ViewComponentFeature();
            _partManager.PopulateFeature(viewComponentFeature);
            viewModel.ViewComponents = viewComponentFeature.ViewComponents.ToList();

            return View(viewModel);
        }
    }
}

```

Saída de exemplo:

Home Page - AppPartS: X +

localhost:28431/Features

Features

**Controllers:**

- FeaturesController
- HomeController
- HelloController
- GenericController`1
- GenericController`1

**Metadata References:**

- C:\dev\ssmith\github.com\aspnet-docs\aspnetcore\mvc\extensibility\app-parts\sample\src\AppPartSample\bin\Debug\netcoreapp1.0\AppPartSample.dll
- C:\Users\steve\_000\.nuget\packages\Microsoft.AspNetCore.Antiforgery\1.0.1\lib\netstandard1.3\Microsoft.AspNetCore.Antiforgery.dll
- C:\Users\steve\_000\.nuget\packages\Microsoft.AspNetCore.Authorization\1.0.0

**Tag Helpers:**

- AnchorTagHelper
- CacheTagHelper
- DistributedCacheTagHelper
- EnvironmentTagHelper
- FormTagHelper

**View Components:**

- AnchorTagHelper
- CacheTagHelper
- DistributedCacheTagHelper
- EnvironmentTagHelper
- FormTagHelper

[Home](#)

# Model binding personalizado no ASP.NET Core

22/11/2018 • 12 minutes to read • [Edit Online](#)

Por [Steve Smith](#)

O model binding permite que as ações do controlador funcionem diretamente com tipos de modelo (passados como argumentos de método), em vez de solicitações HTTP. O mapeamento entre os dados de solicitação de entrada e os modelos de aplicativo é manipulado por associadores de modelos. Os desenvolvedores podem estender a funcionalidade de model binding interna implementando associadores de modelos personalizados (embora, normalmente, você não precise escrever seu próprio provedor).

[Exibir ou baixar a amostra do GitHub](#)

## Limitações dos associadores de modelos padrão

Os associadores de modelos padrão dão suporte à maioria dos tipos de dados comuns do .NET Core e devem atender à maior parte das necessidades dos desenvolvedores. Eles esperam associar a entrada baseada em texto da solicitação diretamente a tipos de modelo. Talvez seja necessário transformar a entrada antes de associá-la. Por exemplo, quando você tem uma chave que pode ser usada para pesquisar dados de modelo. Use um associador de modelos personalizado para buscar dados com base na chave.

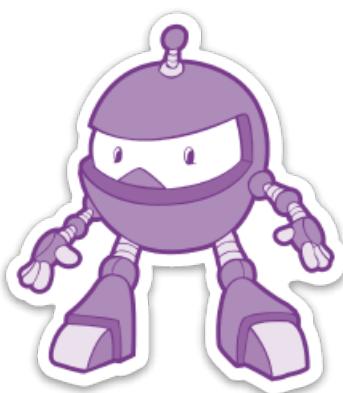
## Análise do model binding

O model binding usa definições específicas para os tipos nos quais opera. Um *tipo simples* é convertido de uma única cadeia de caracteres na entrada. Um *tipo complexo* é convertido de vários valores de entrada. A estrutura determina a diferença de acordo com a existência de um `TypeConverter`. Recomendamos que você crie um conversor de tipo se tiver um mapeamento `string` -> `SomeType` simples que não exige recursos externos.

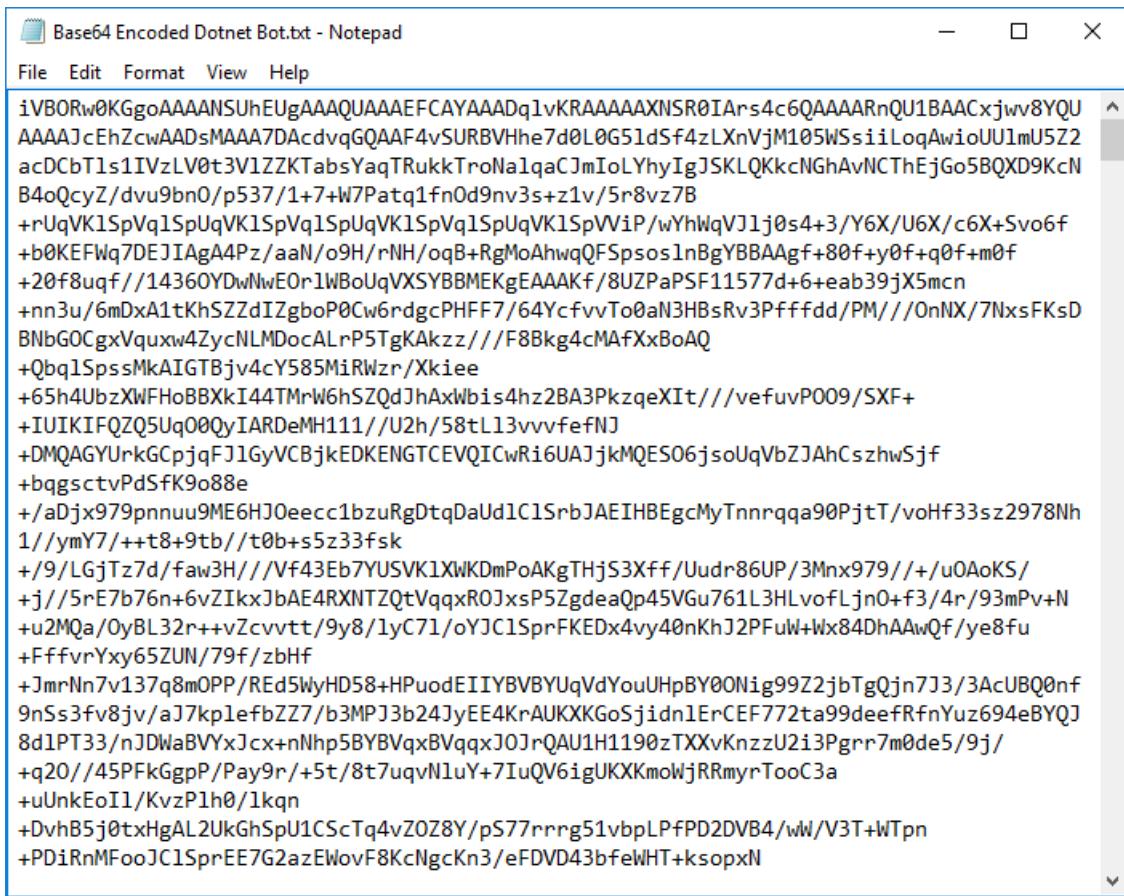
Antes de criar seu próprio associador de modelos personalizado, vale a pena analisar como os associadores de modelos existentes são implementados. Considere o [ByteArrayModelBinder](#), que pode ser usado para converter cadeias de caracteres codificadas em Base64 em matrizes de bytes. As matrizes de bytes costumam ser armazenadas como arquivos ou campos BLOB do banco de dados.

### Trabalhando com o `ByteArrayModelBinder`

Cadeias de caracteres codificadas em Base64 podem ser usadas para representar dados binários. Por exemplo, a imagem a seguir pode ser codificada como uma cadeia de caracteres.



Uma pequena parte da cadeia de caracteres codificada é mostrada na seguinte imagem:



A screenshot of a Windows Notepad window titled "Base64 Encoded Dotnet Bot.txt - Notepad". The window contains a single line of extremely long and complex Base64 encoded text. The text starts with "iVBORw0KGgoAAAANSUhEUgAAAQAAAECAYAAADq1vKRAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQU" and continues for several hundred characters.

Siga as instruções do [LEIAME da amostra](#) para converter a cadeia de caracteres codificada em Base64 em um arquivo.

O ASP.NET Core MVC pode usar uma cadeia de caracteres codificada em Base64 e usar um `ByteArrayModelBinder` para convertê-la em uma matriz de bytes. O `ByteArrayModelBinderProvider` que implementa `IModelBinderProvider` mapeia argumentos `byte[]` para `ByteArrayModelBinder`:

```
public IModelBinder GetBinder(ModelBinderProviderContext context)
{
    if (context == null)
    {
        throw new ArgumentNullException(nameof(context));
    }

    if (context.Metadata.ModelType == typeof(byte[]))
    {
        return new ByteArrayModelBinder();
    }

    return null;
}
```

Ao criar seu próprio associador de modelos personalizado, você pode implementar seu próprio tipo `IModelBinderProvider` ou usar o [ModelBinderAttribute](#).

O seguinte exemplo mostra como usar `ByteArrayModelBinder` para converter uma cadeia de caracteres codificada em Base64 em um `byte[]` e salvar o resultado em um arquivo:

```
// POST: api/image
[HttpPost]
public void Post(byte[] file, string filename)
{
    string filePath = Path.Combine(_env.ContentRootPath, "wwwroot/images/upload", filename);
    if (System.IO.File.Exists(filePath)) return;
    System.IO.File.WriteAllBytes(filePath, file);
}
```

Execute POST em uma cadeia de caracteres codificada em Base64 para esse método de API usando uma ferramenta como o [Postman](#):

Key	Value	Bulk Edit
<input checked="" type="checkbox"/> file	iVBORw0KGgoAAAANSUhEUgAAAQUAAAECAYAAADqlvKRAAAAXNSR...	
<input checked="" type="checkbox"/> filename	bot.png	
New key	value	

Desde que o associador possa associar dados de solicitação a propriedades ou argumentos nomeados de forma adequada, o model binding terá êxito. O seguinte exemplo mostra como usar `ByteArrayModelBinder` com um modelo de exibição:

```
[HttpPost("Profile")]
public void SaveProfile(ProfileViewModel model)
{
    string filePath = Path.Combine(_env.ContentRootPath, "wwwroot/images/upload", model.FileName);
    if (System.IO.File.Exists(model.FileName)) return;
    System.IO.File.WriteAllBytes(filePath, model.File);
}

public class ProfileViewModel
{
    public byte[] File { get; set; }
    public string FileName { get; set; }
}
```

## Amostra de associador de modelos personalizado

Nesta seção, implementaremos um associador de modelos personalizado que:

- Converte dados de solicitação de entrada em argumentos de chave fortemente tipados.
- Usa o Entity Framework Core para buscar a entidade associada.

- Passa a entidade associada como um argumento para o método de ação.

A seguinte amostra usa o atributo `ModelBinder` no modelo `Author`:

```
using CustomModelBindingSample.Binders;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace CustomModelBindingSample.Data
{
    [ModelBinder(BinderType = typeof(AuthorEntityBinder))]
    public class Author
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string GitHub { get; set; }
        public string Twitter { get; set; }
        public string BlogUrl { get; set; }
    }
}
```

No código anterior, o atributo `ModelBinder` especifica o tipo de `IModelBinder` que deve ser usado para associar parâmetros de ação `Author`.

A classe `AuthorEntityBinder` a seguir associa um parâmetro `Author` efetuando fetch da entidade de uma fonte de dados usando o Entity Framework Core e um `authorId`:

```

public class AuthorEntityBinder : IModelBinder
{
    private readonly AppDbContext _db;
    public AuthorEntityBinder(AppDbContext db)
    {
        _db = db;
    }

    public Task BindModelAsync(ModelBindingContext bindingContext)
    {
        if (bindingContext == null)
        {
            throw new ArgumentNullException(nameof(bindingContext));
        }

        var modelName = bindingContext.ModelName;

        // Try to fetch the value of the argument by name
        var valueProviderResult =
            bindingContext.ValueProvider.GetValue(modelName);

        if (valueProviderResult == ValueProviderResult.None)
        {
            return Task.CompletedTask;
        }

        bindingContext.ModelState.SetModelValue(modelName,
            valueProviderResult);

        var value = valueProviderResult.FirstValue;

        // Check if the argument value is null or empty
        if (string.IsNullOrEmpty(value))
        {
            return Task.CompletedTask;
        }

        int id = 0;
        if (!int.TryParse(value, out id))
        {
            // Non-integer arguments result in model state errors
            bindingContext.ModelState.TryAddModelError(
                modelName,
                "Author Id must be an integer.");
            return Task.CompletedTask;
        }

        // Model will be null if not found, including for
        // out of range id values (0, -3, etc.)
        var model = _db.Authors.Find(id);
        bindingContext.Result = ModelBindingResult.Success(model);
        return Task.CompletedTask;
    }
}

```

#### NOTE

A classe `AuthorEntityBinder` precedente é destinada a ilustrar um associador de modelos personalizado. A classe não é destinada a ilustrar as melhores práticas para um cenário de pesquisa. Para pesquisa, associe o `authorId` e consulte o banco de dados em um método de ação. Essa abordagem separa falhas de model binding de casos de `NotFound`.

O seguinte código mostra como usar o `AuthorEntityBinder` em um método de ação:

```
[HttpGet("get/{authorId}")]  
public IActionResult Get(Author author)  
{  
    return Ok(author);  
}
```

O atributo `ModelBinder` pode ser usado para aplicar o `AuthorEntityBinder` aos parâmetros que não usam convenções padrão:

```
[HttpGet("{id}")]  
public IActionResult GetById([ModelBinder(Name = "id")]Author author)  
{  
    if (author == null)  
    {  
        return NotFound();  
    }  
    if (!ModelState.IsValid)  
    {  
        return BadRequest(ModelState);  
    }  
    return Ok(author);  
}
```

Neste exemplo, como o nome do argumento não é o `authorId` padrão, ele é especificado no parâmetro com o atributo `ModelBinder`. Observe que o controlador e o método de ação são simplificados, comparado à pesquisa da entidade no método de ação. A lógica para buscar o autor usando o Entity Framework Core é movida para o associador de modelos. Isso pode ser uma simplificação considerável quando há vários métodos associados ao modelo `Author` e pode ajudá-lo a seguir o princípio DRY.

Aplique o atributo `ModelBinder` a propriedades de modelo individuais (como em um viewmodel) ou a parâmetros de método de ação para especificar um associador de modelos ou nome de modelo específico para apenas esse tipo ou essa ação.

### Implementando um ModelBinderProvider

Em vez de aplicar um atributo, você pode implementar `IModelBinderProvider`. É assim que os associadores de estrutura interna são implementados. Quando você especifica o tipo no qual o associador opera, você especifica o tipo de argumento que ele produz, **não** a entrada aceita pelo associador. O provedor de associador a seguir funciona com o `AuthorEntityBinder`. Quando ele for adicionado à coleção do MVC de provedores, não será necessário usar o atributo `ModelBinder` nos parâmetros `Author` ou de tipo `Author`.

```

using CustomModelBindingSample.Data;
using Microsoft.AspNetCore.Mvc.ModelBinding;
using Microsoft.AspNetCore.Mvc.ModelBinding.Binders;
using System;

namespace CustomModelBindingSample.Binders
{
    public class AuthorEntityBinderProvider : IModelBinderProvider
    {
        public IModelBinder GetBinder(ModelBinderProviderContext context)
        {
            if (context == null)
            {
                throw new ArgumentNullException(nameof(context));
            }

            if (context.Metadata.ModelType == typeof(Author))
            {
                return new BinderTypeModelBinder(typeof(AuthorEntityBinder));
            }

            return null;
        }
    }
}

```

Observação: o código anterior retorna um `BinderTypeModelBinder`. O `BinderTypeModelBinder` atua como um alocador para associadores de modelos e fornece a DI (injeção de dependência). O `AuthorEntityBinder` exige que a DI acesse o EF Core. Use `BinderTypeModelBinder` se o associador de modelos exigir serviços da DI.

Para usar um provedor de associador de modelos personalizado, adicione-o a `ConfigureServices`:

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<AppDbContext>(options => options.UseInMemoryDatabase());

    services.AddMvc(options =>
    {
        // add custom binder to beginning of collection
        options.ModelBinderProviders.Insert(0, new AuthorEntityBinderProvider());
    });
}

```

Ao avaliar associadores de modelos, a coleção de provedores é examinada na ordem. O primeiro provedor que retorna um associador é usado.

A imagem a seguir mostra os associadores de modelos padrão do depurador.

Name	Value
options.ModelBind	Count = 14
[0]	{Microsoft.AspNetCore.Mvc.ModelBinding.BindersBinderTypeModelBinderProvider}
[1]	{Microsoft.AspNetCore.Mvc.ModelBinding.BindersServicesModelBinderProvider}
[2]	{Microsoft.AspNetCore.Mvc.ModelBinding.BindersBodyModelBinderProvider}
[3]	{Microsoft.AspNetCore.Mvc.ModelBinding.BindersHeaderModelBinderProvider}
[4]	{Microsoft.AspNetCore.Mvc.ModelBinding.BindersSimpleTypeModelBinderProvider}
[5]	{Microsoft.AspNetCore.Mvc.ModelBinding.BindersCancellationTokenModelBinderProvider}
[6]	{Microsoft.AspNetCore.Mvc.ModelBinding.BindersByteArrayModelBinderProvider}
[7]	{Microsoft.AspNetCore.Mvc.ModelBinding.BindersFormFileModelBinderProvider}
[8]	{Microsoft.AspNetCore.Mvc.ModelBinding.BindersFormCollectionModelBinderProvider}
[9]	{Microsoft.AspNetCore.Mvc.ModelBinding.BindersKeyValuePairModelBinderProvider}
[10]	{Microsoft.AspNetCore.Mvc.ModelBinding.BindersDictionaryModelBinderProvider}
[11]	{Microsoft.AspNetCore.Mvc.ModelBinding.BindersArrayModelBinderProvider}
[12]	{Microsoft.AspNetCore.Mvc.ModelBinding.BindersCollectionModelBinderProvider}
[13]	{Microsoft.AspNetCore.Mvc.ModelBinding.BindersComplexTypeModelBinderProvider}

A adição do provedor ao final da coleção pode resultar na chamada a um associador de modelos interno antes que o associador personalizado tenha uma oportunidade. Neste exemplo, o provedor personalizado é adicionado ao início da coleção para garantir que ele é usado para argumentos de ação `Author`.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<AppDbContext>(options => options.UseInMemoryDatabase());

    services.AddMvc(options =>
    {
        // add custom binder to beginning of collection
        options.ModelBinderProviders.Insert(0, new AuthorEntityBinderProvider());
    });
}
```

## Recomendações e melhores práticas

Associadores de modelos personalizados:

- Não devem tentar definir códigos de status ou retornar resultados (por exemplo, 404 Não Encontrado). Se o model binding falhar, um [filtro de ação](#) ou uma lógica no próprio método de ação deverá resolver a falha.
- São muito úteis para eliminar código repetitivo e interesses paralelos de métodos de ação.
- Normalmente, não devem ser usados para converter uma cadeia de caracteres em um tipo personalizado; um [TypeConverter](#) geralmente é uma opção melhor.

# Versão de compatibilidade do ASP.NET Core MVC

10/01/2019 • 3 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

O método `SetCompatibilityVersion` permite que um aplicativo aceite ou recuse as possíveis alterações da falha de comportamento introduzidas no ASP.NET Core MVC 2.1 ou posteriores. Essas possíveis alterações da falha de comportamento geralmente afetam como o subsistema MVC se comporta e como **o código do usuário** é chamado pelo tempo de execução. Ao aceitar, você obtém o comportamento mais recente e o comportamento de longo prazo do ASP.NET Core.

O código a seguir define o modo de compatibilidade para o ASP.NET Core 2.2:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc()
        .SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
}
```

É recomendável que você teste seu aplicativo usando a versão mais recente (`CompatibilityVersion.Version_2_2`). Estimamos que a maioria dos aplicativos não terão alterações da falha de comportamento usando a versão mais recente.

Os aplicativos que chamam `SetCompatibilityVersion(CompatibilityVersion.Version_2_0)` são protegidos contra as possíveis alterações da falha de comportamento apresentadas no ASP.NET Core 2.1 MVC e nas versões 2.x posteriores. Essa proteção:

- Não se aplica a todas as alterações da 2.1 e posteriores, ela é direcionada às possíveis alterações da falha de comportamento do tempo de execução do ASP.NET Core no subsistema de MVC.
- Não se estende à próxima versão principal.

A compatibilidade padrão para aplicativos ASP.NET Core 2.1 e 2.x posteriores que **não** é chamada é a `SetCompatibilityVersion` compatibilidade 2.0. Ou seja, não chamar `SetCompatibilityVersion` é o mesmo que chamar `SetCompatibilityVersion(CompatibilityVersion.Version_2_0)`.

O código a seguir define o modo de compatibilidade para o ASP.NET Core 2.2, exceto para os seguintes comportamentos:

- `AllowCombiningAuthorizeFilters`
- `InputFormatterExceptionPolicy`

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc()
        // Include the 2.2 behaviors
        .SetCompatibilityVersion(CompatibilityVersion.Version_2_2)
        // Except for the following.
        .AddMvcOptions(options =>
    {
        // Don't combine authorize filters (keep 2.0 behavior).
        options.AllowCombiningAuthorizeFilters = false;
        // All exceptions thrown by an IInputFormatter are treated
        // as model state errors (keep 2.0 behavior).
        options.InputFormatterExceptionPolicy =
            InputFormatterExceptionPolicy.AllExceptions;
    });
}
```

Para aplicativos que encontram alterações da falha de comportamento, o uso das opções de compatibilidade apropriadas:

- Permite que você use a versão mais recente e recuse alterações da falha de comportamento específicas.
- Tenha tempo para atualizar seu aplicativo para que ele funcione com as alterações mais recentes.

Os comentários do código-fonte da classe [MvcOptions](#) têm uma boa explicação sobre o que mudou e por que as alterações são uma melhoria para a maioria dos usuários.

Futuramente, haverá uma [versão 3.0 do ASP.NET Core](#). Os comportamentos antigos compatíveis por meio de opções de compatibilidade serão removidos na versão 3.0. Consideramos que essas são alterações positivas que beneficiam quase todos os usuários. Com a introdução dessas alterações, a maioria dos aplicativos poderá se beneficiar agora e os demais terão tempo para serem atualizados.

# Criar APIs Web com o ASP.NET Core

17/01/2019 • 13 minutes to read • [Edit Online](#)

Por [Scott Addie](#)

[Exibir ou baixar código de exemplo \(como baixar\)](#)

Este documento explica como criar uma API Web no ASP.NET Core e quando é mais adequado usar cada recurso.

## Derivar a classe por meio do ControllerBase

Herde da classe [ControllerBase](#) em um controlador destinado a funcionar como uma API Web. Por exemplo:

```
[Produces("application/json")]
[Route("api/[controller]")]
public class PetsController : ControllerBase
{
    private readonly PetsRepository _repository;

    public PetsController(PetsRepository repository)
    {
        _repository = repository;
    }

    [HttpGet]
    public async Task<ActionResult<List<Pet>>> GetAllAsync()
    {
        return await _repository.GetPetsAsync();
    }

    [HttpGet("{id}")]
    [ProducesResponseType(404)]
    public async Task<ActionResult<Pet>> GetByIdAsync(int id)
    {
        var pet = await _repository.GetPetAsync(id);

        if (pet == null)
        {
            return NotFound();
        }

        return pet;
    }

    [HttpPost]
    [ProducesResponseType(400)]
    public async Task<ActionResult<Pet>> CreateAsync(Pet pet)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        await _repository.AddPetAsync(pet);

        return CreatedAtAction(nameof(GetByIdAsync),
            new { id = pet.Id }, pet);
    }
}
```

```

[Produces("application/json")]
[Route("api/[controller]")]
public class PetsController : ControllerBase
{
    private readonly PetsRepository _repository;

    public PetsController(PetsRepository repository)
    {
        _repository = repository;
    }

    [HttpGet]
    [ProducesResponseType(typeof(IEnumerable<Pet>), 200)]
    public async Task<IActionResult> GetAllAsync()
    {
        var pets = await _repository.GetPetsAsync();

        return Ok(pets);
    }

    [HttpGet("{id}")]
    [ProducesResponseType(typeof(Pet), 200)]
    [ProducesResponseType(404)]
    public async Task<IActionResult> GetByIdAsync(int id)
    {
        var pet = await _repository.GetPetAsync(id);

        if (pet == null)
        {
            return NotFound();
        }

        return Ok(pet);
    }

    [HttpPost]
    [ProducesResponseType(typeof(Pet), 201)]
    [ProducesResponseType(400)]
    public async Task<IActionResult> CreateAsync([FromBody] Pet pet)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        await _repository.AddPetAsync(pet);

        return CreatedAtAction(nameof(GetByIdAsync),
            new { id = pet.Id }, pet);
    }
}

```

A classe  `ControllerBase` fornece acesso a várias propriedades e métodos. No código anterior, os exemplos incluem `BadRequest(ModelStateDictionary)` e `CreatedAtAction(String, Object, Object)`. Esses métodos são chamados em métodos de ação para retornar os códigos de status HTTP 400 e 201, respectivamente. A propriedade  `ModelState`, também fornecida pelo  `ControllerBase`, é acessada para executar a validação do modelo de solicitação.

## Anotação com o atributo ApiController

O ASP.NET Core 2.1 apresenta o atributo `[ApiController]` para denotar uma classe do controlador API Web. Por exemplo:

```
[Route("api/[controller]")]
[ApiController]
public class ProductsController : ControllerBase
```

Uma versão de compatibilidade do 2.1 ou posterior, definida por meio de [SetCompatibilityVersion](#), é necessária para usar esse atributo no nível do controlador. Por exemplo, o código realçado em `Startup.ConfigureServices` define o sinalizador de compatibilidade 2.1:

```
services.AddMvc()
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
```

Para obter mais informações, consulte [Versão de compatibilidade do ASP.NET Core MVC](#).

No ASP.NET Core 2.2 ou posterior, o atributo `[ApiController]` pode ser aplicado a um assembly. A anotação dessa maneira aplica o comportamento da API Web para todos os controladores no assembly. Lembre-se de que não é possível recusar controladores individuais. Como uma recomendação, os atributos de nível de assembly devem ser aplicados à classe `Startup`:

```
[assembly: ApiController]
namespace WebApiSample.Api._22
{
    public class Startup
    {
```

Uma versão de compatibilidade do 2.2 ou posterior, definida por meio de [SetCompatibilityVersion](#), é necessária para usar esse atributo no nível do assembly.

Geralmente, o atributo `[ApiController]` é associado ao `ControllerBase` para habilitar o comportamento específico de REST para controladores. `ControllerBase` fornece acesso a métodos como [NotFound](#) e [File](#).

Outra abordagem é criar uma classe de base de controlador personalizada anotada com o atributo `[ApiController]`:

```
[ApiController]
public class MyBaseController
{
}
```

As seções a seguir descrevem os recursos de conveniência adicionados pelo atributo.

## Respostas automáticas do HTTP 400

Os erros de validação do modelo disparam uma resposta HTTP 400 automaticamente. Consequentemente, o código a seguir se torna desnecessário em suas ações:

```
if (!ModelState.IsValid)
{
    return BadRequest(ModelState);
}
```

Use [InvalidModelStateResponseFactory](#) para personalizar a saída da resposta resultante.

A desabilitação do comportamento padrão é útil quando a ação pode se recuperar de um erro de validação do modelo. O comportamento padrão é desabilitado quando a propriedade [SuppressModelStateInvalidFilter](#) está definida como `true`. Adicione o seguinte código em `Startup.ConfigureServices` após

```
services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_<version_number>); :
```

```
services.AddMvc()
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_2)
    .ConfigureApiBehaviorOptions(options =>
{
    options.SuppressConsumesConstraintForFormFileParameters = true;
    options.SuppressInferBindingSourcesForParameters = true;
    options.SuppressModelStateInvalidFilter = true;
    options.SuppressMapClientErrors = true;

    options.ClientErrorMapping[404].Link =
        "https://httpstatuses.com/404";
});
```

```
services.Configure<ApiBehaviorOptions>(options =>
{
    options.SuppressConsumesConstraintForFormFileParameters = true;
    options.SuppressInferBindingSourcesForParameters = true;
    options.SuppressModelStateInvalidFilter = true;
});
```

Com um sinalizador de compatibilidade do 2.2 ou posterior, o tipo de resposta padrão para respostas HTTP 400 é [ValidationProblemDetails](#). O tipo `ValidationProblemDetails` está em conformidade com a [especificação RFC 7807](#). Defina a propriedade `SuppressUseValidationProblemDetailsForInvalidModelStateResponses` como `true` para retornar o formato de erro do ASP.NET Core 2.1 de [SerializableError](#). Adicione o seguinte código em `Startup.ConfigureServices`:

```
services.AddMvc()
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_2)
    .ConfigureApiBehaviorOptions(options =>
{
    options
        .SuppressUseValidationProblemDetailsForInvalidModelStateResponses = true;
});
```

## Inferência de parâmetro de origem de associação

Um atributo de origem de associação define o local no qual o valor do parâmetro de uma ação é encontrado. Os seguintes atributos da origem da associação existem:

ATRIBUTO	FONTE DE ASSOCIAÇÃO
<a href="#">[FromBody]</a>	Corpo da solicitação
<a href="#">[FromForm]</a>	Dados do formulário no corpo da solicitação
<a href="#">[FromHeader]</a>	Cabeçalho da solicitação
<a href="#">[FromQuery]</a>	Parâmetro de cadeia de caracteres de consulta de solicitação
<a href="#">[FromRoute]</a>	Dados de rota da solicitação atual
<a href="#">[FromServices]</a>	O serviço de solicitação inserido como um parâmetro de ação

**WARNING**

Não use `[FromRoute]` quando os valores puderem conter `%2f` (ou seja, `/`). `%2f` não ficará sem escape para `/`. Use `[FromQuery]`, se o valor puder conter `%2f`.

Sem o atributo `[ApiController]`, os atributos da origem da associação são definidos explicitamente. Sem `[ApiController]` ou outros atributos de origem de associação, como `[FromQuery]`, o tempo de execução do ASP.NET Core tenta usar o associador de modelos de objeto complexo. O associador de modelos de objeto complexo extrai os dados dos provedores de valor (que têm uma ordem definida). Por exemplo, o "associador de modelos de corpo" está sempre definido como aceitar.

No exemplo a seguir, o atributo `[FromQuery]` indica que o valor do parâmetro `discontinuedOnly` é fornecido na cadeia de caracteres de consulta da URL de solicitação:

```
[HttpGet]
public async Task<ActionResult<List<Product>>> GetAsync(
    [FromQuery] bool discontinuedOnly = false)
{
    List<Product> products = null;

    if (discontinuedOnly)
    {
        products = await _repository.GetDiscontinuedProductsAsync();
    }
    else
    {
        products = await _repository.GetProductsAsync();
    }

    return products;
}
```

Regras de inferência são aplicadas para as fontes de dados padrão dos parâmetros de ação. Essas regras configuram as origens da associação que você provavelmente aplicaria manualmente aos parâmetros de ação. Os atributos da origem da associação se comportam da seguinte maneira:

- **[FromBody]** é inferido para parâmetros de tipo complexo. Uma exceção a essa regra é qualquer tipo interno complexo com um significado especial, como `IFormCollection` e `CancellationToken`. O código de inferência da origem da associação ignora esses tipos especiais. `[FromBody]` não é inferido para tipos simples, como `string` ou `int`. Portanto, o atributo `[FromBody]` deve ser usado para tipos simples quando essa funcionalidade for necessária. Quando uma ação tiver mais de um parâmetro especificado explicitamente (via `[FromBody]`) ou inferido como limite do corpo da solicitação, uma exceção será lançada. Por exemplo, as seguintes assinaturas de ação causam uma exceção:

## NOTE

No ASP.NET Core 2.1, os parâmetros de tipo de coleção, como listas e matrizes, são inferidos incorretamente como `[FromQuery]`. `[FromBody]` deverá ser usado para esses parâmetros se eles forem vinculados ao corpo da solicitação. Esse comportamento é corrigido no ASP.NET Core 2.2 ou posterior, onde os parâmetros do tipo de coleção são inferidos para serem vinculados ao corpo por padrão.

- **[FromForm]** é inferido para parâmetros de ação do tipo `IFormFile` e `IFormFileCollection`. Ele não é inferido para qualquer tipo simples ou definido pelo usuário.
- **[FromRoute]** é inferido para qualquer nome de parâmetro de ação correspondente a um parâmetro no modelo de rota. Quando mais de uma rota correspondem a um parâmetro de ação, qualquer valor de rota é considerado `[FromRoute]`.
- **[FromQuery]** é inferido para todos os outros parâmetros de ação.

As regras de inferência padrão são desabilitadas quando a propriedade

`SuppressInferBindingSourcesForParameters` está definida como `true`. Adicione o seguinte código em

`Startup.ConfigureServices` após

```
services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_<version_number>); :
```

```
services.AddMvc()
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_2)
    .ConfigureApiBehaviorOptions(options =>
{
    options.SuppressConsumesConstraintForFormFileParameters = true;
    options.SuppressInferBindingSourcesForParameters = true;
    options.SuppressModelStateInvalidFilter = true;
    options.SuppressMapClientErrors = true;

    options.ClientErrorMapping[404].Link =
        "https://httpstatuses.com/404";
});
```

```
services.Configure<ApiBehaviorOptions>(options =>
{
    options.SuppressConsumesConstraintForFormFileParameters = true;
    options.SuppressInferBindingSourcesForParameters = true;
    options.SuppressModelStateInvalidFilter = true;
});
```

## Inferência de solicitação de várias partes/dados de formulário

Quando um parâmetro de ação é anotado com o atributo `[FromForm]`, o tipo de conteúdo de solicitação `multipart/form-data` é inferido.

O comportamento padrão é desabilitado quando a propriedade

`SuppressConsumesConstraintForFormFileParameters` está definida como `true`.

Adicione o seguinte código em `Startup.ConfigureServices` :

```

services.AddMvc()
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_2)
    .ConfigureApiBehaviorOptions(options =>
{
    options.SuppressConsumesConstraintForFormFileParameters = true;
    options.SuppressInferBindingSourcesForParameters = true;
    options.SuppressModelStateInvalidFilter = true;
    options.SuppressMapClientErrors = true;

    options.ClientErrorMapping[404].Link =
        "https://httpstatuses.com/404";
});

```

Adicione o seguinte código em `Startup.ConfigureServices` após

```
services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
```

```

services.Configure<ApiBehaviorOptions>(options =>
{
    options.SuppressConsumesConstraintForFormFileParameters = true;
    options.SuppressInferBindingSourcesForParameters = true;
    options.SuppressModelStateInvalidFilter = true;
});

```

## Requisito de roteamento de atributo

O roteamento de atributo se torna um requisito. Por exemplo:

```

[Route("api/[controller]")]
[ApiController]
public class ProductsController : ControllerBase

```

As ações são inacessíveis por meio de [rotas convencionais](#) definidas em [UseMvc](#) ou por [UseMvcWithDefaultRoute](#) em `Startup.Configure`.

## Respostas de detalhes de problema para códigos de status de erro

No ASP.NET Core 2.2 ou posterior, o MVC transforma um resultado de erro (um resultado com o código de status 400 ou superior) em um resultado com [ProblemDetails](#). `ProblemDetails` é:

- Um tipo baseado na [especificação RFC 7807](#).
- Um formato padronizado para especificar detalhes do erro legível por computador em uma resposta HTTP.

Considere o seguinte código em uma ação do controlador:

```

if (product == null)
{
    return NotFound();
}

```

A resposta HTTP para `NotFound` tem um código de status 404 com um corpo `ProblemDetails`. Por exemplo:

```

{
    type: "https://tools.ietf.org/html/rfc7231#section-6.5.4",
    title: "Not Found",
    status: 404,
    traceId: "0HLHLV31KRN83:00000001"
}

```

O recurso de detalhes do problema requer um sinalizador de compatibilidade de 2.2 ou posterior. O comportamento padrão é desabilitado quando a propriedade `SuppressMapClientErrors` está definida como `true`. Adicione o seguinte código em `Startup.ConfigureServices`:

```
services.AddMvc()
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_2)
    .ConfigureApiBehaviorOptions(options =>
{
    options.SuppressConsumesConstraintForFormFileParameters = true;
    options.SuppressInferBindingSourcesForParameters = true;
    options.SuppressModelStateInvalidFilter = true;
    options.SuppressMapClientErrors = true;

    options.ClientErrorMapping[404].Link =
        "https://httpstatuses.com/404";
});
```

Use a propriedade `ClientErrorMapping` para configurar o conteúdo da resposta `ProblemDetails`. Por exemplo, o código a seguir atualiza a propriedade `type` para respostas 404:

```
services.AddMvc()
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_2)
    .ConfigureApiBehaviorOptions(options =>
{
    options.SuppressConsumesConstraintForFormFileParameters = true;
    options.SuppressInferBindingSourcesForParameters = true;
    options.SuppressModelStateInvalidFilter = true;
    options.SuppressMapClientErrors = true;

    options.ClientErrorMapping[404].Link =
        "https://httpstatuses.com/404";
});
```

## Recursos adicionais

- [Tipos de retorno de ação do controlador na API Web ASP.NET Core](#)
- [Formatadores personalizados na API Web ASP.NET Core](#)
- [Formatar dados de resposta na API Web ASP.NET Core](#)
- [Páginas de ajuda da API Web ASP.NET Core com o Swagger/OpenAPI](#)
- [Roteamento para ações do controlador no ASP.NET Core](#)

# Tutorial: Criar uma API Web com o ASP.NET Core MVC

30/01/2019 • 28 minutes to read • [Edit Online](#)

Por [Rick Anderson](#) e [Mike Wasson](#)

Este tutorial ensina os conceitos básicos da criação de uma API Web com o ASP.NET Core.

Neste tutorial, você aprenderá como:

- Criar um projeto de API Web.
- Adicionar uma classe de modelo.
- Criar o contexto de banco de dados.
- Registrar o contexto de banco de dados.
- Adicionar um controlador.
- Adicionar métodos CRUD.
- Configurar o roteamento e caminhos de URL.
- Especificar os valores retornados.
- Chamar a API Web com o Postman.
- Chamar a API Web com o jQuery.

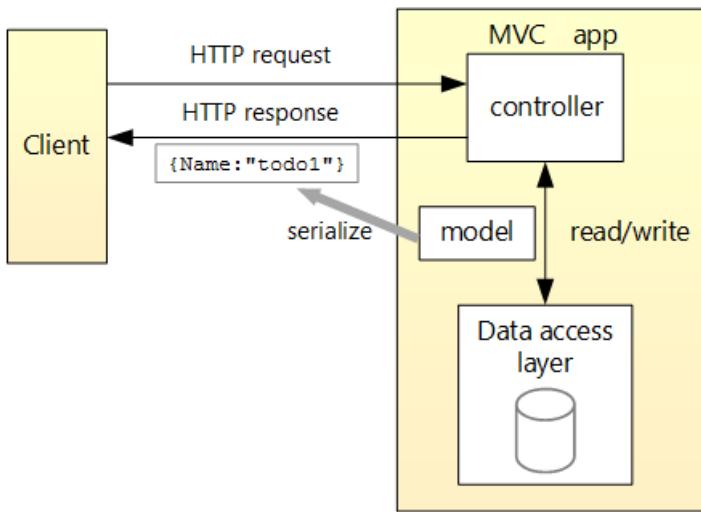
No final, você terá uma API Web que pode gerenciar itens de "tarefas pendentes" armazenados em um banco de dados relacional.

## Visão geral

Este tutorial cria a seguinte API:

API	Descrição	Corpo da solicitação	Corpo da resposta
GET /api/todo	Obter todos os itens de tarefas pendentes	Nenhum	Matriz de itens de tarefas pendentes
GET /api/todo/{id}	Obter um item por ID	Nenhum	Item de tarefas pendentes
POST /api/todo	Adicionar um novo item	Item de tarefas pendentes	Item de tarefas pendentes
PUT /api/todo/{id}	Atualizar um item existente	Item de tarefas pendentes	Nenhum
DELETE /api/todo/{id}	Excluir um item	Nenhum	Nenhum

O diagrama a seguir mostra o design do aplicativo.

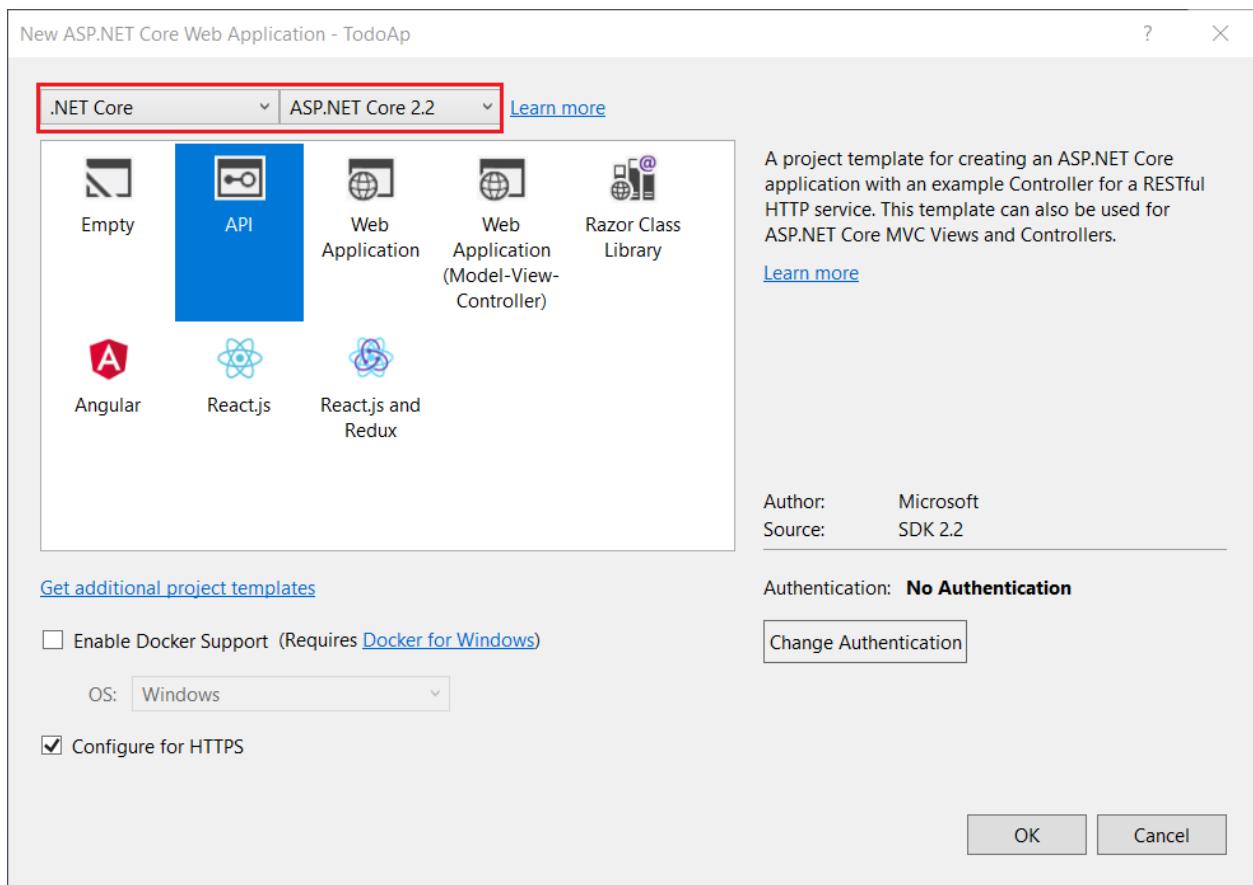


## Pré-requisitos

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- [Visual Studio 2017 versão 15.9 ou posterior](#) com a carga de trabalho **ASP.NET e desenvolvimento para a Web**
- [SDK 2.2 ou posterior do .NET Core](#)

## Criar um projeto Web

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- No menu **Arquivo**, selecione **Novo > Projeto**.
- Selecione o modelo **Aplicativo Web ASP.NET Core**. Nomeie o projeto como *TodoApi* e clique em **OK**.
- Na caixa de diálogo **Novo aplicativo Web ASP.NET Core – TodoApi** e escolha a versão do ASP.NET Core. Selecione o modelo **API** e clique em **OK**. **Não** selecione **Habilitar Suporte ao Docker**.



## Testar a API

O modelo de projeto cria uma API `values`. Chame o método `Get` em um navegador para testar o aplicativo.

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)

Pressione CTRL+F5 para executar o aplicativo. O Visual Studio inicia um navegador e navega para `https://localhost:<port>/api/values`, em que `<port>` é um número de porta escolhido aleatoriamente.

Se você receber uma caixa de diálogo perguntando se você deve confiar no certificado do IIS Express, selecione **Sim**. Na caixa de diálogo **Aviso de Segurança** exibida em seguida, selecione **Sim**.

O seguinte JSON é retornado:

```
["value1","value2"]
```

## Adicionar uma classe de modelo

Um *modelo* é um conjunto de classes que representam os dados gerenciados pelo aplicativo. O modelo para esse aplicativo é uma única classe `TodoItem`.

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- No **Gerenciador de Soluções**, clique com o botão direito do mouse no projeto. Selecione **Adicionar > Nova Pasta**. Nomeie a pasta *Models*.
- Clique com o botão direito do mouse na pasta *Modelos* e selecione **Adicionar > Classe**. Dê à classe o nome `TodoItem` e selecione **Adicionar**.

- Substitua o código do modelo pelo seguinte código:

```
namespace TodoApi.Models
{
    public class TodoItem
    {
        public long Id { get; set; }
        public string Name { get; set; }
        public bool IsComplete { get; set; }
    }
}
```

A propriedade `Id` funciona como a chave exclusiva em um banco de dados relacional.

As classes de modelo podem ser colocadas em qualquer lugar no projeto, mas a pasta `Models` é usada por convenção.

## Adicionar um contexto de banco de dados

O *contexto de banco de dados* é a classe principal que coordena a funcionalidade do Entity Framework para um modelo de dados. Essa classe é criada derivando-a da classe `Microsoft.EntityFrameworkCore.DbContext`.

- [Visual Studio](#)
  - [Visual Studio Code/Visual Studio para Mac](#)
- 
- Clique com o botão direito do mouse na pasta `Modelos` e selecione **Adicionar > Classe**. Nomeie a classe como `TodoContext` e clique em **Adicionar**.
  - Substitua o código do modelo pelo seguinte código:

```
using Microsoft.EntityFrameworkCore;

namespace TodoApi.Models
{
    public class TodoContext : DbContext
    {
        public TodoContext(DbContextOptions<TodoContext> options)
            : base(options)
        {
        }

        public DbSet<TodoItem> TodoItems { get; set; }
    }
}
```

## Registrar o contexto de banco de dados

No ASP.NET Core, serviços como o contexto de BD precisam ser registrados no contêiner de [DI \(injeção de dependência\)](#). O contêiner fornece o serviço aos controladores.

Atualize `Startup.cs` com o seguinte código realçado:

```

// Unused usings removed
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using TodoApi.Models;

namespace TodoApi
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the
        // container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<TodoContext>(opt =>
                opt.UseInMemoryDatabase("TodoList"));
            services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
        }

        // This method gets called by the runtime. Use this method to configure the HTTP
        // request pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                // The default HSTS value is 30 days. You may want to change this for
                // production scenarios, see https://aka.ms/aspnetcore-hsts.
                app.UseHsts();
            }

            app.UseHttpsRedirection();
            app.UseMvc();
        }
    }
}

```

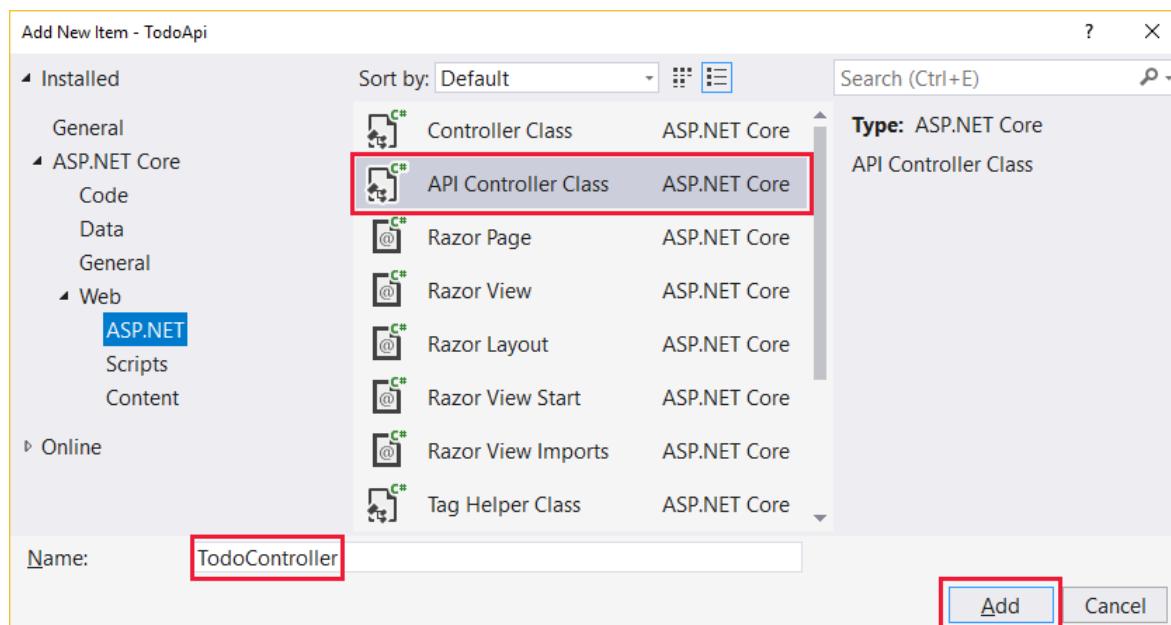
O código anterior:

- Remove as declarações `using` não utilizadas.
- Adiciona o contexto de banco de dados ao contêiner de DI.
- Especifica que o contexto de banco de dados usará um banco de dados em memória.

## Adicionar um controlador

- [Visual Studio](#)
- [Visual Studio Code/Visual Studio para Mac](#)
- Clique com o botão direito do mouse na pasta *Controllers*.
- Selecione **Adicionar > Novo Item**.

- Na caixa de diálogo **Adicionar Novo Item**, selecione o modelo **Classe do Controlador de API**.
- Dê à classe o nome *TodoController* e selecione **Adicionar**.



- Substitua o código do modelo pelo seguinte código:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using TodoApi.Models;

namespace TodoApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class TodoController : ControllerBase
    {
        private readonly TodoContext _context;

        public TodoController(TodoContext context)
        {
            _context = context;

            if (_context.TodoItems.Count() == 0)
            {
                // Create a new TodoItem if collection is empty,
                // which means you can't delete all TodoItems.
                _context.TodoItems.Add(new TodoItem { Name = "Item1" });
                _context.SaveChanges();
            }
        }
    }
}
```

O código anterior:

- Define uma classe de controlador de API sem métodos.
- Decore a classe com o atributo `[ApiController]`. Esse atributo indica se o controlador responde às solicitações da API Web. Para obter informações sobre comportamentos específicos habilitados pelo atributo, confira [Anotação com o atributo ApiController](#).
- Usa a DI para injetar o contexto de banco de dados (`TodoContext`) no controlador. O contexto de banco de

dados é usado em cada um dos métodos **CRUD** no controlador.

- Adiciona um item chamado `Item1` ao banco de dados se o banco de dados está vazio. Esse código está no construtor, de modo que ele seja executado sempre que há uma nova solicitação HTTP. Se você excluir todos os itens, o construtor criará `Item1` novamente na próxima vez que um método de API for chamado. Portanto, pode parecer que a exclusão não funcionou quando ela realmente funcionou.

## Adicionar métodos Get

Para fornecer uma API que recupera itens pendentes, adicione os seguintes métodos à classe `TodoController`:

```
// GET: api/Todo
[HttpGet]
public async Task<ActionResult<IEnumerable<TodoItem>>> GetTodoItems()
{
    return await _context.TodoItems.ToListAsync();
}

// GET: api/Todo/5
[HttpGet("{id}")]
public async Task<ActionResult<TodoItem>> GetTodoItem(long id)
{
    var todoItem = await _context.TodoItems.FindAsync(id);

    if (todoItem == null)
    {
        return NotFound();
    }

    return todoItem;
}
```

Esses métodos implementam dois pontos de extremidade GET:

- `GET /api/todo`
- `GET /api/todo/{id}`

Teste o aplicativo chamando os dois pontos de extremidade em um navegador. Por exemplo:

- `https://localhost:<port>/api/todo`
- `https://localhost:<port>/api/todo/1`

A seguinte resposta HTTP é produzida pela chamada a `GetTodoItems`:

```
[
{
    "id": 1,
    "name": "Item1",
    "isComplete": false
}]
```

## Roteamento e caminhos de URL

O atributo `[HttpGet]` indica um método que responde a uma solicitação HTTP GET. O caminho da URL de cada método é construído da seguinte maneira:

- Comece com a cadeia de caracteres de modelo no atributo `Route` do controlador:

```
namespace TodoApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class TodoController : ControllerBase
    {
        private readonly TodoContext _context;
```

- Substitua `[controller]` pelo nome do controlador, que é o nome da classe do controlador menos o sufixo "Controlador" por convenção. Para esta amostra, o nome da classe do controlador é `TodoController` e, portanto, o nome do controlador é "todo". O [roteamento](#) do ASP.NET Core não diferencia maiúsculas de minúsculas.
- Se o atributo `[HttpGet]` tiver um modelo de rota (por exemplo, `[HttpGet("products")]`), acrescente isso ao caminho. Esta amostra não usa um modelo. Para obter mais informações, confira [Roteamento de atributo com atributos `Http\[Verb\]`](#).

No método `GetTodoItem` a seguir, `"{id}"` é uma variável de espaço reservado para o identificador exclusivo do item pendente. Quando `GetTodoItem` é invocado, o valor de `"{id}"` na URL é fornecido para o método no parâmetro `id`.

```
// GET: api/Todo/5
[HttpGet("{id}")]
public async Task<ActionResult<TodoItem>> GetTodoItem(long id)
{
    var todoItem = await _context.TodoItems.FindAsync(id);

    if (todoItem == null)
    {
        return NotFound();
    }

    return todoItem;
}
```

## Valores de retorno

O tipo de retorno dos métodos `GetTodoItems` e `GetTodoItem` é o [tipo `ActionResult<T>`](#). O ASP.NET Core serializa automaticamente o objeto em [JSON](#) e grava o JSON no corpo da mensagem de resposta. O código de resposta para esse tipo de retorno é 200, supondo que não haja nenhuma exceção sem tratamento. As exceções sem tratamento são convertidas em erros 5xx.

Os tipos de retorno `ActionResult` podem representar uma ampla variedade de códigos de status HTTP. Por exemplo, `GetTodoItem` pode retornar dois valores de status diferentes:

- Se nenhum item corresponder à ID solicitada, o método retornará um código de erro 404 [NotFound](#).
- Caso contrário, o método retornará 200 com um corpo de resposta JSON. Retornar `item` resulta em uma resposta HTTP 200.

## Testar o método `GetTodosItems`

Este tutorial usa o Postman para testar a API Web.

- Instale o [Postman](#)
- Inicie o aplicativo Web.

- Inicie o Postman.
- Desabilite a **Verificação do certificado SSL**
  - Em **Arquivo > Configurações** (guia \*Geral), desabilite **Verificação do certificado SSL**.

**WARNING**

Habilite novamente a verificação do certificado SSL depois de testar o controlador.

- Crie uma solicitação.
  - Defina o método HTTP como **GET**.
  - Defina a URL de solicitação como `https://localhost:<port>/api/todo`. Por exemplo, `https://localhost:5001/api/todo`.
- Defina **Exibição de dois painéis** no Postman.
- Selecione **Enviar**.

The screenshot shows the Postman application window. At the top, there's a toolbar with 'File', 'Edit', 'View', 'Help', and various icons. Below the toolbar, the main interface has a header with 'My Workspace' and a search bar. A sidebar on the left lists 'GetAll' and 'GetAll' under 'Examples (0)'. The main workspace shows a 'GET GetAll' request. The 'Body' tab of the request details panel is selected, showing the URL `https://localhost:5001/api/todo`. The response panel shows a status of '200 OK' with a response time of '299 ms' and a size of '192 B'. The response body is displayed in a JSON editor with the following content:

```

1 [ {
2   "id": 1,
3   "name": "Item1",
4   "isComplete": false
5 }
6 ]
7
  
```

## Adicionar um método Create

Adicione o seguinte método `PostTodoItem`:

```
// POST: api/Todo
[HttpPost]
public async Task<ActionResult<TodoItem>> PostTodoItem(TodoItem item)
{
    _context.TodoItems.Add(item);
    await _context.SaveChangesAsync();

    return CreatedAtAction(nameof(GetTodoItem), new { id = item.Id }, item);
}
```

O código anterior é um método HTTP POST, conforme indicado pelo atributo `[HttpPost]`. O método obtém o valor do item pendente no corpo da solicitação HTTP.

O método `CreatedAtAction` :

- retorna um código de status HTTP 201 em caso de êxito. HTTP 201 é a resposta padrão para um método HTTP POST que cria um novo recurso no servidor.
- Adiciona um cabeçalho `Location` à resposta. O cabeçalho `Location` especifica o URI do item de tarefas pendentes recém-criado. Para obter mais informações, confira [10.2.2 201 Criado](#).
- Faz referência à ação `GetTodoItem` para criar o URI de `Location` do cabeçalho. A palavra-chave `nameof` do C# é usada para evitar o hard-coding do nome da ação, na chamada `CreatedAtAction`.

```
// GET: api/Todo/5
[HttpGet("{id}")]
public async Task<ActionResult<TodoItem>> GetTodoItem(long id)
{
    var todoItem = await _context.TodoItems.FindAsync(id);

    if (todoItem == null)
    {
        return NotFound();
    }

    return todoItem;
}
```

## Testar o método `PostTodoItem`

- Compile o projeto.
- No Postman, defina o método HTTP como `POST`.
- Selecione a guia **Corpo**.
- Selecione o botão de opção **bruto**.
- Defina o tipo como **JSON (aplicativo/json)**.
- No corpo da solicitação, insira JSON para um item pendente:

```
{
    "name": "walk dog",
    "isComplete": true
}
```

- Selecione **Enviar**.

The screenshot shows the Postman application interface. A POST request is being made to `https://localhost:5001/api/todo`. The request body is set to `JSON (application/json)` and contains the following JSON:

```
1 {  
2   "name": "Walk dog",  
3   "isComplete": true  
4 }
```

The response status is `201 Created`, and the response body is identical to the request body:

```
1 {  
2   "id": 2,  
3   "name": "Walk dog",  
4   "isComplete": true  
5 }
```

Se você receber um erro 405 Método Não Permitido, provavelmente, esse será o resultado da não compilação do projeto após a adição do método `PostTodoItem`.

### Testar o URI do cabeçalho de local

- Selecione a guia **Cabeçalhos** no painel **Resposta**.
- Copie o valor do cabeçalho **Local**:

The screenshot shows the Postman application interface with the **Headers** tab selected in the **Resposta** panel. The **Location** header is highlighted with a red box:

Date → Tue, 06 Nov 2018 01:11:47 GMT  
Content-Type → application/json; charset=utf-8  
Server → Kestrel  
Transfer-Encoding → chunked  
Location → `https://localhost:5001/api/Todo/2`

- Defina o método como GET.
- Cole o URI (por exemplo, `https://localhost:5001/api/Todo/2`)
- Selecione **Enviar**.

## Adicionar um método PutTodoItem

Adicione o seguinte método `PutTodoItem`:

```
// PUT: api/Todo/5
[HttpPut("{id}")]
public async Task<IActionResult> PutTodoItem(long id, TodoItem item)
{
    if (id != item.Id)
    {
        return BadRequest();
    }

    _context.Entry(item).State = EntityState.Modified;
    await _context.SaveChangesAsync();

    return NoContent();
}
```

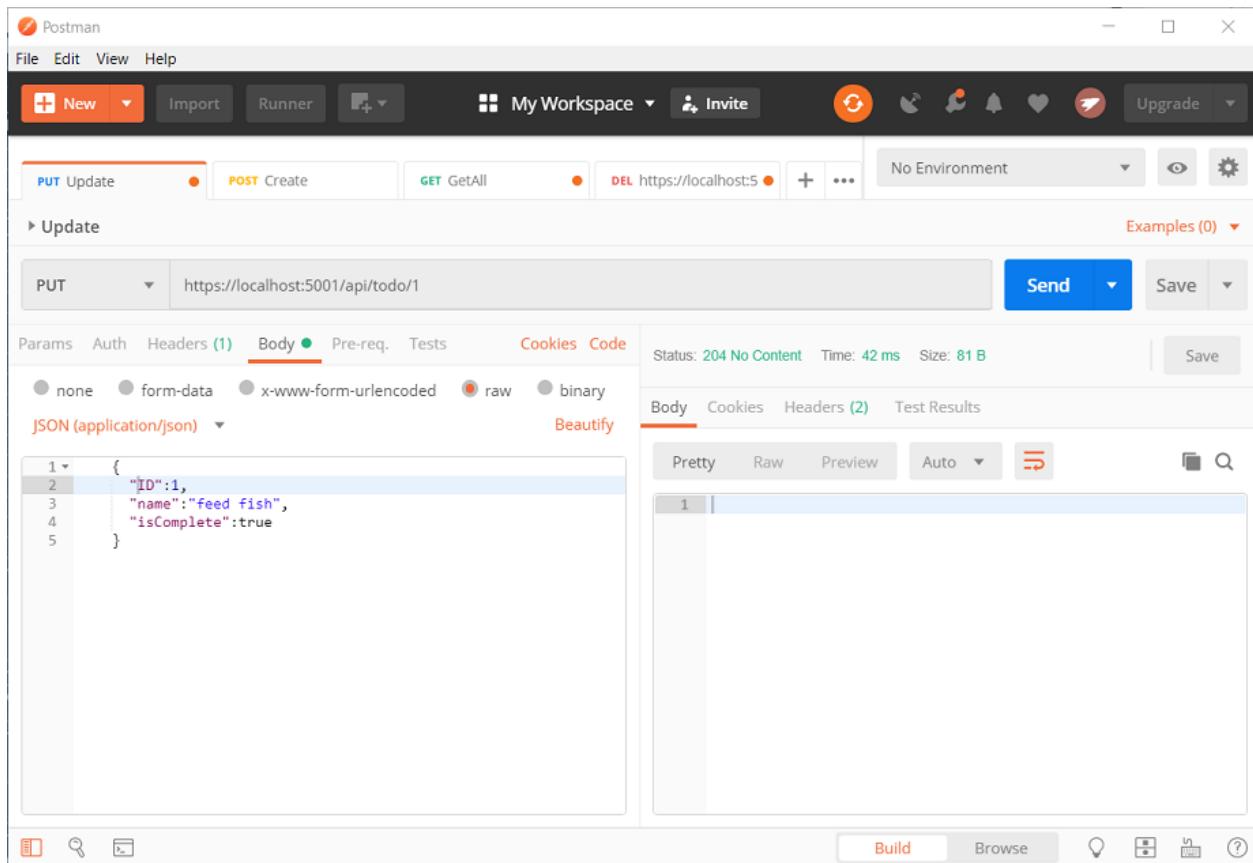
`PutTodoItem` é semelhante a `PostTodoItem`, exceto pelo uso de HTTP PUT. A resposta é [204 \(Sem conteúdo\)](#). De acordo com a especificação de HTTP, uma solicitação PUT exige que o cliente envie a entidade inteira atualizada, não apenas as alterações. Para dar suporte a atualizações parciais, use [HTTP PATCH](#).

### Testar o método PutTodoItem

Atualize o item pendente que tem a ID = 1 e defina seu nome como "feed fish":

```
{
    "ID":1,
    "name":"feed fish",
    "isComplete":true
}
```

A seguinte imagem mostra a atualização do Postman:



## Adicionar um método DeleteTodoItem

Adicione o seguinte método `DeleteTodoItem` :

```
// DELETE: api/Todo/5
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteTodoItem(long id)
{
    var todoItem = await _context.TodoItems.FindAsync(id);

    if (todoItem == null)
    {
        return NotFound();
    }

    _context.TodoItems.Remove(todoItem);
    await _context.SaveChangesAsync();

    return NoContent();
}
```

A resposta `DeleteTodoItem` é **204 (Sem conteúdo)**.

### Testar o método DeleteTodoItem

Use o Postman para excluir um item pendente:

- Defina o método como `DELETE`.
- Defina o URI do objeto a ser excluído, por exemplo, `https://localhost:5001/api/todo/1`
- Selecione **Enviar**

O aplicativo de exemplo permite que você exclua todos os itens, mas quando o último item é excluído, um novo é criado pelo construtor de classe de modelo na próxima vez que a API é chamada.

## Chamar a API com o jQuery

Nesta seção, uma página HTML que usa o jQuery para chamar a API Web é adicionada. O jQuery inicia a solicitação e atualiza a página com os detalhes da resposta da API.

Configure o aplicativo para [fornecer arquivos estáticos](#) e [habilitar o mapeamento de arquivo padrão](#):

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        // The default HSTS value is 30 days. You may want to change this for
        // production scenarios, see https://aka.ms/aspnetcore-hsts.
        app.UseHsts();
    }

    app.UseDefaultFiles();
    app.UseStaticFiles();
    app.UseHttpsRedirection();
    app.UseMvc();
}
```

Crie uma pasta `wwwroot` no diretório do projeto.

Adicione um arquivo HTML chamado `index.html` ao diretório `wwwroot`. Substitua seu conteúdo pela seguinte marcação:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>To-do CRUD</title>
    <style>
        input[type='submit'], button, [aria-label] {
            cursor: pointer;
        }

        #spoiler {
            display: none;
        }

        table {
            font-family: Arial, sans-serif;
            border: 1px solid;
            border-collapse: collapse;
        }

        th {
            background-color: #0066CC;
            color: white;
        }

        td {
            border: 1px solid;
            padding: 5px;
        }
    </style>
</head>
<body>
    <h1>To-do CRUD</h1>
    <h3>Add</h3>
```

```

<form action="javascript:void(0);" method="POST" onsubmit="addItem()">
    <input type="text" id="add-name" placeholder="New to-do">
    <input type="submit" value="Add">
</form>

<div id="spoiler">
    <h3>Edit</h3>
    <form class="my-form">
        <input type="hidden" id="edit-id">
        <input type="checkbox" id="edit-isComplete">
        <input type="text" id="edit-name">
        <input type="submit" value="Save">
        <a onclick="closeInput()" aria-label="Close">&#10006;</a>
    </form>
</div>

<p id="counter"></p>

<table>
    <tr>
        <th>Is Complete</th>
        <th>Name</th>
        <th></th>
        <th></th>
    </tr>
    <tbody id="todos"></tbody>
</table>

<script src="https://code.jquery.com/jquery-3.3.1.min.js"
        integrity="sha256-FgpCb/KJQ1LNf0u91ta32o/NMZxltwRo8QtmkMRdAu8="
        crossorigin="anonymous"></script>
<script src="site.js"></script>
</body>
</html>

```

Adicione um arquivo JavaScript chamado *site.js* ao diretório *wwwroot*. Substitua seu conteúdo pelo código a seguir:

```

const uri = "api/todo";
let todos = null;
function getCount(data) {
    const el = $("#counter");
    let name = "to-do";
    if (data) {
        if (data > 1) {
            name = "to-dos";
        }
        el.text(data + " " + name);
    } else {
        el.text("No " + name);
    }
}

$(document).ready(function() {
    getData();
});

function getData() {
    $.ajax({
        type: "GET",
        url: uri,
        cache: false,
        success: function(data) {
            const tBody = $("#todos");
            $(tBody).empty();
            getCount(data);
            for (let todo of data) {
                const tr = $(`<tr><td>${todo.isComplete}</td><td>${todo.name}</td><td></td><td></td></tr>`);
                tBody.append(tr);
            }
        }
    });
}

```

```

        getCount(data.length);

        $.each(data, function(key, item) {
            const tr = $("<tr></tr>")
                .append(
                    $("<td></td>").append(
                        $("<input/>", {
                            type: "checkbox",
                            disabled: true,
                            checked: item.isComplete
                        })
                    )
                )
                .append($("<td></td>").text(item.name))
                .append(
                    $("<td></td>").append(
                        "<button>Edit</button>").on("click", function() {
                            editItem(item.id);
                        })
                )
            )
            .append(
                $("<td></td>").append(
                    "<button>Delete</button>").on("click", function() {
                        deleteItem(item.id);
                    })
            )
        );
    });

    tr.appendTo(tBody);
});

todos = data;
}
});
}

function addItem() {
    const item = {
        name: $("#add-name").val(),
        isComplete: false
    };

    $.ajax({
        type: "POST",
        accepts: "application/json",
        url: uri,
        contentType: "application/json",
        data: JSON.stringify(item),
        error: function(jqXHR, textStatus, errorThrown) {
            alert("Something went wrong!");
        },
        success: function(result) {
            getData();
            $("#add-name").val("");
        }
    });
}

function deleteItem(id) {
    $.ajax({
        url: uri + "/" + id,
        type: "DELETE",
        success: function(result) {
            getData();
        }
    });
}

```

```

function editItem(id) {
    $.each(todos, function(key, item) {
        if (item.id === id) {
            $("#edit-name").val(item.name);
            $("#edit-id").val(item.id);
            $("#edit-isComplete")[0].checked = item.isComplete;
        }
    });
    $("#spoiler").css({ display: "block" });
}

$(".my-form").on("submit", function() {
    const item = {
        name: $("#edit-name").val(),
        isComplete: $("#edit-isComplete").is(":checked"),
        id: $("#edit-id").val()
    };

    $.ajax({
        url: uri + "/" + $("#edit-id").val(),
        type: "PUT",
        accepts: "application/json",
        contentType: "application/json",
        data: JSON.stringify(item),
        success: function(result) {
            getData();
        }
    });
});

closeInput();
return false;
});

function closeInput() {
    $("#spoiler").css({ display: "none" });
}

```

Uma alteração nas configurações de inicialização do projeto ASP.NET Core pode ser necessária para testar a página HTML localmente:

- Abra `Properties\launchSettings.json`.
- Remova a propriedade `launchUrl` para forçar o aplicativo a ser aberto em `index.html`, o arquivo padrão do projeto.

Há várias maneiras de obter o jQuery. No snippet anterior, a biblioteca é carregada de uma CDN.

Esta amostra chama todos os métodos CRUD da API. Veja a seguir explicações das chamadas à API.

### Obter uma lista de itens pendentes

A função `ajax` do jQuery envia uma solicitação `GET` para a API, que retorna o JSON que representa uma matriz de itens pendentes. A função de retorno de chamada `success` será invocada se a solicitação for bem-sucedida. No retorno de chamada, o DOM é atualizado com as informações do item pendente.

```

$(document).ready(function() {
    getData();
});

function getData() {
    $.ajax({
        type: "GET",
        url: uri,
        cache: false,
        success: function(data) {
            const tBody = $("#todos");

            $(tBody).empty();

            getCount(data.length);

            $.each(data, function(key, item) {
                const tr = $("<tr></tr>")
                    .append(
                        $("<td></td>").append(
                            $("<input/>", {
                                type: "checkbox",
                                disabled: true,
                                checked: item.isComplete
                            })
                        )
                    )
                .append($("<td></td>").text(item.name))
                .append(
                    $("<td></td>").append(
                        $("<button>Edit</button>").on("click", function() {
                            editItem(item.id);
                        })
                    )
                )
                .append(
                    $("<td></td>").append(
                        $("<button>Delete</button>").on("click", function() {
                            deleteItem(item.id);
                        })
                    )
                );
            });

            tr.appendTo(tBody);
        });
    });

    todos = data;
}
});
}

```

## Adicionar um item pendente

A função `ajax` envia uma solicitação `POST` com o item pendente no corpo da solicitação. As opções `accepts` e `contentType` são definidas como `application/json` para especificar o tipo de mídia que está sendo recebido e enviado. O item pendente é convertido em JSON usando `JSON.stringify`. Quando a API retorna um código de status de êxito, a função `getData` é invocada para atualizar a tabela HTML.

```

function addItem() {
    const item = {
        name: $("#add-name").val(),
        isComplete: false
    };

    $.ajax({
        type: "POST",
        accepts: "application/json",
        url: uri,
        contentType: "application/json",
        data: JSON.stringify(item),
        error: function(jqXHR, textStatus, errorThrown) {
            alert("Something went wrong!");
        },
        success: function(result) {
            getData();
            $("#add-name").val("");
        }
    });
}

```

## Atualizar um item pendente

A atualização de um item pendente é semelhante à adição de um. A `url` é alterada para adicionar o identificador exclusivo do item, e o `type` é `PUT`.

```

$.ajax({
    url: uri + "/" + $("#edit-id").val(),
    type: "PUT",
    accepts: "application/json",
    contentType: "application/json",
    data: JSON.stringify(item),
    success: function(result) {
        getData();
    }
});

```

## Excluir um item pendente

A exclusão de um item pendente é feita definindo o `type` na chamada do AJAX como `DELETE` e especificando o identificador exclusivo do item na URL.

```

$.ajax({
    url: uri + "/" + id,
    type: "DELETE",
    success: function(result) {
        getData();
    }
});

```

## Recursos adicionais

[Exibir ou baixar o código de exemplo para este tutorial](#). Consulte [como baixar](#).

Para obter mais informações, consulte os seguintes recursos:

- [Criar APIs Web com o ASP.NET Core](#)
- [Páginas de ajuda da API Web ASP.NET Core com o Swagger/OpenAPI](#)
- [Páginas Razor do ASP.NET Core com EF Core – série de tutoriais](#)

- Roteamento para ações do controlador no ASP.NET Core
- Tipos de retorno de ação do controlador na API Web ASP.NET Core
- Implantar aplicativos ASP.NET Core no Serviço de Aplicativo do Azure
- Hospedar e implantar o ASP.NET Core

## Próximas etapas

Neste tutorial, você aprendeu como:

- Criar um projeto de API Web.
- Adicionar uma classe de modelo.
- Criar o contexto de banco de dados.
- Registrar o contexto de banco de dados.
- Adicionar um controlador.
- Adicionar métodos CRUD.
- Configurar o roteamento e caminhos de URL.
- Especificar os valores retornados.
- Chamar a API Web com o Postman.
- Chamar a API Web com o jQuery.

Avance para o próximo tutorial para saber como gerar páginas de ajuda da API:

[Introdução ao Swashbuckle e ao ASP.NET Core](#)

# Criar uma API Web com o ASP.NET Core e o MongoDB

06/02/2019 • 13 minutes to read • [Edit Online](#)

Por [Pratik Khandelwal](#) e [Scott Addie](#)

Este tutorial cria uma API Web que executa as operações CRUD (criar, ler, atualizar e excluir) em um banco de dados NoSQL do [MongoDB](#).

Neste tutorial, você aprenderá como:

- Configurar o MongoDB
- Criar um banco de dados do MongoDB
- Definir uma coleção e um esquema do MongoDB
- Executar operações CRUD do MongoDB a partir de uma API Web

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Pré-requisitos

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- [SDK 2.2 ou posterior do .NET Core](#)
- [Visual Studio 2017 versão 15.9 ou posterior com a carga de trabalho ASP.NET e desenvolvimento para a Web](#)
- [MongoDB](#)

## Configurar o MongoDB

Se usar o Windows, o MongoDB será instalado em `C:\Arquivos de Programas\MongoDB` por padrão. Adicione `C:\Arquivos de Programas\MongoDB\Servidor\<número_de_versão>\bin` à variável de ambiente `Path`. Essa alteração possibilita o acesso ao MongoDB a partir de qualquer lugar em seu computador de desenvolvimento.

Use o Shell do mongo nas etapas a seguir para criar um banco de dados, fazer coleções e armazenar documentos. Para saber mais sobre os comandos de Shell do mongo, consulte [Como trabalhar com o Shell do mongo](#).

1. Escolha um diretório no seu computador de desenvolvimento para armazenar os dados. Por exemplo, `C:\BooksData` no Windows. Crie o diretório se não houver um. O Shell do mongo não cria novos diretórios.
2. Abra um shell de comando. Execute o comando a seguir para se conectar ao MongoDB na porta padrão 27017. Lembre-se de substituir `<data_directory_path>` pelo diretório escolhido na etapa anterior.

```
mongod --dbpath <data_directory_path>
```

3. Abra outra instância do shell de comando. Conecte-se ao banco de dados de testes padrão executando o seguinte comando:

```
mongo
```

4. Execute o seguinte em um shell de comando:

```
use BookstoreDb
```

Se ele ainda não existir, um banco de dados chamado *BookstoreDb* será criado. Se o banco de dados existir, a conexão dele será aberta para transações.

5. Crie uma coleção `Books` usando o seguinte comando:

```
db.createCollection('Books')
```

O seguinte resultado é exibido:

```
{ "ok" : 1 }
```

6. Defina um esquema para a coleção `Books` e insira dois documentos usando o seguinte comando:

```
db.Books.insertMany([{"Name":'Design Patterns','Price':54.93,'Category':'Computers','Author':'Ralph Johnson'}, {"Name":'Clean Code','Price':43.15,'Category':'Computers','Author':'Robert C. Martin'}])
```

O seguinte resultado é exibido:

```
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("5bfd996f7b8e48dc15ff215d"),
    ObjectId("5bfd996f7b8e48dc15ff215e")
  ]
}
```

7. Visualize os documentos no banco de dados usando o seguinte comando:

```
db.Books.find({}).pretty()
```

O seguinte resultado é exibido:

```
{
  "_id" : ObjectId("5bfd996f7b8e48dc15ff215d"),
  "Name" : "Design Patterns",
  "Price" : 54.93,
  "Category" : "Computers",
  "Author" : "Ralph Johnson"
}
{
  "_id" : ObjectId("5bfd996f7b8e48dc15ff215e"),
  "Name" : "Clean Code",
  "Price" : 43.15,
  "Category" : "Computers",
  "Author" : "Robert C. Martin"
}
```

O esquema adiciona uma propriedade `_id` gerada automaticamente do tipo `ObjectId` para cada documento.

O banco de dados está pronto. Você pode começar a criar a API Web do ASP.NET Core.

## Criar o projeto da API Web do ASP.NET Core

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)

1. Acesse **Arquivo > Novo > Projeto**.
2. Selecione **Aplicativo Web do ASP.NET Core**, nomeie o projeto como *BooksApi* e clique em **OK**.
3. Selecione a estrutura de destino **.NET Core** e **ASP.NET Core 2.1**. Selecione o modelo de projeto **API** e clique em **OK**:
4. Visite a [Galeria do NuGet: MongoDB.Driver](#) para determinar a versão estável mais recente do driver .NET para MongoDB. Na janela **Console do Gerenciador de Pacotes**, navegue até a raiz do projeto. Execute o seguinte comando para instalar o driver .NET para MongoDB:

```
Install-Package MongoDB.Driver -Version {VERSION}
```

## Adicionar um modelo

1. Adicione um diretório *Modelos* à raiz do projeto.
2. Adicione uma classe `Book` ao diretório *Modelos* com o seguinte código:

```
using MongoDB.Bson;
using MongoDB.Bson.Serialization.Attributes;

namespace BooksApi.Models
{
    public class Book
    {
        [BsonId]
        [BsonRepresentation(BsonType.ObjectId)]
        public string Id { get; set; }

        [BsonElement("Name")]
        public string BookName { get; set; }

        [BsonElement("Price")]
        public decimal Price { get; set; }

        [BsonElement("Category")]
        public string Category { get; set; }

        [BsonElement("Author")]
        public string Author { get; set; }
    }
}
```

Na classe anterior, a propriedade `Id`:

- É necessária para mapear o objeto CLR (Common Language Runtime) para a coleção do MongoDB.
- É anotada com `[BsonId]` para designar essa propriedade como a chave primária do documento.

- É anotada com `[BsonRepresentation(BsonType.ObjectId)]` para permitir a passagem do parâmetro como tipo `string`, em vez de `ObjectId`. O Mongo processa a conversão de `string` para `ObjectId`.

Outras propriedades na classe são anotadas com o atributo `[BsonElement]`. O valor do atributo representa o nome da propriedade da coleção do MongoDB.

## Adicionar uma classe de operações CRUD

1. Adicione um diretório `Serviços` à raiz do projeto.
2. Adicione uma classe `BookService` ao diretório `Serviços` com o seguinte código:

```
using System.Collections.Generic;
using System.Linq;
using BooksApi.Models;
using Microsoft.Extensions.Configuration;
using MongoDB.Driver;

namespace BooksApi.Services
{
    public class BookService
    {
        private readonly IMongoCollection<Book> _books;

        public BookService(IConfiguration config)
        {
            var client = new MongoClient(config.GetConnectionString("BookstoreDb"));
            var database = client.GetDatabase("BookstoreDb");
            _books = database.GetCollection<Book>("Books");
        }

        public List<Book> Get()
        {
            return _books.Find(book => true).ToList();
        }

        public Book Get(string id)
        {
            return _books.Find<Book>(book => book.Id == id).FirstOrDefault();
        }

        public Book Create(Book book)
        {
            _books.InsertOne(book);
            return book;
        }

        public void Update(string id, Book bookIn)
        {
            _books.ReplaceOne(book => book.Id == id, bookIn);
        }

        public void Remove(Book bookIn)
        {
            _books.DeleteOne(book => book.Id == bookIn.Id);
        }

        public void Remove(string id)
        {
            _books.DeleteOne(book => book.Id == id);
        }
    }
}
```

3. Adicione uma cadeia de conexão do MongoDB a `appsettings.json`:

```
{  
    "ConnectionStrings": {  
        "BookstoreDb": "mongodb://localhost:27017"  
    },  
    "Logging": {  
        "IncludeScopes": false,  
        "Debug": {  
            "LogLevel": {  
                "Default": "Warning"  
            }  
        },  
        "Console": {  
            "LogLevel": {  
                "Default": "Warning"  
            }  
        }  
    }  
}
```

A propriedade `BookstoreDb` anterior é acessada no construtor da classe `BookService`.

4. No `Startup.ConfigureServices`, registre a classe `BookService` com o sistema de Injeção de Dependência:

```
public void ConfigureServices(IServiceCollection services)  
{  
    services.AddScoped<BookService>();  
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);  
}
```

O registro de serviço anterior é necessário para dar suporte à injeção do construtor no consumo de classes.

A classe `BookService` usa os seguintes membros `MongoDB.Driver` para executar operações CRUD em relação ao banco de dados:

- `MongoClient` – Lê a instância do servidor para executar operações de banco de dados. O construtor dessa classe é fornecido na cadeia de conexão do MongoDB:

```
public BookService(IConfiguration config)  
{  
    var client = new MongoClient(config.GetConnectionString("BookstoreDb"));  
    var database = client.GetDatabase("BookstoreDb");  
    _books = database.GetCollection<Book>("Books");  
}
```

- `IMongoDatabase` – Representa o banco de dados Mongo para execução de operações. Este tutorial usa o método genérico `GetCollection<T>(collection)` na interface para obter acesso a dados em uma coleção específica. Operações CRUD podem ser executadas em relação à coleção depois que esse método é chamado. Na chamada de método `GetCollection<T>(collection)` :

- `collection` representa o nome da coleção.
- `T` representa o tipo de objeto CLR armazenado na coleção.

`GetCollection<T>(collection)` retorna um objeto `MongoCollection` que representa a coleção. Neste tutorial, os seguintes métodos são invocados na coleção:

- `Find<T>` – Retorna todos os documentos na coleção que correspondem aos critérios de pesquisa fornecidos.
- `InsertOne` – Insere o objeto fornecido como um novo documento na coleção.

- `ReplaceOne` – Substitui o documento único que corresponde aos critérios de pesquisa definidos com o objeto fornecido.
- `DeleteOne` – Exclui um documento único que corresponde aos critérios de pesquisa fornecidos.

## Adicionar um controlador

1. Adicione uma classe `BooksController` ao diretório `Controladores` com o seguinte código:

```
using System.Collections.Generic;
using BooksApi.Models;
using BooksApi.Services;
using Microsoft.AspNetCore.Mvc;

namespace BooksApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class BooksController : ControllerBase
    {
        private readonly BookService _bookService;

        public BooksController(BookService bookService)
        {
            _bookService = bookService;
        }

        [HttpGet]
        public ActionResult<List<Book>> Get()
        {
            return _bookService.Get();
        }

        [HttpGet("{id:length(24)}", Name = "GetBook")]
        public ActionResult<Book> Get(string id)
        {
            var book = _bookService.Get(id);

            if (book == null)
            {
                return NotFound();
            }

            return book;
        }

        [HttpPost]
        public ActionResult<Book> Create(Book book)
        {
            _bookService.Create(book);

            return CreatedAtRoute("GetBook", new { id = book.Id.ToString() }, book);
        }

        [HttpPut("{id:length(24)}")]
        public IActionResult Update(string id, Book bookIn)
        {
            var book = _bookService.Get(id);

            if (book == null)
            {
                return NotFound();
            }

            _bookService.Update(id, bookIn);

            return NoContent();
        }
    }
}
```

```

        }

        [HttpDelete("{id:length(24)}")]
        public IActionResult Delete(string id)
        {
            var book = _bookService.Get(id);

            if (book == null)
            {
                return NotFound();
            }

            _bookService.Remove(book.Id);

            return NoContent();
        }
    }
}

```

O controlador da API Web anterior:

- Usa a classe `BookService` para executar operações CRUD.
  - Contém métodos de ação para dar suporte a solicitações GET, POST, PUT e DELETE HTTP.
2. Compile e execute o aplicativo.
  3. Navegue até `http://localhost:<port>/api/books` no seu navegador. A seguinte resposta JSON é exibida:

```
[
  {
    "id": "5bfd996f7b8e48dc15ff215d",
    "bookName": "Design Patterns",
    "price": 54.93,
    "category": "Computers",
    "author": "Ralph Johnson"
  },
  {
    "id": "5bfd996f7b8e48dc15ff215e",
    "bookName": "Clean Code",
    "price": 43.15,
    "category": "Computers",
    "author": "Robert C. Martin"
  }
]
```

## Próximas etapas

Para saber mais sobre a criação de APIs Web do ASP.NET Core, confira os seguintes recursos:

- [Criar APIs Web com o ASP.NET Core](#)
- [Tipos de retorno de ação do controlador na API Web ASP.NET Core](#)

# Páginas de ajuda da API Web ASP.NET Core com o Swagger/OpenAPI

13/12/2018 • 4 minutes to read • [Edit Online](#)

Por [Christoph Nienaber](#) e [Rico Suter](#)

Ao consumir uma API Web, entender seus vários métodos pode ser um desafio para um desenvolvedor. O [Swagger](#), também conhecido como [OpenAPI](#), resolve o problema da geração de páginas de ajuda e de documentação úteis para APIs Web. Ele oferece benefícios, como documentação interativa, geração de SDK de cliente e capacidade de descoberta de API.

Neste artigo, são exibidas as implementações do .NET do Swagger [Swashbuckle.AspNetCore](#) e [NSwag](#):

- O **Swashbuckle.AspNetCore** é um projeto de software livre para geração de documentos do Swagger para APIs Web ASP.NET Core.
- O **NSwag** é outro projeto de software livre para geração de documentos do Swagger e a integração da [interface do usuário do Swagger](#) ou do [ReDoc](#) em APIs Web ASP.NET Core. Além disso, o NSwag oferece abordagens para gerar o código de cliente C# e TypeScript para sua API.

## O que é o Swagger / OpenAPI?

O Swagger é uma especificação independente de linguagem para descrever APIs [REST](#). O projeto de Swagger foi doado para a [Iniciativa OpenAPI](#), na qual agora é chamado de OpenAPI. Ambos os nomes são intercambiáveis, mas OpenAPI é o preferencial. Ele permite que computadores e pessoas entendam os recursos de um serviço sem nenhum acesso direto à implementação (código-fonte, acesso à rede, documentação). Uma meta é minimizar a quantidade de trabalho necessário para conectar-se a serviços desassociados. Outra meta é reduzir o período de tempo necessário para documentar um serviço com precisão.

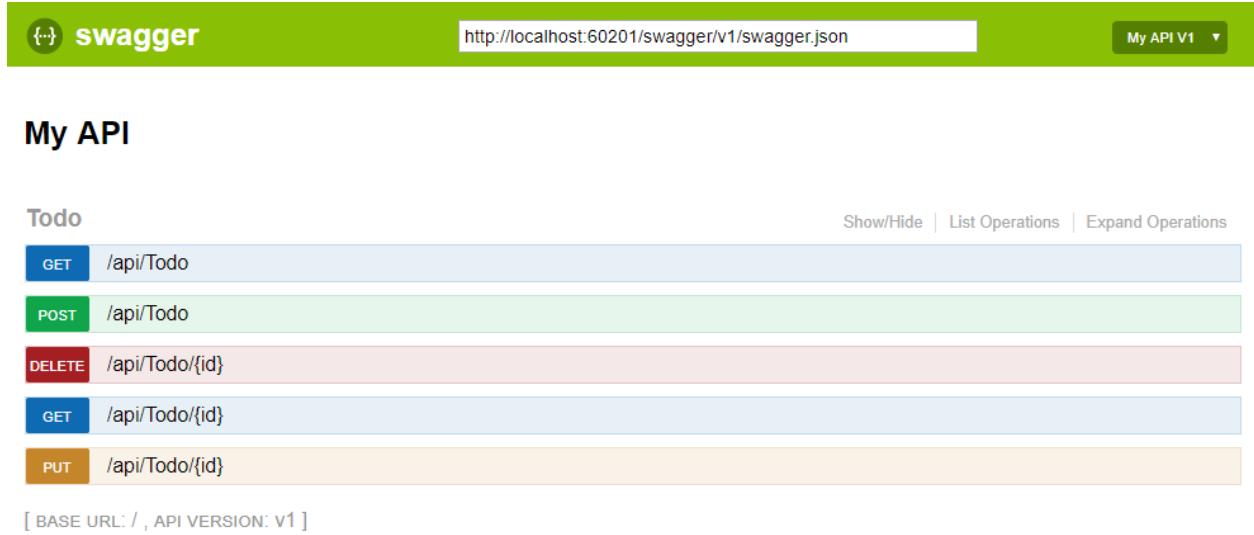
## Especificação do Swagger (swagger.json)

O ponto central para o fluxo do Swagger é a especificação do Swagger que é, por padrão, um documento chamado *swagger.json*. Ele é gerado pela cadeia de ferramentas do Swagger (ou por implementações de terceiros) com base no seu serviço. Ele descreve os recursos da API e como acessá-lo com HTTP. Ele gera a interface do usuário do Swagger e é usado pela cadeia de ferramentas para habilitar a geração e a descoberta de código de cliente. Aqui está um exemplo de uma especificação do Swagger, resumida:

```
{
    "swagger": "2.0",
    "info": {
        "version": "v1",
        "title": "API V1"
    },
    "basePath": "/",
    "paths": {
        "/api/Todo": {
            "get": {
                "tags": [
                    "Todo"
                ],
                "operationId": "ApiTodoGet",
                "consumes": [],
                "produces": [
                    "text/plain",
                    "application/json",
                    "text/json"
                ],
                "responses": {
                    "200": {
                        "description": "Success",
                        "schema": {
                            "type": "array",
                            "items": {
                                "$ref": "#/definitions/TodoItem"
                            }
                        }
                    }
                }
            },
            "post": {
                ...
            }
        },
        "/api/Todo/{id)": {
            "get": {
                ...
            },
            "put": {
                ...
            },
            "delete": {
                ...
            },
            "definitions": {
                "TodoItem": {
                    "type": "object",
                    "properties": {
                        "id": {
                            "format": "int64",
                            "type": "integer"
                        },
                        "name": {
                            "type": "string"
                        },
                        "isComplete": {
                            "default": false,
                            "type": "boolean"
                        }
                    }
                }
            }
        },
        "securityDefinitions": {}
    }
}
```

# Interface do usuário do Swagger

A [Interface do usuário do Swagger](#) oferece uma interface do usuário baseada na Web que conta com informações sobre o serviço, usando a especificação do Swagger gerada. O Swashbuckle e o NSwag incluem uma versão incorporada da interface do usuário do Swagger, para que ele possa ser hospedado em seu aplicativo ASP.NET Core usando uma chamada de registro de middleware. A interface do usuário da Web tem esta aparência:



The screenshot shows the Swagger UI interface for a 'Todo' API. At the top, there's a green header bar with the 'swagger' logo and a URL input field containing 'http://localhost:60201/swagger/v1/swagger.json'. On the right of the header is a dropdown menu set to 'My API V1'. Below the header, the title 'My API' is displayed in bold black font. Underneath the title, the section 'Todo' is shown. To the right of 'Todo' are three buttons: 'Show/Hide', 'List Operations', and 'Expand Operations'. The main area displays five API operations: 1. GET /api/Todo (blue background) 2. POST /api/Todo (green background) 3. DELETE /api/Todo/{id} (red background) 4. GET /api/Todo/{id} (light blue background) 5. PUT /api/Todo/{id} (orange background). Below these operations is a note '[ BASE URL: / , API VERSION: v1 ]'.

Todo método de ação pública nos controladores pode ser testado da interface do usuário. Clique em um nome de método para expandir a seção. Adicione os parâmetros necessários e, em seguida, clique em **Experimente!**.

GET /api/Todo

## Response Class (Status 200)

Success

[Model](#) [Example Value](#)

```
[  
  {  
    "id": 0,  
    "name": "string",  
    "isComplete": false  
  }  
]
```

Response Content Type [text/plain](#)[Try it out!](#) [Hide Response](#)

## Curl

```
curl -X GET --header 'Accept: application/json' 'http://localhost:60201/api/Todo'
```

## Request URL

<http://localhost:60201/api/Todo>

## Response Body

```
[  
  {  
    "id": 1,  
    "name": "Item1",  
    "isComplete": false  
  }  
]
```

## Response Code

200

## Response Headers

```
{  
  "date": "Thu, 31 Aug 2017 17:29:04 GMT",  
  "server": "Kestrel",  
  "transfer-encoding": "chunked",  
  "content-type": "application/json; charset=utf-8"  
}
```

## NOTE

A versão da interface do usuário do Swagger usada para as capturas de tela é a versão 2. Para obter um exemplo da versão 3, confira [Exemplo Petstore](#).

## Próximas etapas

- [Introdução ao Swashbuckle](#)
- [Introdução ao NSwag](#)

# Introdução ao Swashbuckle e ao ASP.NET Core

10/01/2019 • 19 minutes to read • [Edit Online](#)

Por [Shayne Boyer](#) e [Scott Addie](#)

[Exibir ou baixar código de exemplo \(como baixar\)](#)

Há três componentes principais do Swashbuckle:

- [Swashbuckle.AspNetCore.Swagger](#): um modelo de objeto e um middleware do Swagger para expor objetos `SwaggerDocument` como pontos de extremidade JSON.
- [Swashbuckle.AspNetCore.SwaggerGen](#): um gerador do Swagger cria objetos `SwaggerDocument` diretamente de modelos, controladores e rotas. Normalmente, ele é combinado com o middleware de ponto de extremidade do Swagger para expor automaticamente o JSON do Swagger.
- [Swashbuckle.AspNetCore.SwaggerUI](#): uma versão incorporada da ferramenta de interface do usuário do Swagger. Ele interpreta o JSON do Swagger para criar uma experiência avançada e personalizável para descrever a funcionalidade da API Web. Ela inclui o agente de teste interno para os métodos públicos.

## Instalação do pacote

O Swashbuckle pode ser adicionado com as seguintes abordagens:

- [Visual Studio](#)
- [Visual Studio para Mac](#)
- [Visual Studio Code](#)
- [CLI do .NET Core](#)
- Da janela **Console do Gerenciador de Pacotes**:
  - Acesse **Exibição > Outras Janelas > Console do Gerenciador de Pacotes**
  - Navegue para o diretório no qual o arquivo `TodoApi.csproj` está localizado
  - Execute o seguinte comando:

```
Install-Package Swashbuckle.AspNetCore
```

- Da caixa de diálogo **Gerenciar Pacotes NuGet**:
  - Clique com o botão direito do mouse no projeto em **Gerenciador de Soluções > Gerenciar Pacotes NuGet**
  - Defina a **Origem do pacote** para "nuget.org"
  - Insira "Swashbuckle.AspNetCore" na caixa de pesquisa
  - Selecione o pacote "Swashbuckle.AspNetCore" na guia **Procurar** e clique em **Instalar**

## Adicionar e configurar o middleware do Swagger

Adicione o gerador do Swagger à coleção de serviços no método `Startup.ConfigureServices` :

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<TodoContext>(opt =>
        opt.UseInMemoryDatabase("TodoList"));
    services.AddMvc();

    // Register the Swagger generator, defining 1 or more Swagger documents
    services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1", new Info { Title = "My API", Version = "v1" });
    });
}

```

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<TodoContext>(opt =>
        opt.UseInMemoryDatabase("TodoList"));
    services.AddMvc()
        .SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    // Register the Swagger generator, defining 1 or more Swagger documents
    services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1", new Info { Title = "My API", Version = "v1" });
    });
}

```

Importe o namespace a seguir para usar a classe `Info`:

```
using Swashbuckle.AspNetCore.Swagger;
```

No método `Startup.Configure`, habilite o middleware para atender ao documento JSON gerado e à interface do usuário do Swagger:

```

public void Configure(IApplicationBuilder app)
{
    // Enable middleware to serve generated Swagger as a JSON endpoint.
    app.UseSwagger();

    // Enable middleware to serve swagger-ui (HTML, JS, CSS, etc.),
    // specifying the Swagger JSON endpoint.
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "My API V1");
    });

    app.UseMvc();
}

```

A chamada do método `UseSwaggerUI` precedente habilita o [middleware de arquivos estáticos](#). Se você estiver direcionando ao .NET Framework ou ao .NET Core 1.x, adicione o pacote NuGet [Microsoft.AspNetCore.StaticFiles](#) ao projeto.

Inicie o aplicativo e navegue até `http://localhost:<port>/swagger/v1/swagger.json`. O documento gerado que descreve os pontos de extremidade é exibido conforme é mostrado na [Especificação do Swagger \(swagger.json\)](#).

A interface do usuário do Swagger pode ser encontrada em `http://localhost:<port>/swagger`. Explore a API por meio da interface do usuário do Swagger e incorpore-a em outros programas.

## TIP

Para atender à interface do usuário do Swagger na raiz do aplicativo (`http://localhost:<port>/`), defina a propriedade `RoutePrefix` como uma cadeia de caracteres vazia:

```
app.UseSwaggerUI(c =>
{
    c.SwaggerEndpoint("/swagger/v1/swagger.json", "My API V1");
    c.RoutePrefix = string.Empty;
});
```

Se estiver usando diretórios com o IIS ou um proxy reverso, defina o ponto de extremidade do Swagger como um caminho relativo usando o prefixo `./`. Por exemplo, `./swagger/v1/swagger.json`. Usar o `/swagger/v1/swagger.json` instrui o aplicativo a procurar o arquivo JSON na raiz verdadeira da URL (mais o prefixo da rota, se usado). Por exemplo, use `http://localhost:<port>/<route_prefix>/swagger/v1/swagger.json` em vez de `http://localhost:<port>/<virtual_directory>/<route_prefix>/swagger/v1/swagger.json`.

## Personalizar e estender

O Swagger fornece opções para documentar o modelo de objeto e personalizar a interface do usuário para corresponder ao seu tema.

### Descrição e informações da API

A ação de configuração passada para o método `AddSwaggerGen` adiciona informações como o autor, a licença e a descrição:

```
// Register the Swagger generator, defining 1 or more Swagger documents
services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new Info
    {
        Version = "v1",
        Title = "ToDo API",
        Description = "A simple example ASP.NET Core Web API",
        TermsOfService = "None",
        Contact = new Contact
        {
            Name = "Shayne Boyer",
            Email = string.Empty,
            Url = "https://twitter.com/spboyer"
        },
        License = new License
        {
            Name = "Use under LICX",
            Url = "https://example.com/license"
        }
    });
});
```

A interface do usuário do Swagger exibe as informações da versão:

The screenshot shows the Swagger UI interface for a 'ToDo API'. The title 'ToDo API' is displayed with a 'v1' badge. Below it, there's a link to '/swagger/v1/swagger.json'. A descriptive text block states: 'A simple example ASP.NET Core Web API' followed by links to 'Terms of service', 'Shayne Boyer - Website', and 'Use under LICX'. The entire title and descriptive block are enclosed in a red rectangular border.

## comentários XML

Comentários XML podem ser habilitados com as seguintes abordagens:

- [Visual Studio](#)
- [Visual Studio para Mac](#)
- [Visual Studio Code](#)
- [CLI do .NET Core](#)
- Clique com o botão direito do mouse no projeto no **Gerenciador de Soluções** e selecione **Editar <nome\_do\_projeto>.csproj**.
- Manualmente, adicione as linhas destacadas ao arquivo `.csproj`:

```
<PropertyGroup>
  <GenerateDocumentationFile>true</GenerateDocumentationFile>
  <NoWarn>$(NoWarn);1591</NoWarn>
</PropertyGroup>
```

- Clique com o botão direito do mouse no projeto no **Gerenciador de Soluções** e selecione **Propriedades**.
- Marque a caixa **Arquivo de documentação XML** na seção **Saída** da guia **Build**.

A habilitação de comentários XML fornece informações de depuração para os membros e os tipos públicos não documentados. Os membros e tipos não documentados são indicados por mensagem de aviso. Por exemplo, a seguinte mensagem indica uma violação do código de aviso 1591:

```
warning CS1591: Missing XML comment for publicly visible type or member 'TodoController.GetAll()'
```

Para suprimir os avisos de todo o projeto, defina uma lista separada por ponto e vírgula dos códigos de aviso a serem ignorados no arquivo do projeto. Acrescentar os códigos de aviso ao `$(NoWarn);` também aplica os valores padrão C#.

```
<PropertyGroup>
  <GenerateDocumentationFile>true</GenerateDocumentationFile>
  <NoWarn>$(NoWarn);1591</NoWarn>
</PropertyGroup>
```

```
<PropertyGroup>
<DocumentationFile>bin\$(Configuration)\$(TargetFramework)\$(AssemblyName).xml</DocumentationFile>
<NoWarn>$(NoWarn);1591</NoWarn>
</PropertyGroup>
```

Para suprimir avisos somente para membros específicos, coloque o código nas diretivas de pré-processador `#pragma warning`. Essa abordagem é útil para o código que não deve ser exposto por meio dos documentos da API. No exemplo a seguir, o código de aviso CS1591 é ignorado para toda a classe `Program`. A imposição do código de aviso é restaurada no fechamento da definição de classe. Especifique vários códigos de aviso com uma lista delimitada por vírgulas.

```
namespace TodoApi
{
#pragma warning disable CS1591
    public class Program
    {
        public static void Main(string[] args) =>
            BuildWebHost(args).Run();

        public static IWebHost BuildWebHost(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>()
                .Build();
    }
#pragma warning restore CS1591
}
```

Configure o Swagger para usar o arquivo XML gerado. Para sistemas operacionais Linux ou que não sejam Windows, os caminhos e nomes de arquivo podem diferenciar maiúsculas de minúsculas. Por exemplo, um arquivo `TodoApi.XML` é válido no Windows, mas não no CentOS.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<TodoContext>(opt =>
        opt.UseInMemoryDatabase("TodoList"));
    services.AddMvc()
        .SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    // Register the Swagger generator, defining 1 or more Swagger documents
    services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1", new Info
        {
            Version = "v1",
            Title = "ToDo API",
            Description = "A simple example ASP.NET Core Web API",
            TermsOfService = "None",
            Contact = new Contact
            {
                Name = "Shayne Boyer",
                Email = string.Empty,
                Url = "https://twitter.com/spboyer"
            },
            License = new License
            {
                Name = "Use under LICX",
                Url = "https://example.com/license"
            }
        });
    });

    // Set the comments path for the Swagger JSON and UI.
    var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
    var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);
    c.IncludeXmlComments(xmlPath);
});

}
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<TodoContext>(opt =>
        opt.UseInMemoryDatabase("TodoList"));
    services.AddMvc();

    // Register the Swagger generator, defining 1 or more Swagger documents
    services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1", new Info
        {
            Version = "v1",
            Title = "ToDo API",
            Description = "A simple example ASP.NET Core Web API",
            TermsOfService = "None",
            Contact = new Contact
            {
                Name = "Shayne Boyer",
                Email = string.Empty,
                Url = "https://twitter.com/spboyer"
            },
            License = new License
            {
                Name = "Use under LICX",
                Url = "https://example.com/license"
            }
        });
    });

    // Set the comments path for the Swagger JSON and UI.
    var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
    var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);
    c.IncludeXmlComments(xmlPath);
});

}
```

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<TodoContext>(opt =>
        opt.UseInMemoryDatabase("TodoList"));
    services.AddMvc();

    // Register the Swagger generator, defining 1 or more Swagger documents
    services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1", new Info
        {
            Version = "v1",
            Title = "ToDo API",
            Description = "A simple example ASP.NET Core Web API",
            TermsOfService = "None",
            Contact = new Contact
            {
                Name = "Shayne Boyer",
                Email = string.Empty,
                Url = "https://twitter.com/spboyer"
            },
            License = new License
            {
                Name = "Use under LICX",
                Url = "https://example.com/license"
            }
        });
    });

    // Set the comments path for the Swagger JSON and UI.
    var xmlFile = $"{Assembly.GetEntryAssembly().GetName().Name}.xml";
    var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);
    c.IncludeXmlComments(xmlPath);
}
}

```

No código anterior, a [Reflexão](#) é usada para criar um nome de arquivo XML correspondente ao do projeto de API Web. A propriedade [AppContext.BaseDirectory](#) é usada para construir um caminho para o arquivo XML.

Adicionar comentários de barra tripla a uma ação aprimora a interface do usuário do Swagger adicionando a descrição ao cabeçalho da seção. Adicione um elemento [`<summary>`](#) acima da ação `Delete` :

```

/// <summary>
/// Deletes a specific TodoItem.
/// </summary>
/// <param name="id"></param>
[HttpDelete("{id}")]
public IActionResult Delete(long id)
{
    var todo = _context.TodoItems.Find(id);

    if (todo == null)
    {
        return NotFound();
    }

    _context.TodoItems.Remove(todo);
    _context.SaveChanges();

    return NoContent();
}

```

A interface do usuário do Swagger exibe o texto interno do elemento `<summary>` do código anterior:

**DELETE** /api/Todo/{id} Deletes a specific TodoItem.

**Parameters**

Name Description

**id** \* required  
integer  
(path)

**Responses** Response content type application/json

Code Description

200 Success

A interface do usuário é controlada pelo esquema JSON gerado:

```

"delete": {
    "tags": [
        "Todo"
    ],
    "summary": "Deletes a specific TodoItem.",
    "operationId": "ApiTodoByIdDelete",
    "consumes": [],
    "produces": [],
    "parameters": [
        {
            "name": "id",
            "in": "path",
            "description": "",
            "required": true,
            "type": "integer",
            "format": "int64"
        }
    ],
    "responses": {
        "200": {
            "description": "Success"
        }
    }
}

```

Adicione um elemento <remarks> na documentação do método da ação Create. Ele complementa as informações especificadas no elemento <summary> e fornece uma interface de usuário do Swagger mais robusta. O conteúdo do elemento <remarks> pode consistir em texto, JSON ou XML.

```

/// <summary>
/// Creates a TodoItem.
/// </summary>
/// <remarks>
/// Sample request:
///
///     POST /Todo
///     {
///         "id": 1,
///         "name": "Item1",
///         "isComplete": true
///     }
///
/// </remarks>
/// <param name="item"></param>
/// <returns>A newly created TodoItem</returns>
/// <response code="201">Returns the newly created item</response>
/// <response code="400">If the item is null</response>
[HttpPost]
[ProducesResponseType(typeof(TodoItem), 201)]
[ProducesResponseType(400)]
public IActionResult Create([FromBody] TodoItem item)
{
    if (item == null)
    {
        return BadRequest();
    }

    _context.TodoItems.Add(item);
    _context.SaveChanges();

    return CreatedAtRoute("GetTodo", new { id = item.Id }, item);
}

```

```

/// <summary>
/// Creates a TodoItem.
/// </summary>
/// <remarks>
/// Sample request:
///
///     POST /Todo
///     {
///         "id": 1,
///         "name": "Item1",
///         "isComplete": true
///     }
///
/// </remarks>
/// <param name="item"></param>
/// <returns>A newly created TodoItem</returns>
/// <response code="201">Returns the newly created item</response>
/// <response code="400">If the item is null</response>
[HttpPost]
[ProducesResponseType(201)]
[ProducesResponseType(400)]
public ActionResult<TodoItem> Create(TodoItem item)
{
    _context.TodoItems.Add(item);
    _context.SaveChanges();

    return CreatedAtRoute("GetTodo", new { id = item.Id }, item);
}

```

Observe os aprimoramentos da interface do usuário com esses comentários adicionais:

**POST** /api/Todo Creates a TodoItem.

Sample request:

```
POST /Todo
{
    "id": 1,
    "name": "Item1",
    "isComplete": true
}
```

## Anotações de dados

Decore o modelo com atributos, encontrados no namespace [System.ComponentModel.DataAnnotations](#), para ajudar a gerar os componentes da interface do usuário do Swagger.

Adicione o atributo `[Required]` à propriedade `Name` da classe `TodoItem`:

```
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

namespace TodoApi.Models
{
    public class TodoItem
    {
        public long Id { get; set; }

        [Required]
        public string Name { get; set; }

        [DefaultValue(false)]
        public bool IsComplete { get; set; }
    }
}
```

A presença desse atributo altera o comportamento da interface do usuário e altera o esquema JSON subjacente:

```
"definitions": {
    "TodoItem": {
        "required": [
            "name"
        ],
        "type": "object",
        "properties": {
            "id": {
                "format": "int64",
                "type": "integer"
            },
            "name": {
                "type": "string"
            },
            "isComplete": {
                "default": false,
                "type": "boolean"
            }
        }
    }
},
```

Adicione o atributo `[Produces("application/json")]` ao controlador da API. Sua finalidade é declarar que as ações do controlador permitem o tipo de conteúdo de resposta *application/json*:

```
[Produces("application/json")]
[Route("api/[controller]")]
public class TodoController : ControllerBase
{
    private readonly TodoContext _context;
```

```
[Produces("application/json")]
[Route("api/[controller]")]
[ApiController]
public class TodoController : ControllerBase
{
    private readonly TodoContext _context;
```

A lista suspensa **Tipo de Conteúdo de Resposta** seleciona esse tipo de conteúdo como o padrão para ações GET do controlador:

The screenshot shows the Swagger UI interface for a `GET /api/Todo` endpoint. The 'Responses' section is highlighted with a red box around the 'Response content type' dropdown, which is set to `application/json`. Other visible sections include 'Parameters' (No parameters) and a 'Try it out' button.

À medida que aumenta o uso de anotações de dados na API Web, a interface do usuário e as páginas de ajuda da API se tornam mais descriptivas e úteis.

### Descrever os tipos de resposta

Os desenvolvedores que usam uma API Web estão mais preocupados com o que é retornado—, especificamente, os tipos de resposta e os códigos de erro (se eles não forem padrão). Os tipos de resposta e os códigos de erro são indicados nos comentários XML e nas anotações de dados.

A ação `Create` retorna um código de status HTTP 201 em caso de sucesso. Um código de status HTTP 400 é retornado quando o corpo da solicitação postada é nulo. Sem a documentação adequada na interface do usuário do Swagger, o consumidor não tem conhecimento desses resultados esperados. Corrija esse problema adicionando as linhas realçadas no exemplo a seguir:

```

/// <summary>
/// Creates a TodoItem.
/// </summary>
/// <remarks>
/// Sample request:
///
///     POST /Todo
/// {
///     "id": 1,
///     "name": "Item1",
///     "isComplete": true
/// }
///
/// </remarks>
/// <param name="item"></param>
/// <returns>A newly created TodoItem</returns>
/// <response code="201">Returns the newly created item</response>
/// <response code="400">If the item is null</response>
[HttpPost]
[ProducesResponseType(typeof(TodoItem), 201)]
[ProducesResponseType(400)]
public IActionResult Create([FromBody] TodoItem item)
{
    if (item == null)
    {
        return BadRequest();
    }

    _context.TodoItems.Add(item);
    _context.SaveChanges();

    return CreatedAtRoute("GetTodo", new { id = item.Id }, item);
}

```

```

/// <summary>
/// Creates a TodoItem.
/// </summary>
/// <remarks>
/// Sample request:
///
///     POST /Todo
/// {
///     "id": 1,
///     "name": "Item1",
///     "isComplete": true
/// }
///
/// </remarks>
/// <param name="item"></param>
/// <returns>A newly created TodoItem</returns>
/// <response code="201">Returns the newly created item</response>
/// <response code="400">If the item is null</response>
[HttpPost]
[ProducesResponseType(201)]
[ProducesResponseType(400)]
public ActionResult<TodoItem> Create(TodoItem item)
{
    _context.TodoItems.Add(item);
    _context.SaveChanges();

    return CreatedAtRoute("GetTodo", new { id = item.Id }, item);
}

```

A interface do usuário do Swagger agora documenta claramente os códigos de resposta HTTP esperados:

The screenshot shows the Swagger UI interface for a POST operation. At the top, it says "Responses" and "Response content type application/json". Below that, there's a table with columns "Code" and "Description". The first row shows a 201 status with the description "Returns the newly created item". A red box highlights this row. Below it, there's a "Example Value" button and a "Model" link. The model is shown as a JSON schema: { "id": 0, "name": "string", "isComplete": false }. The second row shows a 400 status with the description "If the item is null".

No ASP.NET Core 2.2 ou posterior, as convenções podem ser usadas como uma alternativa para decorar explicitamente as ações individuais com `[ProducesResponseType]`. Para obter mais informações, consulte [Usar convenções de API Web](#).

### Personalizar a interface do usuário

A interface do usuário de estoque é apresentável e funcional. No entanto, as páginas de documentação da API devem representar a sua marca ou o seu tema. Para inserir sua marca nos componentes do Swashbuckle é necessário adicionar recursos para atender aos arquivos estáticos e criar a estrutura de pasta para hospedar esses arquivos.

Se você estiver direcionando ao .NET Framework ou ao .NET Core 1.x, adicione o pacote do NuGet [Microsoft.AspNetCore.StaticFiles](#) ao projeto:

```
<PackageReference Include="Microsoft.AspNetCore.StaticFiles" Version="2.0.0" />
```

O pacote do NuGet anterior já estará instalado se você estiver direcionando ao .NET Core 2.x e usando o [metapackage](#).

Habilitar o middleware de arquivos estáticos:

```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles();

    // Enable middleware to serve generated Swagger as a JSON endpoint.
    app.UseSwagger();

    // Enable middleware to serve swagger-ui (HTML, JS, CSS, etc.),
    // specifying the Swagger JSON endpoint.
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "My API V1");
    });

    app.UseMvc();
}
```

Adquira o conteúdo da pasta `dist` do [repositório GitHub da interface do usuário do Swagger](#). Essa pasta contém os ativos necessários para a página da interface do usuário do Swagger.

Crie uma pasta `swagger/wwwroot/ui` e copie para ela o conteúdo da pasta `dist`.

Crie um arquivo `custom.css` em `swagger/wwwroot/ui`, com o seguinte CSS para personalizar o cabeçalho da página:

```
.swagger-ui .topbar {  
    background-color: #000;  
    border-bottom: 3px solid #547f00;  
}
```

Referencie `custom.css` no arquivo `index.html` depois de todos os outros arquivos CSS:

```
<link href="https://fonts.googleapis.com/css?  
family=Open+Sans:400,700|Source+Code+Pro:300,600|Titillium+Web:400,600,700" rel="stylesheet">  
<link rel="stylesheet" type="text/css" href="./swagger-ui.css">  
<link rel="stylesheet" type="text/css" href="custom.css">
```

Navegue até a página `index.html` em `http://localhost:<port>/swagger/ui/index.html`. Digite `http://localhost:<port>/swagger/v1/swagger.json` na caixa de texto do cabeçalho e clique no botão **Explorar**. A página resultante será semelhante ao seguinte:

# ToDo API v1

<http://localhost:49253/swagger/v1/swagger.json>

A simple example ASP.NET Core Web API

[Terms of service](#)

[Shayne Boyer - Website](#)

[Use under LICX](#)

## Todo

GET

/api/Todo

POST

/api/Todo Creates a Todolitem.

GET

/api/Todo/{id}

PUT

/api/Todo/{id}

DELETE

/api/Todo/{id} Deletes a specific Todolitem.

## Models

Todolitem >

Há muito mais que você pode fazer com a página. Consulte as funcionalidades completas para os recursos de interface do usuário no [repositório GitHub da interface do usuário do Swagger](#).

# Introdução ao NSwag e ao ASP.NET Core

16/01/2019 • 12 minutes to read • [Edit Online](#)

Por [Christoph Nienaber](#), [Rico Suter](#) e [Dave Brock](#)

[Exibir ou baixar código de exemplo \(como baixar\)](#)

[Exibir ou baixar código de exemplo \(como baixar\)](#)

NSwag oferece os seguintes recursos:

- A capacidade de utilizar a interface do usuário do Swagger e o gerador do Swagger.
- Recursos flexíveis de geração de código.

Com o NSwag, você não precisa de uma API existente—, pois pode usar APIs de terceiros que incorporam o Swagger e geram uma implementação de cliente. O NSwag permite acelerar o ciclo de desenvolvimento e adaptar-se facilmente às alterações na API.

## Registrar o middleware do NSwag

Registre o middleware do NSwag para:

- Gerar a especificação do Swagger para a API Web implementada.
- Oferecer a interface do usuário do Swagger para navegar e testar a API Web.

Para usar o middleware ASP.NET Core do [NSwag](#), instale o pacote do NuGet [NSwag.AspNetCore](#). Este pacote contém o middleware para gerar e oferecer a especificação do Swagger, interface do usuário do Swagger (v2 e v3) e a [interface do usuário do ReDoc](#).

Use uma das seguintes abordagens para instalar o pacote do NuGet do NSwag:

- [Visual Studio](#)
- [Visual Studio para Mac](#)
- [Visual Studio Code](#)
- [CLI do .NET Core](#)
- Da janela **Console do Gerenciador de Pacotes**:
  - Acesse **Exibição > Outras Janelas > Console do Gerenciador de Pacotes**
  - Navegue para o diretório no qual o arquivo *TodoApi.csproj* está localizado
  - Execute o seguinte comando:

```
Install-Package NSwag.AspNetCore
```

- Da caixa de diálogo **Gerenciar Pacotes NuGet**:
  - Clique com o botão direito do mouse no projeto em **Gerenciador de Soluções > Gerenciar Pacotes NuGet**
  - Defina a **Origem do pacote** para "nuget.org"
  - Insira "NSwag.AspNetCore" na caixa de pesquisa
  - Selecione o pacote "NSwag.AspNetCore" na guia **Procurar** e clique em **Instalar**

## Adicionar e configurar o middleware do Swagger

Adicione e configure o Swagger em seu aplicativo ASP.NET Core, executando as seguintes etapas na classe

`Startup`:

- Importe os seguintes namespaces:

```
using Newtonsoft.Json;
using NSwag.AspNetCore;
```

- No método `ConfigureServices`, registre os serviços obrigatórios do Swagger:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<TodoContext>(opt =>
        opt.UseInMemoryDatabase("Todolist"));
    services.AddMvc();

    // Register the Swagger services
    services.AddSwaggerDocument();
}
```

- No método `Configure`, habilite o middleware para atender à especificação do Swagger gerado e à interface do usuário do Swagger:

```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles();

    // Register the Swagger generator and the Swagger UI middlewares
    app.UseSwagger();
    app.UseSwaggerUi3();

    app.UseMvc();
}
```

- Inicie o aplicativo. Navegue até:

- `http://localhost:<port>/swagger` para exibir a interface do usuário do Swagger.
- `http://localhost:<port>/swagger/v1/swagger.json` para exibir a especificação do Swagger.

## Geração de código

Você pode tirar proveito dos recursos de geração de código do NSwag, escolhendo uma das opções a seguir:

- [NSwagStudio](#) – um aplicativo da área de trabalho do Windows para geração do código do cliente da API em C# ou TypeScript.
- Pacotes NuGet [NSwag.CodeGeneration.CSharp](#) ou [NSwag.CodeGeneration.TypeScript](#) para a geração de código dentro do seu projeto.
- NSwag na [linha de comando](#).
- O pacote NuGet [NSwag.MSBuild](#).

### Gerar o código com NSwagStudio

- Instale o NSwagStudio, seguindo as instruções no [repositório GitHub do NSwagStudio](#).
- Inicie o NSwagStudio e insira a URL do arquivo `swagger.json` na caixa de texto **URL de Especificação do Swagger**. Por exemplo, `http://localhost:44354/swagger/v1/swagger.json`.

- Clique no botão **Criar Cópia local** para gerar uma representação JSON de sua especificação do Swagger.

### Input: Swagger Specification

#### Runtime

NetCore22

Specifies the used command line binary; should match the selected assembly type.

**Default Variables ('foo=bar,baz=bar'), usage: \${foo}**

Web API or ASP.NET Core via Reflection (deprecated)	
JSON Schema	.NET Assembly
Swagger Specification	ASP.NET Core via API Explorer

**Swagger Specification URL:**

<https://localhost:44354/swagger/v1/swagger.json>

**Swagger Specification JSON (if specified, the URL is ignored):**

```

1 {
2   "x-generator": "NSwag v11.19.2.0 (NJsonSchema v9.10.14-rc.1)",
3   "swagger": "2.0",
4   "info": {
5     "title": "My Title",
6     "version": "1.0.0"
7   },
8   "host": "localhost:44354",
9   "schemes": [
10     "https"
11   ],
12   "consumes": [
13     "application/json-patch+json"
14   ]
15 }
```

- Na área **Saídas**, marque a caixa de seleção **Cliente CSharp**. Dependendo do seu projeto, você também pode escolher **Cliente TypeScript** ou **Controlador da API Web CSharp**. Se você selecionar **Controlador da API Web do CSharp**, uma especificação de serviço recompilará o serviço, que servirá como uma geração inversa.
- Clique em **Gerar Saídas** para produzir uma implementação completa de cliente em C# do projeto *TodoApi.NSwag*. Para ver o código de cliente gerado, clique na guia **Cliente do CSharp**:

```

//-----
// <auto-generated>
//   Generated using the NSwag toolchain v12.0.9.0 (NJsonSchema v9.13.10.0 (Newtonsoft.Json v11.0.0.0))
// (http://NSwag.org)
// </auto-generated>
//-----

namespace MyNamespace
{
    #pragma warning disable

    [System.CodeDom.Compiler.GeneratedCode("NSwag", "12.0.9.0 (NJsonSchema v9.13.10.0 (Newtonsoft.Json v11.0.0.0))")]
    public partial class TodoClient
    {
        private string _baseUrl = "https://localhost:44354";
        private System.Net.Http.HttpClient _httpClient;
        private System.Lazy<Newtonsoft.Json.JsonSerializerSettings> _settings;

        public TodoClient(System.Net.Http.HttpClient httpClient)
        {
            _httpClient = httpClient;
            _settings = new System.Lazy<Newtonsoft.Json.JsonSerializerSettings>(() =>
            {
                var settings = new Newtonsoft.Json.JsonSerializerSettings();
                UpdateJsonSerializerSettings(settings);
                return settings;
            });
        }

        public string BaseUrl
        {
            get { return _baseUrl; }
            set { _baseUrl = value; }
        }

        // code omitted for brevity
    }
}

```

#### TIP

O código do cliente em C# é gerado com base em seleções na guia **Configurações**. Modifique as configurações para executar tarefas como geração de método síncrono e renomeação de namespace padrão.

- Copie o código C# gerado em um arquivo no projeto do cliente que consumirá a API.
- Inicie o consumo da API Web:

```

var todoClient = new TodoClient();

// Gets all to-dos from the API
var allTodos = await todoClient.GetAllAsync();

// Create a new TodoItem, and save it via the API.
var createdTodo = await todoClient.CreateAsync(new TodoItem());

// Get a single to-do by ID
var foundTodo = await todoClient.GetByIdAsync(1);

```

## Personalizar a documentação da API

O Swagger fornece opções para documentar o modelo de objeto para facilitar o consumo da API Web.

## Descrição e informações da API

No método `Startup.ConfigureServices`, uma ação de configuração passada para o método `AddSwaggerDocument` adiciona informações como o autor, a licença e a descrição:

```
services.AddSwaggerDocument(config =>
{
    config.PostProcess = document =>
    {
        document.Info.Version = "v1";
        document.Info.Title = "ToDo API";
        document.Info.Description = "A simple ASP.NET Core web API";
        document.Info.TermsOfService = "None";
        document.Info.Contact = new NSwag.SwaggerContact
        {
            Name = "Shayne Boyer",
            Email = string.Empty,
            Url = "https://twitter.com/spboyer"
        };
        document.Info.License = new NSwag.SwaggerLicense
        {
            Name = "Use under LICX",
            Url = "https://example.com/license"
        };
    };
});
```

A interface do usuário do Swagger exibe as informações da versão:

The screenshot shows the Swagger UI interface for a 'ToDo API'. At the top, there's a green header with the 'swagger' logo. Below it, the main title 'ToDo API' is displayed with a small 'v1' badge next to it. Underneath the title, there's a link '[ Base URL: localhost:44354 ] /swagger/v1/swagger.json'. The main content area is titled 'A simple ASP.NET Core web API' and contains several blue links: 'Terms of service', 'Shayne Boyer - Website', and 'Use under LICX'.

## comentários XML

Para habilitar os comentários XML, execute as seguintes etapas:

- [Visual Studio](#)
  - [Visual Studio para Mac](#)
  - [Visual Studio Code](#)
- 
- Clique com o botão direito do mouse no projeto no **Gerenciador de Soluções** e selecione **Editar <nome\_do\_projeto>.csproj**.
  - Manualmente, adicione as linhas destacadas ao arquivo `.csproj`:

```
<PropertyGroup>
  <GenerateDocumentationFile>true</GenerateDocumentationFile>
  <NoWarn>$({NoWarn});1591</NoWarn>
</PropertyGroup>
```

- Clique com o botão direito do mouse no projeto no **Gerenciador de Soluções** e selecione **Propriedades**
- Marque a caixa **Arquivo de documentação XML** na seção **Saída** da guia **Build**

### Anotações de dados

Como o NSwag usa [Reflexão](#), e o tipo recomendado de retorno para ações da API Web é [IActionResult](#), ele não consegue inferir o que sua ação está fazendo e o que ela retorna.

Considere o exemplo a seguir:

```
[HttpPost]
public IActionResult Create([FromBody] TodoItem item)
{
    if (item == null)
    {
        return BadRequest();
    }

    _context.TodoItems.Add(item);
    _context.SaveChanges();

    return CreatedAtRoute("GetTodo", new { id = item.Id }, item);
}
```

A ação anterior retorna `IActionResult`, mas dentro da ação, ela está retornando [CreatedAtRoute](#) ou [BadRequest](#). Use anotações de dados para informar aos clientes quais códigos de status HTTP esta ação costuma retornar. Decore a ação com os seguintes atributos:

```
[ProducesResponseType(typeof(TodoItem), 201)] // Created
[ProducesResponseType(400)] // BadRequest
```

Como o NSwag usa [Reflection](#), e o tipo de retorno recomendado para as ações da API Web é [ActionResult<T>](#), ele só consegue inferir o tipo de retorno definido por `T`. Não é possível inferir automaticamente outros tipos de retorno possíveis.

Considere o exemplo a seguir:

```
[HttpPost]
public ActionResult<TodoItem> Create(TodoItem item)
{
    _context.TodoItems.Add(item);
    _context.SaveChanges();

    return CreatedAtRoute("GetTodo", new { id = item.Id }, item);
}
```

A ação anterior retorna `ActionResult<T>`. Dentro da ação, ela está retornando [CreatedAtRoute](#). Como o controlador está decorado com o atributo [ApiController](#), uma resposta [BadRequest](#) também é possível. Para obter mais informações, veja [Respostas automáticas HTTP 400](#). Use anotações de dados para informar aos clientes quais códigos de status HTTP esta ação costuma retornar. Decore a ação com os seguintes atributos:

```
[ProducesResponseType(201)]      // Created  
[ProducesResponseType(400)]      // BadRequest
```

No ASP.NET Core 2.2 ou posterior, é possível usar as convenções em vez de decorar explicitamente as ações individuais com `[ProducesResponseType]`. Para obter mais informações, consulte [Usar convenções de API Web](#).

O gerador do Swagger agora pode descrever essa ação precisamente e os clientes gerados conseguem saber o que recebem ao chamar o ponto de extremidade. Recomendamos decorar todas as ações com esses atributos.

Para obter diretrizes sobre quais respostas HTTP as ações da API devem retornar, confira a [Especificação RFC 7231](#).

# Tipos de retorno de ação do controlador na API Web ASP.NET Core

16/01/2019 • 9 minutes to read • [Edit Online](#)

Por [Scott Addie](#)

[Exibir ou baixar código de exemplo \(como baixar\)](#)

O ASP.NET Core oferece as seguintes opções para os tipos de retorno da ação do controlador da API Web:

- [Tipo específico](#)
- [IActionResult](#)
- [ActionResult<T>](#)
- [Tipo específico](#)
- [IActionResult](#)

Este documento explica quando é mais adequado usar cada tipo de retorno.

## Tipo específico

A ação mais simples retorna um tipo de dados complexo ou primitivo (por exemplo, `string` ou um tipo de objeto personalizado). Considere a seguinte ação, que retorna uma coleção de objetos `Product` personalizados:

```
[HttpGet]
public IEnumerable<Product> Get()
{
    return _repository.GetProducts();
}
```

Sem condições conhecidas contra as quais se proteger durante a execução da ação, retornar um tipo específico pode ser suficiente. A ação anterior não aceita parâmetros, assim, validação de restrições de parâmetro não é necessária.

Quando condições conhecidas precisarem ser incluídas em uma ação, vários caminhos de retorno serão introduzidos. Nesse caso, é comum combinar um tipo de retorno [ActionResult](#) com o tipo de retorno primitivo ou complexo. [IActionResult](#) ou [ActionResult<T>](#) é necessário para acomodar esse tipo de ação.

## Tipo [IActionResult](#)

O tipo de retorno [IActionResult](#) é apropriado quando vários tipos de retorno [ActionResult](#) são possíveis em uma ação. Os tipos `ActionResult` representam vários códigos de status HTTP. Alguns tipos de retorno comuns que se enquadram nessa categoria são [BadRequestResult](#) (400) [NotFoundResult](#) (404) e [OkObjectResult](#) (200).

Porque há vários tipos de retorno e caminhos na ação, o uso liberal do atributo `[ProducesResponseType]` é necessário. Esse atributo produz detalhes de resposta mais descritivos para páginas de ajuda da API geradas por ferramentas como [Swagger](#). `[ProducesResponseType]` indica os tipos conhecidos e os códigos de status HTTP a serem retornados pela ação.

### Ação síncrona

Considere a seguinte ação síncrona em que há dois tipos de retorno possíveis:

```
[HttpGet("{id}")]
[ProducesResponseType(200, Type = typeof(Product))]
[ProducesResponseType(404)]
public IActionResult GetById(int id)
{
    if (!_repository.TryGetProduct(id, out var product))
    {
        return NotFound();
    }

    return Ok(product);
}
```

Na ação anterior, um código de status 404 é retornado quando o produto representado por `id` não existe no armazenamento de dados subjacente. O método auxiliar `NotFound` é invocado como um atalho para `return new NotFoundResult();`. Se o produto existir, um objeto `Product` que representa o conteúdo será retornado com um código de status 200. O método auxiliar `OK` é invocado como a forma abreviada de `return new OkObjectResult(product);`.

## Ação assíncrona

Considere a seguinte ação assíncrona em que há dois tipos de retorno possíveis:

```
[HttpPost]
[ProducesResponseType(201, Type = typeof(Product))]
[ProducesResponseType(400)]
public async Task<IActionResult> CreateAsync([FromBody] Product product)
{
    if (product.Description.Contains("XYZ Widget"))
    {
        return BadRequest();
    }

    await _repository.AddProductAsync(product);

    return CreatedAtAction(nameof(GetById), new { id = product.Id }, product);
}
```

No código anterior:

- O tempo de execução do ASP.NET Core retorna um código de status 400 (`BadRequest`) quando a descrição do produto contém "Widget XYZ".
- O método `CreatedAtAction` gera um código de status 201 quando um produto é criado. Nesse caminho de código, o objeto `Product` é retornado.

Por exemplo, o modelo a seguir indica que as solicitações devem incluir as propriedades `Name` e `Description`. Portanto, a falha em fornecer `Name` e `Description` na solicitação causa falha na validação do modelo.

```
public class Product
{
    public int Id { get; set; }

    [Required]
    public string Name { get; set; }

    [Required]
    public string Description { get; set; }
}
```

Se o atributo `[ApiController]` no ASP.NET Core 2.1 ou posterior for aplicado, erros de validação de modelo

resultarão em um código de status 400. Para obter mais informações, veja [Respostas automáticas HTTP 400](#).

## Tipo ActionResult<T>

O ASP.NET Core 2.1 apresenta o tipo de retorno `ActionResult<T>` para ações do controlador de API Web. Permite que você retorne um tipo derivado de `ActionResult` ou retorne um [tipo específico](#). `ActionResult<T>` oferece os seguintes benefícios em relação ao [tipo IActionResult](#):

- O atributo `[ProducesResponseType]` pode ter sua propriedade `Type` excluída. Por exemplo, `[ProducesResponseType(200, Type = typeof(Product))]` é simplificado para `[ProducesResponseType(200)]`. O tipo de retorno esperado da ação é inferido do `T` em `ActionResult<T>`.
- [Operadores de conversão implícita](#) são compatíveis com a conversão de `T` e `ActionResult` em `ActionResult<T>`. `T` converte em `ObjectResult`, o que significa que `return new ObjectResult(T);` é simplificado para `return T;`.

C# não dá suporte a operadores de conversão implícita em interfaces. Consequentemente, a conversão da interface para um tipo concreto é necessário para usar `ActionResult<T>`. Por exemplo, o uso de `IEnumerable` no exemplo a seguir não funciona:

```
[HttpGet]
public ActionResult<IEnumerable<Product>> Get()
{
    return _repository.GetProducts();
}
```

Uma opção para corrigir o código anterior é retornar a `_repository.GetProducts().ToList();`.

A maioria das ações tem um tipo de retorno específico. Condições inesperadas podem ocorrer durante a execução da ação, caso em que o tipo específico não é retornado. Por exemplo, o parâmetro de entrada de uma ação pode falhar na validação do modelo. Nesse caso, é comum retornar o tipo `ActionResult` adequado, em vez do tipo específico.

### Ação síncrona

Considere uma ação síncrona em que há dois tipos de retorno possíveis:

```
[HttpGet("{id}")]
[ProducesResponseType(200)]
[ProducesResponseType(404)]
public ActionResult<Product> GetById(int id)
{
    if (!_repository.TryGetProduct(id, out var product))
    {
        return NotFound();
    }

    return product;
}
```

No código anterior, um código de status 404 é retornado quando o produto não existe no banco de dados. Se o produto existir, o objeto `Product` correspondente será retornado. Antes do ASP.NET Core 2.1, a linha `return product;` teria sido `return Ok(product);`.

**TIP**

Do ASP.NET Core 2.1 em diante, a inferência de origem de associação de parâmetro de ação é habilitada quando uma classe de controlador é decorada com o atributo `[ApiController]`. Um nome de parâmetro correspondente a um nome do modelo de rota é associado automaticamente usando os dados de rota de solicitação. Consequentemente, o parâmetro `id` da ação anterior não é explicitamente anotado com o atributo `[FromRoute]`.

## Ação assíncrona

Considere uma ação assíncrona em que há dois tipos de retorno possíveis:

```
[HttpPost]
[ProducesResponseType(201)]
[ProducesResponseType(400)]
public async Task<ActionResult<Product>> CreateAsync(Product product)
{
    if (product.Description.Contains("XYZ Widget"))
    {
        return BadRequest();
    }

    await _repository.AddProductAsync(product);

    return CreatedAtAction(nameof(GetById), new { id = product.Id }, product);
}
```

No código anterior:

- O tempo de execução do ASP.NET Core retorna um código de status 400 (`BadRequest`) quando:
  - O atributo `[ApiController]` tiver sido aplicado e o modelo de validação falhar.
  - A descrição do produto contém "Widget XYZ".
- O método `CreatedAtAction` gera um código de status 201 quando um produto é criado. Nesse caminho de código, o objeto `Product` é retornado.

**TIP**

Do ASP.NET Core 2.1 em diante, a inferência de origem de associação de parâmetro de ação é habilitada quando uma classe de controlador é decorada com o atributo `[ApiController]`. Parâmetros de tipo complexo são vinculados automaticamente usando o corpo da solicitação. Consequentemente, o parâmetro `product` da ação anterior não é explicitamente anotado com o atributo `[FromBody]`.

## Recursos adicionais

- [Tratar solicitações com controladores no ASP.NET Core MVC](#)
- [Validação de modelo no ASP.NET Core MVC](#)
- [Páginas de ajuda da API Web ASP.NET Core com o Swagger/OpenAPI](#)

# Formatar dados de resposta na API Web ASP.NET Core

30/10/2018 • 16 minutes to read • [Edit Online](#)

Por [Steve Smith](#)

O ASP.NET Core MVC tem suporte interno para formatação de dados de resposta, usando formatos fixos ou em resposta às especificações do cliente.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Resultados da ação específicos a um formato

Alguns tipos de resultado de ação são específicos a um formato específico, como `JsonResult` e `ContentResult`.

As ações podem retornar resultados específicos que são sempre formatados de determinada maneira. Por exemplo, o retorno de um `JsonResult` retornará dados formatados em JSON, independentemente das preferências do cliente. Da mesma forma, o retorno de um `ContentResult` retornará dados de cadeia de caracteres formatados como texto sem formatação (assim como o simples retorno de uma cadeia de caracteres).

### NOTE

Uma ação não precisa retornar nenhum tipo específico; o MVC dá suporte a qualquer valor retornado de objeto. Se uma ação retorna uma implementação `IActionResult` e o controlador herda de `Controller`, os desenvolvedores têm muitos métodos auxiliares correspondentes a muitas das opções. Os resultados de ações que retornam objetos que não são tipos `IActionResult` serão serializados usando a implementação `IOutputFormatter` apropriada.

Para retornar dados em um formato específico de um controlador que herda da classe base `Controller`, use o método auxiliar interno `Json` para retornar JSON e `Content` para texto sem formatação. O método de ação deve retornar o tipo de resultado específico (por exemplo, `JsonResult`) ou `IActionResult`.

Retornando dados formatados em JSON:

```
// GET: api/authors
[HttpGet]
public JsonResult Get()
{
    return Json(_authorRepository.List());
}
```

Resposta de exemplo desta ação:

The screenshot shows the Microsoft Edge developer tools Network tab. A request for `localhost:9664/api/authors` is selected. The response body contains the JSON array `[{"Name": "Steve Smith", "Twitter": "ardalis"}, {"Name": "Rick Anderson", "Twitter": "RickAndMSFT"}]`. The Headers section shows the Content-Type header is `application/json`. The Response Headers section shows the Content-Type header is `application/json; charset=utf-8`.

Name / Path	Protocol	Method	Result / Description	Content type
authors http://localhost:9664/api/	HTTP	GET	200 OK	application/json

Request URL: `http://localhost:9664/api/authors`  
 Request Method: `GET`  
 Status Code: `200 / OK`

Request Headers:

- `Accept: text/html, application/xhtml+xml, image/jxr, */*`
- `Accept-Encoding: gzip, deflate`
- `Accept-Language: en-US, en; q=0.5`
- `Connection: Keep-Alive`
- `Host: localhost:9664`
- `User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)...`

Response Headers:

- `Content-Length: 93`
- `Content-Type: application/json; charset=utf-8`
- `Date: Sun, 01 May 2016 23:33:19 GMT`
- `Server: Kestrel`
- `X-Powered-By: ASP.NET`
- `X-SourceFiles: =?UTF-8?B?QzpcZGV2XGdpdGh1YbhC3B...`

0 errors | 1 request | 93 B transferred | 4.76 ms taken (DOMContentLoaded: 26 ms, load: 43 ms)

Observe que o tipo de conteúdo da resposta é `application/json`, conforme mostrado na lista de solicitações de rede e na seção Cabeçalhos de Resposta. Além disso, observe a lista de opções apresentada pelo navegador (nesse caso, Microsoft Edge) no cabeçalho Accept da seção Cabeçalhos de Solicitação. A técnica atual é ignorar esse cabeçalho. Abaixo, abordamos como obedecê-lo.

Para retornar dados formatados como texto sem formatação, use `ContentResult` e o auxiliar `Content`:

```
// GET api/authors/about
[HttpGet("About")]
public ContentResult About()
{
    return Content("An API listing authors of docs.asp.net.");
}
```

Uma resposta dessa ação:

An API listing authors of docs.asp.net.

Name / Path	Protocol	Method	Result / Description	Content type	Re
about http://localhost:9664/api/authors/	HTTP	GET	200 OK	text/plain	15

Request URL: <http://localhost:9664/api/authors/about>  
 Request Method: GET  
 Status Code: 200 / OK

**Request Headers**

- Accept: text/html, application/xhtml+xml, image/jxr, \*/\*
- Accept-Encoding: gzip, deflate
- Accept-Language: en-US, en; q=0.5
- Connection: Keep-Alive
- Host: localhost:9664
- User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64...

**Response Headers**

- Content-Encoding: gzip
- Content-Length: 152
- Content-Type: text/plain; charset=utf-8
- Date: Sun, 01 May 2016 23:40:02 GMT
- Server: Kestrel
- Vary: Accept-Encoding

0 errors | 1 request | 152 B transferred | 1.19 s taken (DOMContentLoaded: 1.21 s, load: 1.22 s)

Observe, neste caso, que o `Content-Type` retornado é `text/plain`. Também obtenha esse mesmo comportamento usando apenas um tipo de resposta de cadeia de caracteres:

```
// GET api/authors/version
[HttpGet("version")]
public string Version()
{
    return "Version 1.0.0";
}
```

#### TIP

Para ações não triviais com vários tipos de retorno ou várias opções (por exemplo, códigos de status HTTP diferentes com base no resultado das operações executadas), prefira `IActionResult` como o tipo de retorno.

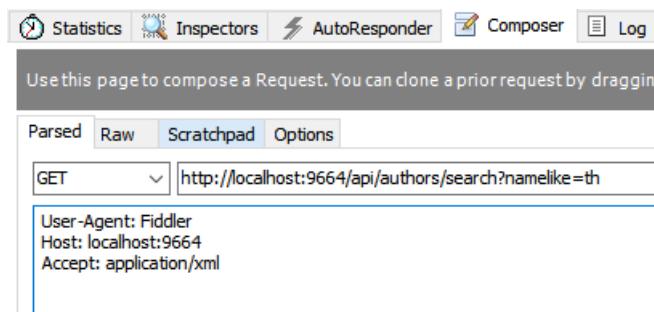
## Negociação de conteúdo

A negociação de conteúdo (*conneg* de forma abreviada) ocorre quando o cliente especifica um **cabeçalho Accept**. O formato padrão usado pelo ASP.NET Core MVC é JSON. A negociação de conteúdo é implementada por `ObjectResult`. Ela também foi desenvolvida com base nos resultados de ação específica de código de status retornados dos métodos auxiliares (que se baseiam em `ObjectResult`). Também retorne um tipo de modelo (uma classe que você definiu como o tipo de transferência de dados) e a estrutura o encapsulará automaticamente em um `ObjectResult` para você.

O seguinte método de ação usa os métodos auxiliares `Ok` e `NotFound`:

```
// GET: api/authors/search?namelike=th
[HttpGet("Search")]
public IActionResult Search(string namelike)
{
    var result = _authorRepository.GetByNameSubstring(namelike);
    if (!result.Any())
    {
        return NotFound(namelike);
    }
    return Ok(result);
}
```

Uma resposta formatada em JSON será retornada, a menos que outro formato tenha sido solicitado e o servidor possa retornar o formato solicitado. Use uma ferramenta como o [Fiddler](#) para criar uma solicitação que inclui um cabeçalho `Accept` e especifique outro formato. Nesse caso, se o servidor tiver um *formatador* que pode produzir uma resposta no formato solicitado, o resultado será retornado no formato preferencial do cliente.



Na captura de tela acima, o Fiddler Composer foi usado para gerar uma solicitação, especificando `Accept: application/xml`. Por padrão, o ASP.NET Core MVC dá suporte apenas ao JSON e, portanto, mesmo quando outro formato é especificado, o resultado retornado ainda é formatado em JSON. Você verá como adicionar outros formatadores na próxima seção.

As ações do controlador podem retornar POCOs (Objetos CLR Básicos); nesse caso, o ASP.NET Core MVC criará automaticamente um `ObjectResult` que encapsula o objeto. O cliente receberá o objeto serializado formatado (o formato JSON é o padrão; você pode configurar XML ou outros formatos). Se o objeto que está sendo retornado for `null`, a estrutura retornará uma resposta `204 No Content`.

Retornando um tipo de objeto:

```
// GET api/authors/ardalis
[HttpGet("{alias}")]
public Author Get(string alias)
{
    return _authorRepository.GetByAlias(alias);
}
```

Na amostra, uma solicitação para um alias de autor válido receberá uma resposta 200 OK com os dados do autor. Uma solicitação para um alias inválido receberá uma resposta 204 Sem Conteúdo. Capturas de tela mostrando a resposta nos formatos XML e JSON são mostradas abaixo.

### Processo de negociação de conteúdo

A *negociação* de conteúdo ocorre apenas se um cabeçalho `Accept` é exibido na solicitação. Quando uma solicitação contiver um cabeçalho `Accept`, a estrutura enumerará os tipos de mídia no cabeçalho `Accept` na ordem de preferência e tentará encontrar um formatador que pode produzir uma resposta em um dos formatos especificados pelo cabeçalho `Accept`. Caso nenhum formatador que pode atender à solicitação do cliente seja encontrado, a estrutura tentará encontrar o primeiro formatador que pode produzir uma resposta (a menos que o desenvolvedor tenha configurado a opção em `MvcOptions` para retornar 406 Não Aceitável). Se a solicitação

especificar XML, mas o formatador XML não tiver sido configurado, o formatador JSON será usado. Geralmente, se nenhum formatador que pode fornecer o formato solicitado for configurado, o primeiro formatador que pode formatar o objeto será usado. Se nenhum cabeçalho for fornecido, o primeiro formatador que pode manipular o objeto a ser retornado será usado para serializar a resposta. Nesse caso, não ocorre nenhuma negociação – o servidor determina qual formato será usado.

#### NOTE

Se o cabeçalho `Accept` contiver `/*`, o Cabeçalho será ignorado, a menos que `RespectBrowserAcceptHeader` seja definido como verdadeiro em `MvcOptions`.

## Navegadores e negociação de conteúdo

Ao contrário dos clientes de API típicos, os navegadores da Web tendem a fornecer cabeçalhos `Accept` que incluem uma ampla variedade de formatos, incluindo caracteres curinga. Por padrão, quando a estrutura detectar que a solicitação é proveniente de um navegador, ela ignorará o cabeçalho `Accept` e retornará o conteúdo no formato padrão configurado do aplicativo (JSON, a menos que outro formato seja configurado). Isso fornece uma experiência mais consistente no uso de diferentes navegadores para consumir APIs.

Se preferir que o aplicativo respeite os cabeçalhos `Accept` do navegador, defina isso como parte da configuração do MVC definindo `RespectBrowserAcceptHeader` como `true` no método `ConfigureServices` em `Startup.cs`.

```
services.AddMvc(options =>
{
    options.RespectBrowserAcceptHeader = true; // false by default
});
```

## Configurando formatadores

Se o aplicativo precisar dar suporte a outros formatos além do padrão de JSON, adicione pacotes NuGet e configure o MVC para dar suporte a eles. Há formatadores separados para entrada e saída. Os formatadores de entrada são usados pelo [Model Binding](#); os formatadores de saída são usados para formatar as respostas.

Também configure [Formatadores Personalizados](#).

### Adicionando suporte para o formato XML

Para adicionar suporte para a formatação XML, instale o pacote NuGet `Microsoft.AspNetCore.Mvc.Formatters.Xml`.

Adicione os `XmlSerializerFormatters` à configuração do MVC em `Startup.cs`:

```
services.AddMvc()
    .AddXmlSerializerFormatters();
```

Como alternativa, você pode adicionar apenas o formatador de saída:

```
services.AddMvc(options =>
{
    options.OutputFormatters.Add(new XmlSerializerOutputFormatter());
});
```

Essas duas abordagens serializarão os resultados usando `System.Xml.Serialization.XmlSerializer`. Se preferir, use o `System.Runtime.Serialization.DataContractSerializer` adicionando seu formatador associado:

```

services.AddMvc(options =>
{
    options.OutputFormatters.Add(new XmlDataContractSerializerOutputFormatter());
});

```

Depois de adicionar suporte para a formatação XML, os métodos do controlador deverão retornar o formato apropriado com base no cabeçalho `Accept` da solicitação, como demonstra este exemplo do Fiddler:

The screenshot shows the Fiddler interface with the following details:

- Request Headers:** GET http://localhost:9664/api/authors/ardalis HTTP/1.1, User-Agent: Fiddler, Host: localhost:9664, Accept: application/xml.
- Response Headers:** HTTP/1.1 200 OK, Content-Type: application/xml; charset=utf-8, Server: Kestrel, X-SourceFiles: =?UTF-8?B?, X-Powered-By: ASP.NET, Date: Wed, 01 Jun 2016 13:34:40 GMT, Content-Length: 166.
- Response Body:** XML representation of an Author object: <Author xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><Name>Steve Smith</Name><Twitter>ardalis</Twitter></Author>.

Na guia Inspecionadores, é possível ver que a solicitação GET Bruta foi feita com um conjunto de cabeçalhos `Accept: application/xml`. O painel de resposta mostra o cabeçalho `Content-Type: application/xml` e o objeto `Author` foi serializado em XML.

Use a guia Criador para modificar a solicitação para especificar `application/json` no cabeçalho `Accept`. Execute a solicitação e a resposta será formatada como JSON:

The screenshot shows the Fiddler interface with the following details:

- Request Headers:** GET http://localhost:9664/api/authors/ardalis HTTP/1.1, User-Agent: Fiddler, Host: localhost:9664, Accept: application/json.
- Response Headers:** HTTP/1.1 200 OK, Transfer-Encoding: chunked, Content-Type: application/json; charset=utf-8, Server: Kestrel, X-SourceFiles: =?UTF-8?B?, X-Powered-By: ASP.NET, Date: Wed, 01 Jun 2016 13:40:03 GMT.
- Response Body:** JSON array: [{"Name": "Steve Smith", "Twitter": "ardalis"}].

Nessa captura de tela, é possível ver a solicitação definir um cabeçalho `Accept: application/json` e a resposta especificar o mesmo como seu `Content-Type`. O objeto `Author` é mostrado no corpo da resposta, em formato JSON.

## Forçando um formato específico

Caso deseje restringir os formatos de resposta de uma ação específica, aplique o filtro `[Produces]`. O filtro `[Produces]` especifica os formatos de resposta de uma ação específica (ou o controlador). Como a maioria dos `Filtros`, isso pode ser aplicado no escopo da ação, do controlador ou global.

```
[Produces("application/json")]
public class AuthorsController
```

O filtro `[Produces]` forçará todas as ações em `AuthorsController` a retornar respostas formatadas em JSON, mesmo se outros formatadores foram configurados para o aplicativo e o cliente forneceu um cabeçalho `Accept` solicitando outro formato disponível. Consulte [Filtros](#) para saber mais, incluindo como aplicar filtros globalmente.

### Formatadores de casos especiais

Alguns casos especiais são implementados com formatadores internos. Por padrão, os tipos de retorno `string` serão formatados como `text/plain` (`text/html`, se solicitado por meio do cabeçalho `Accept`). Esse comportamento pode ser removido com a remoção do `TextOutputFormatter`. Remova formatadores no método `Configure` em `Startup.cs` (mostrado abaixo). Ações que têm um tipo de retorno de objeto de modelo retornarão uma resposta 204 Sem Conteúdo ao retornar `null`. Esse comportamento pode ser removido com a remoção do `HttpNoContentOutputFormatter`. O código a seguir remove o `TextOutputFormatter` e o `HttpNoContentOutputFormatter`.

```
services.AddMvc(options =>
{
    options.OutputFormatters.RemoveType<TextOutputFormatter>();
    options.OutputFormatters.RemoveType<HttpNoContentOutputFormatter>();
});
```

Sem o `TextOutputFormatter`, os tipos de retorno `string` retornam 406 Não Aceitável, por exemplo. Observe que se um formatador XML existir, ele formatará tipos de retorno `string` se o `TextOutputFormatter` for removido.

Sem o `HttpNoContentOutputFormatter`, os objetos nulos são formatados com o formatador configurado. Por exemplo, o formatador JSON apenas retornará uma resposta com um corpo `null`, enquanto o formatador XML retornará um elemento XML vazio com o atributo `xsi:nil="true"` definido.

## Mapeamentos de URL do formato da resposta

Os clientes podem solicitar um formato específico como parte da URL, como na cadeia de caracteres de consulta ou parte do caminho, ou usando uma extensão de arquivo específica a um formato, como `.xml` ou `.json`. O mapeamento do caminho da solicitação deve ser especificado na rota que está sendo usada pela API. Por exemplo:

```
[FormatFilter]
public class ProductsController
{
    [Route("[controller]/[action]/[id].{format?}")]
    public Product GetById(int id)
```

Essa rota permitirá que o formato solicitado seja especificado como uma extensão de arquivo opcional. O atributo `[FormatFilter]` verifica a existência do valor de formato no `RouteData` e mapeará o formato da resposta para o formatador adequado quando a resposta for criada.

ROTA	FORMATADOR
<code>/products/.GetById/5</code>	O formatador de saída padrão
<code>/products/.GetById/5.json</code>	O formatador JSON (se configurado)

ROTA	FORMATADOR
/products/GetById/5.xml	O formatador XML (se configurado)

# Formatadores personalizados na API Web ASP.NET Core

10/01/2019 • 8 minutes to read • [Edit Online](#)

Por [Tom Dykstra](#)

O ASP.NET Core MVC tem suporte interno para troca de dados em APIs Web usando XML ou JSON. Este artigo mostra como adicionar suporte para formatos adicionais criando formatadores personalizados.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Quando usar formatadores personalizados

Use um formatador personalizado quando quiser que o processo de [negociação de conteúdo](#) dê suporte a um tipo de conteúdo que não tem suporte dos formatadores internos (JSON e XML).

Por exemplo, se alguns dos clientes de sua API Web puderem usar o formato [Protobuf](#), talvez você queira usar Protobuf com esses clientes porque é mais eficiente. Ou talvez você queira que sua API Web envie endereços e nomes de contato no formato [vCard](#), um formato usado normalmente para troca de dados de contato. O aplicativo de exemplo fornecido com este artigo implementa um formatador vCard simples.

## Visão geral de como usar um formatador personalizado

Estas são as etapas para criar e usar um formatador personalizado:

- Criar uma classe de formatador de saída se quiser serializar os dados a serem enviados ao cliente.
- Criar uma classe de formatador de entrada se quiser desserializar os dados recebidos do cliente.
- Adicionar instâncias de seus formatadores às coleções `InputFormatters` e `OutputFormatters` em [MvcOptions](#).

As seções a seguir fornecem diretrizes e exemplos de código para cada uma dessas etapas.

## Como criar uma classe de formatador personalizado

Para criar um formatador:

- Derive a classe da classe base apropriada.
- Especifique codificações e tipos de mídia válidos no construtor.
- Substitua os métodos `CanReadType` / `CanWriteType`
- Substitua os métodos `ReadRequestBodyAsync` / `WriteResponseBodyAsync`

### Derivar da classe base apropriada

Para tipos de mídia de texto (por exemplo, vCard), derive da classe base [TextInputFormatter](#) ou [TextOutputFormatter](#).

```
public class VcardOutputFormatter : TextOutputFormatter
```

Para obter um exemplo de formatador de entrada, consulte o [aplicativo de exemplo](#).

Para tipos binários, derive da classe base [InputFormatter](#) ou [OutputFormatter](#).

### Especifique codificações e tipos de mídia válidos

No construtor, especifique codificações e tipos de mídia válidos adicionando-os às coleções `SupportedMediaTypes` e `SupportedEncodings`.

```
public VcardOutputFormatter()
{
    SupportedMediaTypes.Add(MediaTypeHeaderValue.Parse("text/vcard"));

    SupportedEncodings.Add(Encoding.UTF8);
    SupportedEncodings.Add(Encoding.Unicode);
}
```

Para obter um exemplo de formatador de entrada, consulte o [aplicativo de exemplo](#).

#### NOTE

Não é possível fazer a injeção de dependência de construtor em uma classe de formatador. Por exemplo, não é possível obter um agente adicionando um parâmetro de agente ao construtor. Para acessar serviços, você precisa usar o objeto de contexto que é passado para seus métodos. O exemplo de código [abaixo](#) mostra como isso é feito.

### Substituir `CanReadType`/`CanWriteType`

Especifique o tipo no qual desserializar ou do qual serializar substituindo os métodos `CanReadType` ou `CanWriteType`. Por exemplo, talvez você só possa criar texto de vCard de um tipo `Contact` e vice-versa.

```
protected override bool CanWriteType(Type type)
{
    if (typeof(Contact).IsAssignableFrom(type)
        || typeof(IEnumerable<Contact>).IsAssignableFrom(type))
    {
        return base.CanWriteType(type);
    }
    return false;
}
```

Para obter um exemplo de formatador de entrada, consulte o [aplicativo de exemplo](#).

#### O método `CanWriteResult`

Em alguns cenários, você precisa substituir `CanWriteResult` em vez de `CanWriteType`. Use `CanWriteResult` se as condições a seguir forem verdadeiras:

- O método de ação retorna uma classe de modelo.
- Há classes derivadas que podem ser retornadas em tempo de execução.
- Você precisa saber em tempo de execução qual classe derivada foi retornada pela ação.

Por exemplo, suponha que sua assinatura do método de ação retorne um tipo `Person`, mas ele pode retornar um tipo `Student` ou `Instructor` que deriva de `Person`. Se você quiser que o formatador trate apenas de objetos `Student`, verifique o tipo de `Objeto` no objeto de contexto fornecido ao método `CanWriteResult`. Observe que não é necessário usar `CanWriteResult` quando o método de ação retorna `IActionResult`; nesse caso, o método `CanWriteType` recebe o tipo de tempo de execução.

### Substituir `ReadRequestBodyAsync`/`WriteResponseBodyAsync`

Você faz o trabalho real de desserialização ou serialização em `ReadRequestBodyAsync` ou `WriteResponseBodyAsync`. As linhas destacadas no exemplo a seguir mostram como obter serviços do contêiner de injeção de dependência (não é possível obtê-los dos parâmetros do construtor).

```

public override Task WriteResponseBodyAsync(OutputFormatterWriteContext context, Encoding selectedEncoding)
{
    IServiceProvider serviceProvider = context.HttpContext.RequestServices;
    var logger = serviceProvider.GetService(typeof(ILogger<VcardOutputFormatter>)) as ILogger;

    var response = context.HttpContext.Response;

    var buffer = new StringBuilder();
    if (context.Object is IEnumerable<Contact>)
    {
        foreach (Contact contact in context.Object as IEnumerable<Contact>)
        {
            FormatVcard(buffer, contact, logger);
        }
    }
    else
    {
        var contact = context.Object as Contact;
        FormatVcard(buffer, contact, logger);
    }
    return response.WriteAsync(buffer.ToString());
}

private static void FormatVcard(StringBuilder buffer, Contact contact, ILogger logger)
{
    buffer.AppendLine("BEGIN:VCARD");
    buffer.AppendLine("VERSION:2.1");
    buffer.AppendFormat($"N:{contact.LastName};{contact.FirstName}\r\n");
    buffer.AppendFormat($"FN:{contact.FirstName} {contact.LastName}\r\n");
    buffer.AppendFormat($"UID:{contact.ID}\r\n");
    buffer.AppendLine("END:VCARD");
    logger.LogInformation($"Writing {contact.FirstName} {contact.LastName}");
}

```

Para obter um exemplo de formatador de entrada, consulte o [aplicativo de exemplo](#).

## Como configurar o MVC para usar um formatador personalizado

Para usar um formatador personalizado, adicione uma instância da classe de formatador à coleção

`InputFormatters` OU `OutputFormatters`.

```

services.AddMvc(options =>
{
    options.InputFormatters.Insert(0, new VcardInputFormatter());
    options.OutputFormatters.Insert(0, new VcardOutputFormatter());
});

```

Formatadores são avaliados na ordem em que você os insere. O primeiro deles tem precedência.

## Próximas etapas

- [Código de exemplo do formatador de texto sem formatação no GitHub](#).
- [Aplicativo de exemplo para este documento](#), que implementa formatadores de entrada e saída simples de vCard. O aplicativo lê e grava vCards parecidos com o exemplo a seguir:

```
BEGIN:VCARD  
VERSION:2.1  
N:Davolio;Nancy  
FN:Nancy Davolio  
UID:20293482-9240-4d68-b475-325df4a83728  
END:VCARD
```

Para ver a saída do vCard, execute o aplicativo e envie uma solicitação Get com o cabeçalho de aceitação "texto/vcard" para <http://localhost:63313/api/contacts/> (ao executar com Visual Studio) ou <http://localhost:5000/api/contacts/> (ao executar da linha de comando).

Para adicionar um vCard à coleção de contatos na memória, envie uma solicitação Post para a mesma URL, com cabeçalho Content-Type "texto/vcard" e com o texto do vCard no corpo, formatado como o exemplo acima.

# Usar os analisadores da API Web

10/01/2019 • 3 minutes to read • [Edit Online](#)

O ASP.NET Core 2.2 (e posterior) inclui o pacote NuGet [Microsoft.AspNetCore.Mvc.Api.Analyzers](#) contendo analisadores para APIs Web. Os analisadores trabalham com controladores anotados com [ApiControllerAttribute](#) enquanto fazem a compilação em [convenções de API](#).

## Instalação do pacote

`Microsoft.AspNetCore.Mvc.Api.Analyzers` pode ser adicionado com uma das seguintes abordagens:

- [Visual Studio](#)
- [Visual Studio para Mac](#)
- [Visual Studio Code](#)
- [CLI do .NET Core](#)
- Da janela **Console do Gerenciador de Pacotes**:
  - Acesse **Exibição > Outras Janelas > Console do Gerenciador de Pacotes**.
  - Navegue até o diretório no qual o arquivo *ApiConventions.csproj* está localizado.
  - Execute o seguinte comando:

```
Install-Package Microsoft.AspNetCore.Mvc.Api.Analyzers
```
- Da caixa de diálogo **Gerenciar Pacotes NuGet**:
  - Clique com o botão direito do mouse no projeto em **Gerenciador de Soluções > Gerenciar Pacotes NuGet**.
  - Defina a **Origem do pacote** como "nuget.org".
  - Insira "Microsoft.AspNetCore.Mvc.Api.Analyzers" na caixa de pesquisa.
  - Selecione o pacote "Microsoft.AspNetCore.Mvc.Api.Analyzers" na guia **Procurar** e clique em **Instalar**.

## Analisadores para convenções de API

Documentos de OpenAPI contêm códigos de status e tipos de resposta que uma ação pode retornar. No ASP.NET Core MVC, atributos como [ProducesResponseTypeAttribute](#) e [ProducesAttribute](#) são usados para documentar uma ação. [Páginas de ajuda da API Web ASP.NET Core com o Swagger/OpenAPI](#) apresenta mais detalhes sobre como documentar sua API.

Um dos analisadores no pacote inspeciona controladores anotados com [ApiControllerAttribute](#) e identifica ações que não documentam totalmente as respostas. Considere o exemplo a seguir:

```
// GET api/contacts/{guid}
[HttpGet("{id}", Name = "GetById")]
[ProducesResponseType(typeof(Contact), StatusCodes.Status200OK)]
public IActionResult Get(string id)
{
    var contact = _contacts.Get(id);

    if (contact == null)
    {
        return NotFound();
    }

    return Ok(contact);
}
```

A ação precedente documenta o tipo de retorno com êxito do HTTP 200, mas não documenta o código de status com falha do HTTP 404. O analisador relata a documentação ausente para o código de status HTTP 404 como um aviso. É fornecida uma opção para consertar o problema.

## Recursos adicionais

- [Usar convenções de API Web](#)
- [Páginas de ajuda da API Web ASP.NET Core com o Swagger/OpenAPI](#)
- [Anotação com o atributo ApiController](#)

# Usar convenções de API Web

06/02/2019 • 6 minutes to read • [Edit Online](#)

Por [Pranav Krishnamoorthy](#) e [Scott Addie](#)

O ASP.NET Core 2.2 (e posterior) inclui uma forma de extrair a [documentação da API](#) comum e aplicá-la a várias ações, controladores ou a todos os controladores em um assembly. As convenções de API Web são um substituto para ambientar as ações individuais com [\[ProducesResponseType\]](#).

Uma convenção permite que você:

- Defina os tipos de retorno mais comuns e códigos de status retornados de um tipo de ação específico.
- Identifica as ações que desviam do padrão definido.

O ASP.NET Core MVC 2.2 (e posterior) inclui um conjunto de convenções padrão em [Microsoft.AspNetCore.Mvc.DefaultApiConventions](#). As convenções são baseadas no controlador ([ValuesController.cs](#)) fornecido no modelo de projeto da [API](#) do ASP.NET Core. Se suas ações seguem o padrão no modelo, você deve ter êxito ao usar as convenções padrão. Se as convenções padrão não atenderem às suas necessidades, consulte [Criar convenções de API Web](#).

No tempo de execução, [Microsoft.AspNetCore.Mvc.ApiExplorer](#) reconhece as convenções. [ApiExplorer](#) é a abstração do MVC para comunicação com geradores de documento da [OpenAPI](#) (também conhecida como Swagger). Os atributos da convenção aplicada são associados a uma ação e estão incluídos na documentação da OpenAPI da ação. Os [Analizadores de API](#) também reconhecem as convenções. Se a ação for não convencional (por exemplo, ela retorna um código de status que não está documentado pela convenção aplicada), um aviso incentivará você a fazer a documentação do código de status.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Aplicar convenções de API Web

As convenções não fazem composição. Cada ação pode ser associada a exatamente uma convenção. As convenções mais específicas têm precedência sobre as menos específicas. A seleção é não determinística quando duas ou mais convenções da mesma prioridade se aplicam a uma ação. As seguintes opções existem para aplicar uma convenção a uma ação, da mais específica à menos específica:

1. [Microsoft.AspNetCore.Mvc.ApiConventionMethodAttribute](#) — aplica-se a ações individuais e especifica o tipo de convenção e o método de convenção que se aplica.

No exemplo a seguir, o método de convenção [Microsoft.AspNetCore.Mvc.DefaultApiConventions.Put](#) do tipo de convenção padrão é aplicado à ação [Update](#):

```
// PUT api/contactsconvention/{guid}
[HttpPut("{id}")]
[ApiConventionMethod(typeof(DefaultApiConventions),
    nameof(DefaultApiConventions.Put))]
public IActionResult Update(string id, Contact contact)
{
    var contactToUpdate = _contacts.Get(id);

    if (contactToUpdate == null)
    {
        return NotFound();
    }

    _contacts.Update(contact);

    return NoContent();
}
```

o método de convenção `Microsoft.AspNetCore.Mvc.DefaultApiConventions.Put` aplica os seguintes atributos à ação:

```
[ProducesDefaultResponseType]
[ProducesResponseType(204)]
[ProducesResponseType(404)]
[ProducesResponseType(400)]
```

Para saber mais sobre `[ProducesDefaultResponseType]`, confira [Resposta padrão](#).

1. `Microsoft.AspNetCore.Mvc.ApiConventionTypeAttribute` aplicado a um controlador —. Aplica-se o tipo de convenção especificada a todas as ações no controlador. Um método de convenção é decorado com dicas que determinam as ações às quais o método de convenção se aplica. Para obter mais informações sobre dicas, consulte [Criar convenções da API Web](#).

No exemplo a seguir, o conjunto padrão de convenções é aplicado a todas as ações no `ContactsConventionController`:

```
[ApiController]
[ApiConventionType(typeof(DefaultApiConventions))]
[Route("api/[controller]")]
public class ContactsConventionController : ControllerBase
{
```

2. `Microsoft.AspNetCore.Mvc.ApiConventionTypeAttribute` aplicado a um assembly —. Aplica-se o tipo de convenção especificada a todos os controladores no assembly atual. Como recomendação, aplique atributos no nível do assembly ao arquivo `Startup.cs`.

No exemplo a seguir, o conjunto padrão de convenções é aplicado a todos os controladores no assembly:

```
[assembly: ApiConventionType(typeof(DefaultApiConventions))]
namespace ApiConventions
{
    public class Startup
    {
```

## Criar convenções de API Web

Se as convenções padrão da API não atenderem às suas necessidades, crie suas próprias convenções. Uma

convenção é:

- um tipo estático com métodos.
- Com capacidade de definir [tipos de resposta](#) e [requisitos de nomenclatura](#) em ações.

## Tipos de resposta

Esses métodos são anotados com atributos `[ProducesResponseType]` ou `[ProducesDefaultResponseType]`. Por exemplo:

```
public static class MyAppConventions
{
    [ProducesResponseType(200)]
    [ProducesResponseType(404)]
    public static void Find(int id)
    {
    }
}
```

Se atributos de metadados mais específicos estão ausentes, a aplicação dessa convenção a um assembly impõe que:

- O método de convenção se aplica a qualquer ação denominada `Find`.
- Um parâmetro denominado `id` está presente na ação `Find`.

## Requisitos de nomenclatura

Os atributos `[ApiConventionNameMatch]` e `[ApiConventionTypeMatch]` podem ser aplicados ao método de convenção que determina as ações às quais eles são aplicáveis. Por exemplo:

```
[ProducesResponseType(200)]
[ProducesResponseType(404)]
[ApiConventionNameMatch(ApiConventionNameMatchBehavior.Prefix)]
public static void Find(
    [ApiConventionNameMatch(ApiConventionNameMatchBehavior.Suffix)]
    int id)
{ }
```

No exemplo anterior:

- A opção `Microsoft.AspNetCore.Mvc.ApiExplorer.ApiConventionNameMatchBehavior.Prefix` aplicada ao método indica que a convenção corresponde a qualquer ação desde que seja prefixada com "Find". Exemplos de ações correspondentes incluem `Find`, `FindPet` e `FindByID`.
- O `Microsoft.AspNetCore.Mvc.ApiExplorer.ApiConventionNameMatchBehavior.Suffix` aplicado ao parâmetro indica que a convenção corresponde a métodos com exatamente um parâmetro encerrando no identificador do sufixo. Exemplos incluem parâmetros como `id` ou `petId`. `ApiConventionTypeMatch` pode ser aplicado da mesma forma aos tipos para restringir o tipo do parâmetro. Um argumento `params[]` indica parâmetros restantes que não precisam ser explicitamente correspondidos.

## Recursos adicionais

- [Usar os analisadores da API Web](#)
- [Páginas de ajuda da API Web ASP.NET Core com o Swagger/OpenAPI](#)

# Introdução ao SignalR do ASP.NET Core

24/01/2019 • 4 minutes to read • [Edit Online](#)

## O que é o SignalR?

SignalR do ASP.NET Core é uma biblioteca de software livre que simplifica a adição da funcionalidade da web em tempo real aos aplicativos. A funcionalidade da web em tempo real permite que o código do lado do servidor para conteúdo de envio por push aos clientes instantaneamente.

Bons candidatos para o SignalR:

- Aplicativos que exigem atualizações de alta frequência do servidor. Exemplos são jogos, redes sociais, de votação, leilão, mapas e aplicativos GPS.
- Os painéis e monitoramento de aplicativos. Exemplos incluem painéis da empresa, atualizações de vendas instantâneas, ou alertas de viagem.
- Aplicativos de colaboração. Quadro de comunicações aplicativos e software de reunião de equipe são exemplos de aplicativos de colaboração.
- Aplicativos que exigem as notificações. Redes sociais, email, bate-papo, jogos, alertas de viagem e muitos outros aplicativos usam notificações.

O SignalR fornece uma API para a criação de servidor para cliente [chamadas de procedimento remoto \(RPC\)](#). As RPCs chamam funções JavaScript em clientes do código do lado do servidor do .NET Core.

Aqui estão alguns recursos do SignalR para ASP.NET Core:

- Lida com gerenciamento de conexão automaticamente.
- Envia mensagens para todos os clientes conectados simultaneamente. Por exemplo, uma sala de bate-papo.
- Envia mensagens para clientes específicos ou grupos de clientes.
- É dimensionada para lidar com o aumento de tráfego.

A fonte é hospedada em um [SignalR repositório no GitHub](#).

## Transportes

O SignalR dá suporte a várias técnicas para manipular as comunicações em tempo real:

- [WebSockets](#)
- Eventos enviados pelo servidor
- Sondagem longa

O SignalR escolhe automaticamente o melhor método de transporte que está dentro de recursos do servidor e cliente.

## Hubs

Usa o SignalR *hubs* para comunicação entre clientes e servidores.

Um hub é um pipeline de alto nível que permite que um cliente e servidor chamar métodos em si. O SignalR manipula a expedição entre limites de máquina automaticamente, permitindo que os clientes chamar métodos no servidor e vice-versa. Você pode passar parâmetros fortemente tipados para métodos, que habilita a associação de modelo. O SignalR fornece dois protocolos de hub interno: um protocolo de texto com base em JSON e um protocolo binário, com base em [MessagePack](#). MessagePack geralmente cria mensagens menores em

comparação comparadas JSON. Devem dar suporte a navegadores mais antigos [nível XHR 2](#) para fornecer suporte de protocolo MessagePack.

Os hubs de chamar o código do lado do cliente enviando mensagens que contêm o nome e parâmetros do método do lado do cliente. Os objetos enviados como parâmetros de método são desserializados usando o protocolo configurado. O cliente tenta corresponder o nome a um método no código do lado do cliente. Quando o cliente encontra uma correspondência, ele chama o método e passa a ele os dados de parâmetro desserializado.

## Recursos adicionais

- [Introdução ao SignalR para ASP.NET Core](#)
- [Plataformas com suporte](#)
- [Hubs](#)
- [Cliente JavaScript](#)

# Plataformas com suporte do SignalR do ASP.NET Core

25/01/2019 • 2 minutes to read • [Edit Online](#)

## Requisitos de sistema do servidor

SignalR para ASP.NET Core dá suporte a qualquer plataforma de servidor que dá suporte ao ASP.NET Core.

## Cliente JavaScript

O [cliente JavaScript](#) é executado no NodeJS 8 e versões posteriores e os seguintes navegadores:

NAVIGADOR	VERSÃO
Microsoft Edge	atual
Mozilla Firefox	atual
Google Chrome; inclui o Android	atual
Safari; inclui o iOS	atual
Microsoft Internet Explorer	11

## Cliente .NET

O [cliente .NET](#) é executado em qualquer plataforma com suporte pelo ASP.NET Core. Por exemplo, [desenvolvedores Xamarin podem usar o SignalR](#) para a criação de aplicativos Android usando o xamarin. Android 8.4.0.1 e posterior e aplicativos iOS usando xamarin. IOS 11.14.0.4 e versões posteriores.

Se o servidor executa o IIS, o transporte de WebSockets requer o IIS 8.0 ou superior no Windows Server 2012 ou superior. Outros transportes têm suporte em todas as plataformas.

## Cliente de Java

O [cliente Java](#) dá suporte a Java 8 e versões posteriores.

## Não há suporte para clientes

Os clientes a seguir estão disponíveis, mas são experimentais ou não oficial. Eles não têm suporte no momento e nunca podem ser.

- [Cliente C++](#)
- [Cliente SWIFT](#)

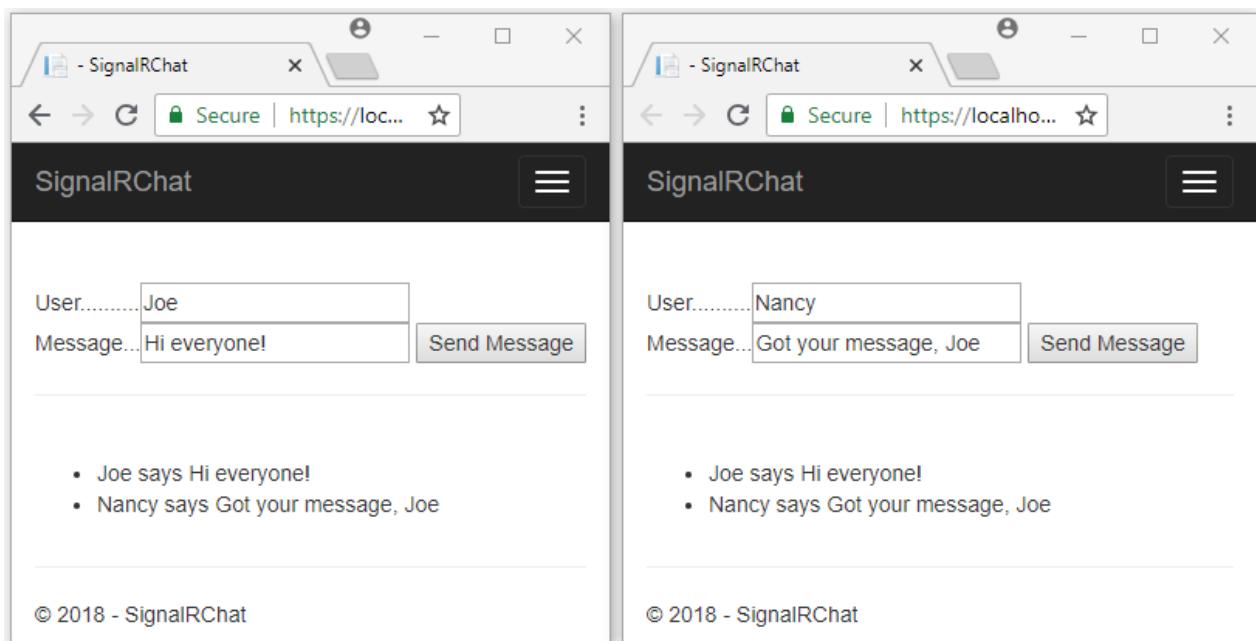
# Tutorial: Introdução ao SignalR para ASP.NET Core

28/01/2019 • 11 minutes to read • [Edit Online](#)

Este tutorial ensina as noções básicas da criação de um aplicativo em tempo real usando o SignalR. Você aprenderá como:

- Crie um projeto Web.
- Adicionar uma biblioteca de clientes do SignalR.
- Criar um hub do SignalR.
- Configurar o projeto para usar o SignalR.
- Adicione o código que envia mensagens de qualquer cliente para todos os clientes conectados.

No final, você terá um aplicativo de chat funcionando:



[Exibir ou baixar um código de exemplo \(como baixar\).](#)

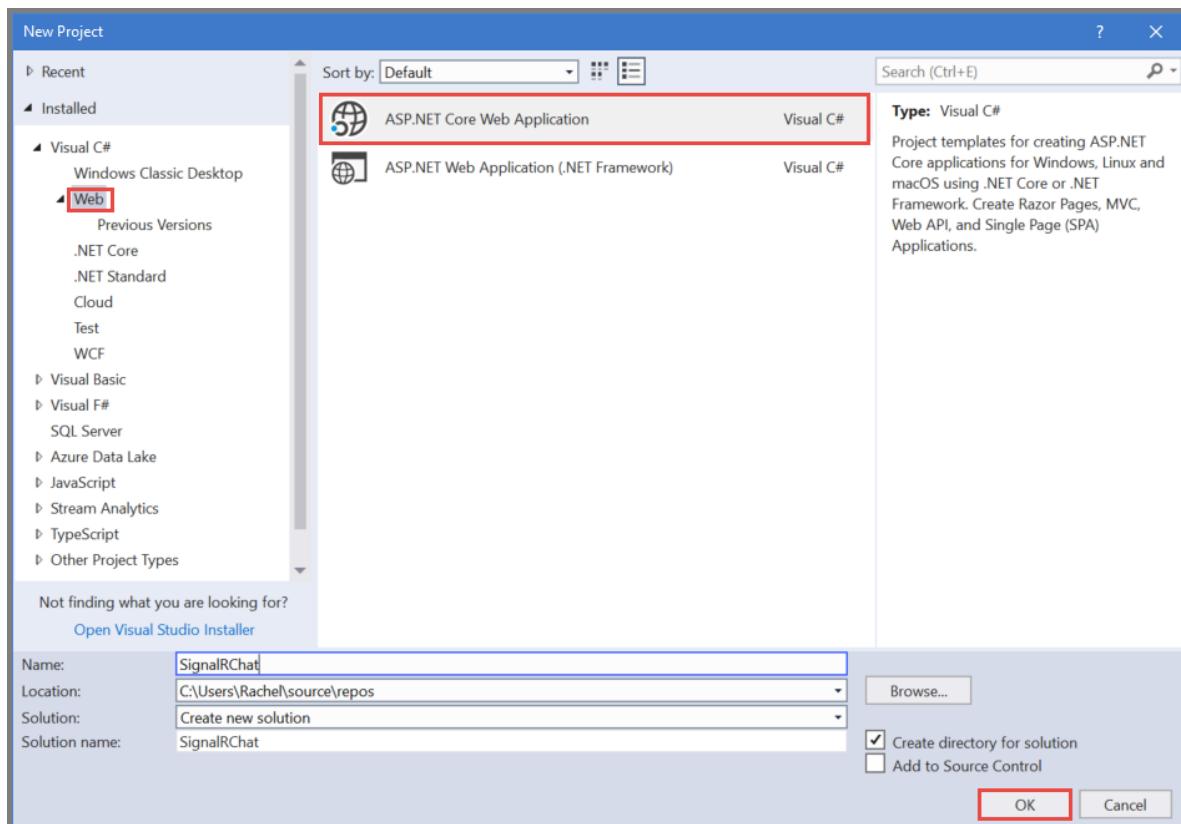
## Pré-requisitos

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- [Visual Studio 2017 versão 15.9 ou posterior](#) com a carga de trabalho **ASP.NET e desenvolvimento para a Web**
- [SDK 2.2 ou posterior do .NET Core](#)

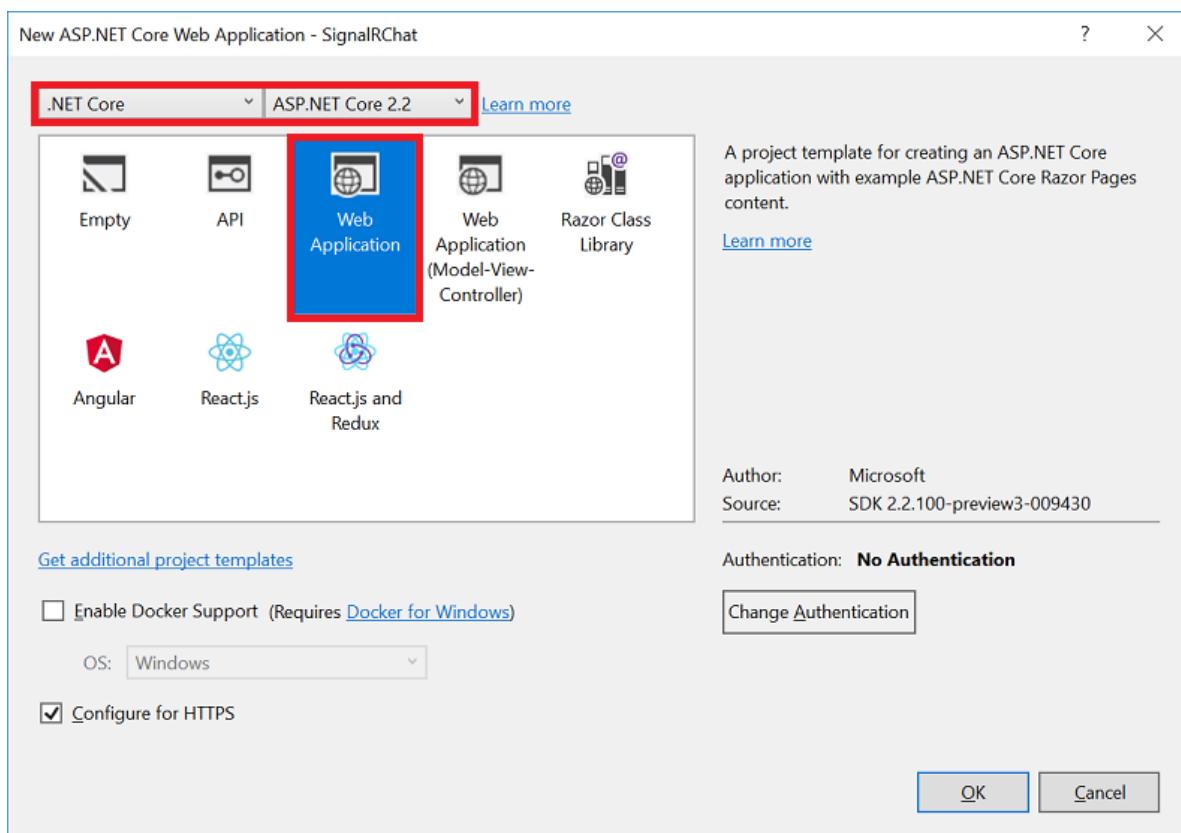
## Criar um projeto Web

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- No menu, selecione **Arquivo > Novo Projeto**.

- Na caixa de diálogo **Novo Projeto**, selecione **Instalado > Visual C# > Web > Aplicativo Web ASP.NET Core**. Dê ao projeto o nome de *SignalRChat*.



- Selecione **Aplicativo Web** para criar um projeto que usa Razor Pages.
- Selecione uma estrutura de destino do **.NET Core**, selecione **ASP.NET Core 2.2** e clique em **OK**.

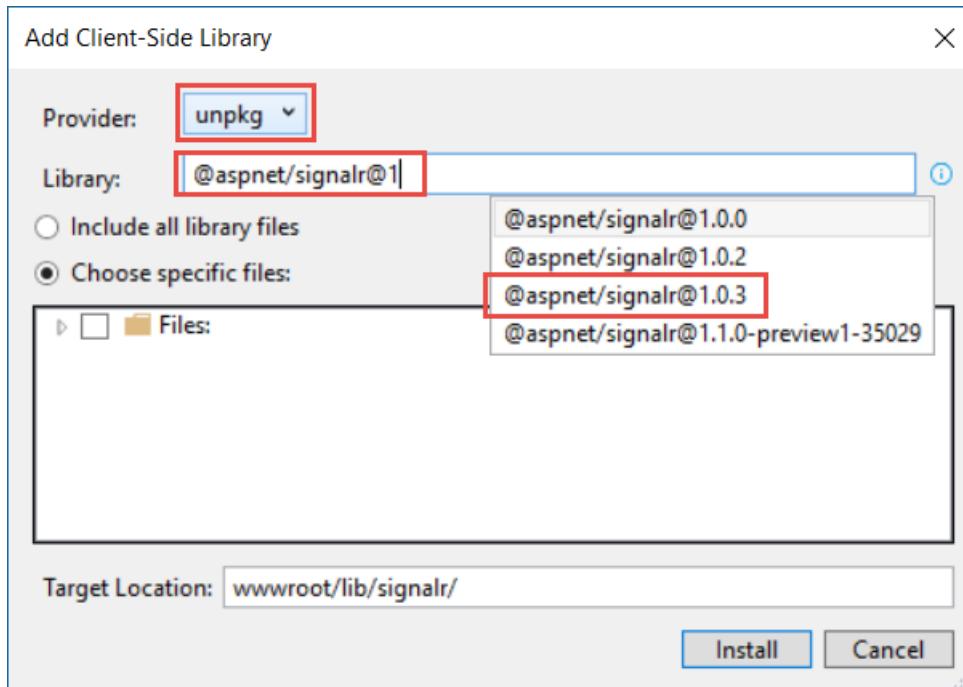


## Adicionar a biblioteca de clientes do SignalR

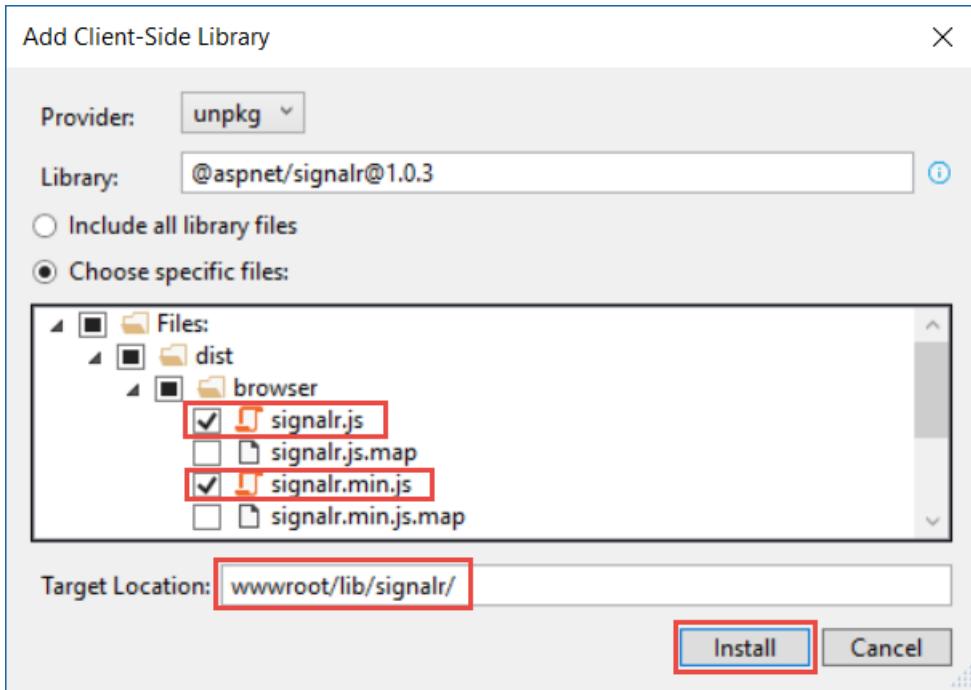
A biblioteca do servidor SignalR está incluída no metapacote `Microsoft.AspNetCore.App`. A biblioteca de clientes

do JavaScript não é incluída automaticamente no projeto. Neste tutorial, você usará o LibMan (Library Manager) para obter a biblioteca de clientes de `unpkg`. `unpkg` é uma CDN (rede de distribuição de conteúdo) que pode distribuir qualquer conteúdo do npm, o gerenciador de pacotes do Node.js.

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- No **Gerenciador de Soluções**, clique com o botão direito do mouse no projeto e selecione **Adicionar > Biblioteca do Lado do Cliente**.
- Na caixa de diálogo **Adicionar Biblioteca do Lado do Cliente**, para **Provedor**, selecione **unpkg**.
- Para **Biblioteca**, insira `@aspnet/signalr@1` e selecione a versão mais recente que não seja uma versão prévia.



- Selecione **Escolher arquivos específicos**, expanda a pasta *distribuidor/navegador* e selecione `signalr.js` e `signalr:min.js`.
- Defina **Localização de Destino** como `wwwroot/lib/signalr/` e selecione **Instalar**.



O LibMan cria uma pasta `wwwroot/lib/signalr` e copia os arquivos selecionados para ela.

## Criar um hub do SignalR

Um *hub* é uma classe que funciona como um pipeline de alto nível que lida com a comunicação entre cliente e servidor.

- Na pasta do projeto SignalRChat, crie uma pasta *Hubs*.
- Na pasta *Hubs*, crie um arquivo *ChatHub.cs* com o código a seguir:

```
using Microsoft.AspNetCore.SignalR;
using System.Threading.Tasks;

namespace SignalRChat.Hubs
{
    public class ChatHub : Hub
    {
        public async Task SendMessage(string user, string message)
        {
            await Clients.All.SendAsync("ReceiveMessage", user, message);
        }
    }
}
```

A classe `ChatHub` é herda da classe `Hub` do SignalR. A classe `Hub` gerencia conexões, grupos e sistemas de mensagens.

O método `SendMessage` pode ser chamado por um cliente conectado para enviar uma mensagem a todos os clientes. O código cliente do JavaScript que chama o método é mostrado posteriormente no tutorial. O código do SignalR é assíncrono para fornecer o máximo de escalabilidade.

## Configurar o SignalR

O servidor do SignalR precisa ser configurado para passar solicitações do SignalR ao SignalR.

- Adicione o seguinte código realçado ao arquivo *Startup.cs*.

```

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using SignalRChat.Hubs;

namespace SignalRChat
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.Configure<CookiePolicyOptions>(options =>
            {
                // This lambda determines whether user consent for non-essential cookies is needed for
                a given request.
                options.CheckConsentNeeded = context => true;
                options.MinimumSameSitePolicy = SameSiteMode.None;
            });

            services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

            services.AddSignalR();
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request
        pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Error");
                app.UseHsts();
            }

            app.UseHttpsRedirection();
            app.UseStaticFiles();
            app.UseCookiePolicy();
            app.UseSignalR(routes =>
            {
                routes.MapHub<ChatHub>("/chatHub");
            });
            app.UseMvc();
        }
    }
}

```

Essas alterações adicionam o SignalR ao sistema de injeção de dependência e ao pipeline do middleware do ASP.NET Core.

## Adicionar o código de cliente do SignalR

- Substitua o conteúdo *Pages\Index.cshtml* pelo código a seguir:

```
@page
<div class="container">
    <div class="row">&nbsp;</div>
    <div class="row">
        <div class="col-6">&nbsp;</div>
        <div class="col-6">
            User.....<input type="text" id="userInput" />
            <br />
            Message...<input type="text" id="messageInput" />
            <input type="button" id="sendButton" value="Send Message" />
        </div>
    </div>
    <div class="row">
        <div class="col-12">
            <hr />
        </div>
    </div>
    <div class="row">
        <div class="col-6">&nbsp;</div>
        <div class="col-6">
            <ul id="messagesList"></ul>
        </div>
    </div>
</div>
<script src="~/lib/signalr/dist/browser/signalr.js"></script>
<script src="~/js/chat.js"></script>
```

O código anterior:

- Cria as caixas de texto para o nome e a mensagem de texto e um botão Enviar.
  - Cria uma lista com `id="messagesList"` para exibir as mensagens recebidas do hub do SignalR.
  - Inclui referências de script ao SignalR e ao código do aplicativo *chat.js* que você criará na próxima etapa.
- Na pasta *wwwroot/js*, crie um arquivo *chat.js* com o código a seguir:

```

"use strict";

var connection = new signalR.HubConnectionBuilder().withUrl("/chatHub").build();

//Disable send button until connection is established
document.getElementById("sendButton").disabled = true;

connection.on("ReceiveMessage", function (user, message) {
    var msg = message.replace(/&/g, "&").replace(/</g, "<").replace(/>/g, ">");
    var encodedMsg = user + " says " + msg;
    var li = document.createElement("li");
    li.textContent = encodedMsg;
    document.getElementById("messagesList").appendChild(li);
});

connection.start().then(function(){
    document.getElementById("sendButton").disabled = false;
}).catch(function (err) {
    return console.error(err.toString());
});

document.getElementById("sendButton").addEventListener("click", function (event) {
    var user = document.getElementById("userInput").value;
    var message = document.getElementById("messageInput").value;
    connection.invoke("SendMessage", user, message).catch(function (err) {
        return console.error(err.toString());
    });
    event.preventDefault();
});

```

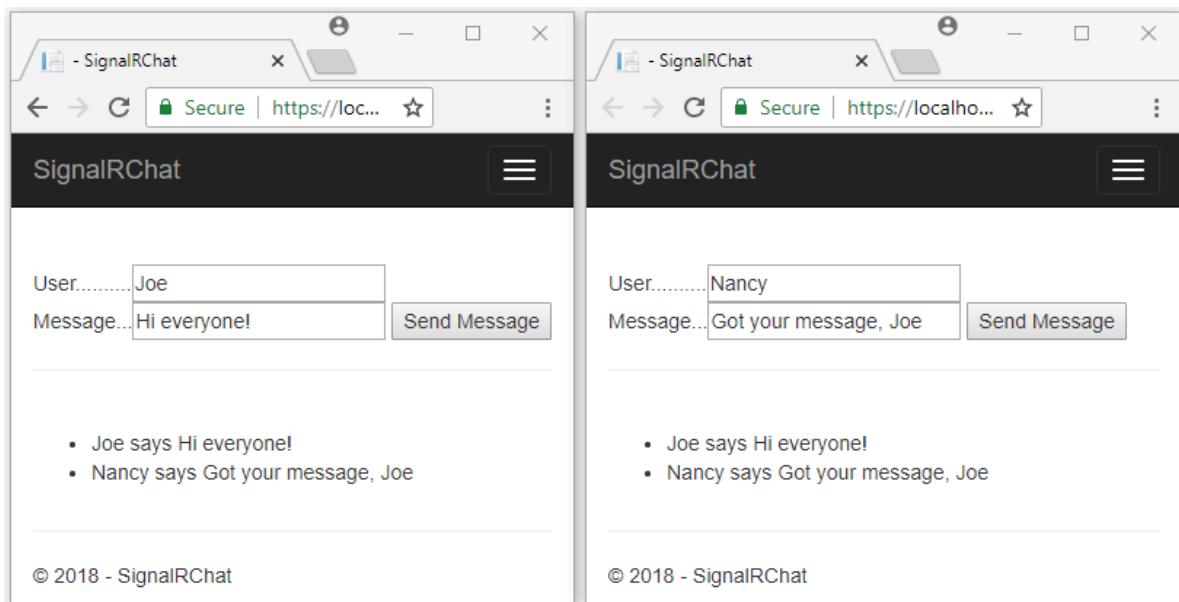
O código anterior:

- Cria e inicia uma conexão.
- Adiciona no botão Enviar um manipulador que envia mensagens ao hub.
- Adiciona no objeto de conexão um manipulador que recebe mensagens do hub e as adiciona à lista.

## Executar o aplicativo

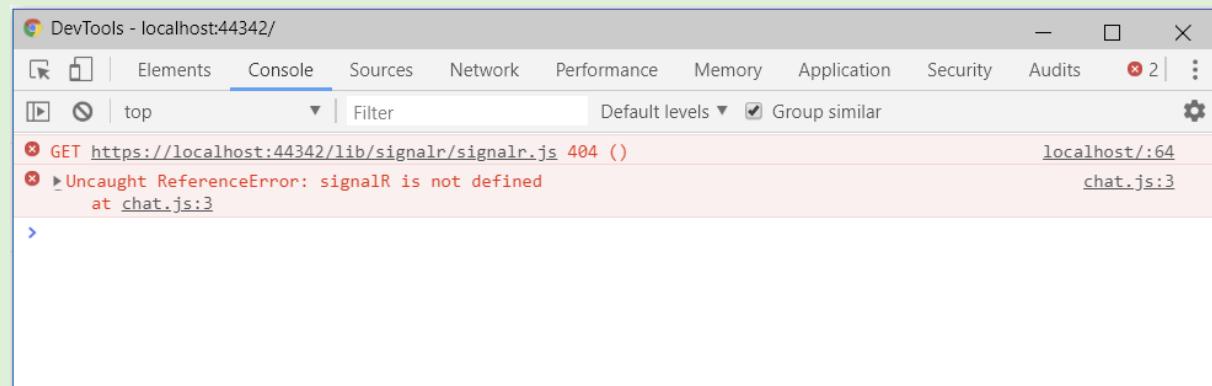
- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- Pressione **CTRL + F5** para executar o aplicativo sem depuração.
- Copie a URL da barra de endereços, abra outra instância ou guia do navegador e cole a URL na barra de endereços.
- Escolha qualquer navegador, insira um nome e uma mensagem e selecione o botão **Enviar Mensagem**.

O nome e a mensagem são exibidos em ambas as páginas instantaneamente.



#### TIP

Se o aplicativo não funcionar, abra as ferramentas para desenvolvedores do navegador (F12) e acesse o console. Você pode encontrar erros relacionados ao código HTML e JavaScript. Por exemplo, suponha que você coloque `signalr.js` em uma pasta diferente daquela direcionada. Nesse caso, a referência a esse arquivo não funcionará e ocorrerá um erro 404 no console.



## Próximas etapas

Neste tutorial, você aprendeu como:

- Criar um projeto de aplicativo Web.
- Adicionar uma biblioteca de clientes do SignalR.
- Criar um hub do SignalR.
- Configurar o projeto para usar o SignalR.
- Adicionar o código que usa o hub para enviar mensagens de qualquer cliente para todos os clientes conectados.

Para saber mais sobre o SignalR, confira a introdução:

[Introdução ao ASP.NET Core SignalR](#)

# Usar o SignalR do ASP.NET Core com TypeScript e Webpack

28/01/2019 • 19 minutes to read • [Edit Online](#)

Por [Sébastien Sougnez](#) e [Scott Addie](#)

O [Webpack](#) habilita os desenvolvedores a agrupar e criar recursos de um aplicativo Web do lado do cliente. Este tutorial demonstra como usar o Webpack em um aplicativo Web SignalR do ASP.NET Core cujo cliente é escrito em [TypeScript](#).

Neste tutorial, você aprenderá como:

- Gerar um aplicativo inicial SignalR do ASP.NET Core por scaffold
- Configurar o cliente TypeScript do SignalR
- Configurar um pipeline de build usando o Webpack
- Configurar o servidor SignalR
- Habilitar a comunicação entre o cliente e o servidor

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Prerequisites

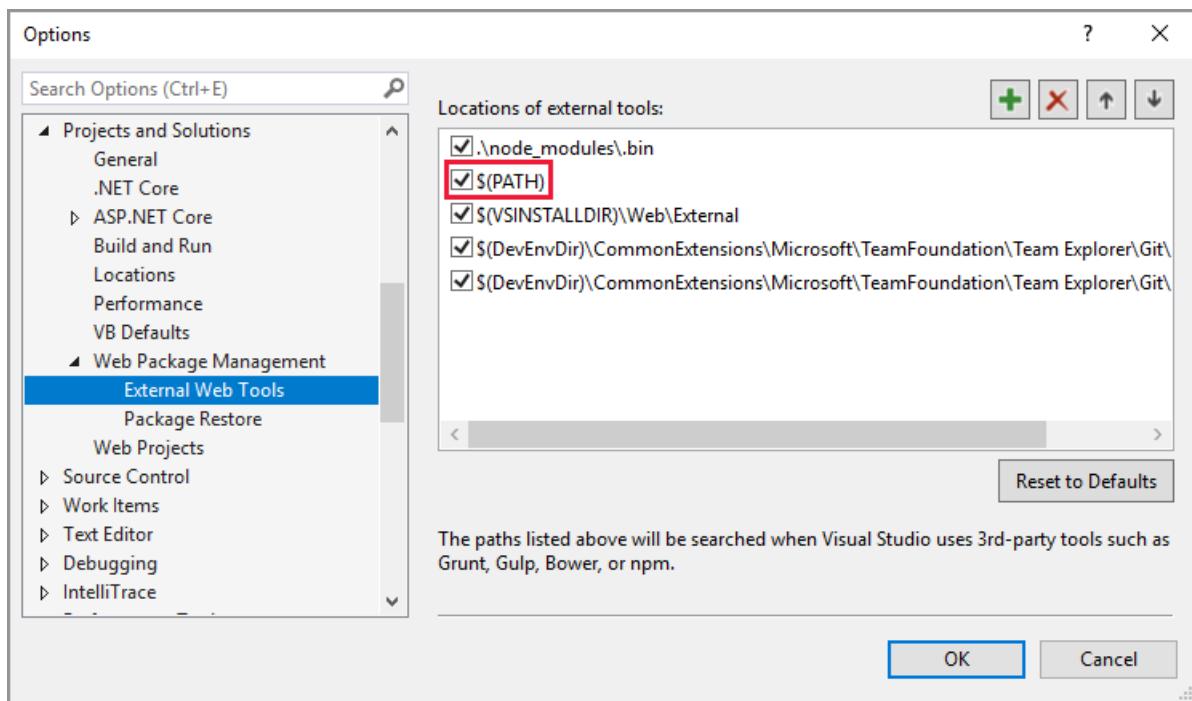
- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio 2017 versão 15.9 ou posterior](#) com a carga de trabalho **ASP.NET e desenvolvimento para a Web**
- [SDK 2.2 ou posterior do .NET Core](#)

## Criar o aplicativo Web do ASP.NET Core

- [Visual Studio](#)
- [Visual Studio Code](#)

Configure o Visual Studio para pesquisar o npm na variável de ambiente *PATH*. Por padrão, o Visual Studio usa a versão do npm encontrada no diretório de instalação. Siga estas instruções no Visual Studio:

1. Navegue para **Ferramentas > Opções > Projetos e Soluções > Gerenciamento de Pacotes da Web > Ferramentas da Web Externas**.
2. Selecione a entrada `$(PATH)` na lista. Clique na seta para cima para mover a entrada para a segunda posição da lista.



A configuração do Visual Studio foi concluída. É hora de criar o projeto.

1. Use a opção do menu **Arquivo > Novo > Projeto** e escolha o modelo **Aplicativo Web do ASP.NET Core**.
2. Dê ao projeto o nome *SignalRWebPack* e selecione **OK**.
3. Selecione *.NET Core* no menu suspenso da estrutura de destino e selecione *ASP.NET Core 2.2* no menu suspenso do seletor de estrutura. Selecione o modelo **Vazio** e selecione **OK**.

## Configurar Webpack e TypeScript

As etapas a seguir configuram a conversão do TypeScript para JavaScript e o agrupamento de recursos no lado do cliente.

1. Execute o seguinte comando na raiz do projeto para criar um arquivo *package.json*:

```
npm init -y
```

2. Adicione a propriedade destacada ao arquivo *package.json*:

```
{
  "name": "SignalRWebPack",
  "version": "1.0.0",
  "private": true,
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Definir a propriedade `private` como `true` evita avisos de instalação de pacote na próxima etapa.

3. Instalar os pacotes de npm exigidos. Execute o seguinte comando na raiz do projeto:

```
npm install -D -E clean-webpack-plugin@0.1.19 css-loader@0.28.11 html-webpack-plugin@3.2.0 mini-css-extract-plugin@0.4.0 ts-loader@4.4.1 typescript@2.9.2 webpack@4.12.0 webpack-cli@3.0.6
```

Alguns detalhes de comando a se observar:

- Um número de versão segue o sinal `@` para cada nome de pacote. O npm instala essas versões específicas do pacote.
- A opção `-E` desabilita o comportamento padrão do npm de escrever os operadores de intervalo de [versão semântica](#) para `package.json`. Por exemplo, `"webpack": "4.12.0"` é usado em vez de `"webpack": "^4.12.0"`. Essa opção evita atualizações não intencionais para versões de pacote mais recentes.

Veja os documentos oficiais de [instalação de npm](#) para saber mais detalhes.

#### 4. Substitua a propriedade `scripts` do arquivo `package.json` com o seguinte snippet:

```
"scripts": {  
  "build": "webpack --mode=development --watch",  
  "release": "webpack --mode=production",  
  "publish": "npm run release && dotnet publish -c Release"  
},
```

Uma explicação sobre os scripts:

- `build`: agrupa os recursos do lado do cliente no modo de desenvolvimento e busca alterações no arquivo. O observador de arquivos faz com que o lote se regenere toda vez que um arquivo de projeto é alterado. A opção `mode` desabilita otimizações de produção, como tree shaking e minificação. Use somente `build` no desenvolvimento.
- `release`: agrupa recursos do lado do cliente no modo de produção.
- `publish`: executa o script `release` para agrupar recursos do lado do cliente no modo de produção. Chama o comando [publicar](#) da CLI do .NET Core para publicar o aplicativo.

#### 5. Crie um arquivo chamado `webpack.config.js` na raiz do projeto, com o seguinte conteúdo:

```

const path = require("path");
const HtmlWebpackPlugin = require("html-webpack-plugin");
const CleanWebpackPlugin = require("clean-webpack-plugin");
const MiniCssExtractPlugin = require("mini-css-extract-plugin");

module.exports = {
  entry: "./src/index.ts",
  output: {
    path: path.resolve(__dirname, "wwwroot"),
    filename: "[name].[chunkhash].js",
    publicPath: "/"
  },
  resolve: {
    extensions: [".js", ".ts"]
  },
  module: {
    rules: [
      {
        test: /\.ts$/,
        use: "ts-loader"
      },
      {
        test: /\.css$/,
        use: [MiniCssExtractPlugin.loader, "css-loader"]
      }
    ]
  },
  plugins: [
    new CleanWebpackPlugin(["wwwroot/*"]),
    new HtmlWebpackPlugin({
      template: "./src/index.html"
    }),
    new MiniCssExtractPlugin({
      filename: "css/[name].[chunkhash].css"
    })
  ]
};


```

O arquivo precedente configura a compilação Webpack. Detalhes de configuração a serem notados:

- A propriedade `output` substitui o valor padrão do `dist`. Em vez disso, o lote é emitido no diretório `wwwroot`.
  - A matriz `resolve.extensions` inclui `.js` para importar o JavaScript do cliente SignalR.
6. Crie um novo diretório `src` na raiz do projeto. O propósito é armazenar os ativos do lado do cliente do projeto.
  7. Crie `src/index.html` com o seguinte conteúdo.

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>ASP.NET Core SignalR</title>
</head>
<body>
    <div id="divMessages" class="messages">
    </div>
    <div class="input-zone">
        <label id="lblMessage" for="tbMessage">Message:</label>
        <input id="tbMessage" class="input-zone-input" type="text" />
        <button id="btnSend">Send</button>
    </div>
</body>
</html>

```

A HTML precedente define a marcação clichê da página inicial.

8. Crie um novo diretório `src/css`. Seu propósito é armazenar os arquivos do projeto `.css`.
9. Crie `src/css/main.css` com o seguinte conteúdo:

```

*, *::before, *::after {
    box-sizing: border-box;
}

html, body {
    margin: 0;
    padding: 0;
}

.input-zone {
    align-items: center;
    display: flex;
    flex-direction: row;
    margin: 10px;
}

.input-zone-input {
    flex: 1;
    margin-right: 10px;
}

.message-author {
    font-weight: bold;
}

.messages {
    border: 1px solid #000;
    margin: 10px;
    max-height: 300px;
    min-height: 300px;
    overflow-y: auto;
    padding: 5px;
}

```

O arquivo precedente `main.css` define o estilo do aplicativo.

10. Crie `src/tsconfig.json` com o seguinte conteúdo:

```
{
  "compilerOptions": {
    "target": "es5"
  }
}
```

O código precedente configura o compilador TypeScript para produzir um JavaScript compatível com [ECMAScript 5](#).

11. Crie `src/index.ts` com o seguinte conteúdo:

```
import "./css/main.css";

const divMessages: HTMLDivElement = document.querySelector("#divMessages");
const tbMessage: HTMLInputElement = document.querySelector("#tbMessage");
const btnSend: HTMLButtonElement = document.querySelector("#btnSend");
const username = new Date().getTime();

tbMessage.addEventListener("keyup", (e: KeyboardEvent) => {
  if (e.keyCode === 13) {
    send();
  }
});

btnSend.addEventListener("click", send);

function send() {
```

O TypeScript precedente recupera referências a elementos DOM e anexa dois manipuladores de eventos:

- `keyup` : esse evento é acionado quando o usuário digita algo na caixa de texto identificada como `tbMessage`. A função `send` é chamada quando o usuário pressionar a tecla **Enter**.
- `click` : esse evento é acionado quando o usuário clica no botão **Enviar**. A função `send` é chamada.

## Configurar o aplicativo do ASP.NET Core

1. O código fornecido no método `Startup.Configure` exibe *Olá, Mundo*. Substitua a chamada para o método `app.Run` com chamadas para `UseDefaultFiles` e `UseStaticFiles`.

```
app.UseDefaultFiles();
app.UseStaticFiles();
```

O código precedente permite que o servidor localize e forneça o arquivo `index.html`, se o usuário inserir a URL completa ou a URL raiz do aplicativo Web.

2. Chame `AddSignalR` no método `Startup.ConfigureServices`. Adiciona serviços SignalR ao seu projeto.

```
services.AddSignalR();
```

3. Mapeie uma rota `/hub` para o hub `ChatHub`. Adicione as linhas a seguir ao final do método `Startup.Configure`:

```
app.UseSignalR(options =>
{
    options.MapHub<ChatHub>("/hub");
});
```

4. Crie um novo diretório, chamado *Hubs*, na raiz do projeto. A finalidade é armazenar o hub SignalR, que é criado na próxima etapa.
5. Crie o hub *Hubs/ChatHub.cs* com o código a seguir:

```
using Microsoft.AspNetCore.SignalR;
using System.Threading.Tasks;

namespace SignalRWebPack.Hubs
{
    public class ChatHub : Hub
    {
    }
}
```

6. Adicione o código a seguir ao topo do arquivo *Startup.cs* para resolver a referência `ChatHub`:

```
using SignalRWebPack.Hubs;
```

## Habilitar a comunicação entre o cliente e o servidor

Atualmente, o aplicativo exibe um formulário simples para enviar mensagens. Nada acontece quando você tenta fazer alguma coisa. O servidor está escutando uma rota específica, mas não faz nada com as mensagens enviadas.

1. Execute o comando a seguir na raiz do projeto:

```
npm install @aspnet/signalr
```

O comando precedente instala o [cliente TypeScript do SignalR](#), que permite ao cliente enviar mensagens para o servidor.

2. Adicione o código destacado ao arquivo *src/index.ts*:

```

import "./css/main.css";
import * as signalR from "@aspnet/signalr";

const divMessages: HTMLDivElement = document.querySelector("#divMessages");
const tbMessage: HTMLInputElement = document.querySelector("#tbMessage");
const btnSend: HTMLButtonElement = document.querySelector("#btnSend");
const username = new Date().getTime();

const connection = new signalR.HubConnectionBuilder()
    .withUrl("/hub")
    .build();

connection.start().catch(err => document.write(err));

connection.on("messageReceived", (username: string, message: string) => {
    let m = document.createElement("div");

    m.innerHTML =
        `<div class="message-author">${username}</div><div>${message}</div>`;

    divMessages.appendChild(m);
    divMessages.scrollTop = divMessages.scrollHeight;
});

tbMessage.addEventListener("keyup", (e: KeyboardEvent) => {
    if (e.keyCode === 13) {
        send();
    }
});

btnSend.addEventListener("click", send);

function send() {
}

```

O código precedente é compatível com o recebimento de mensagens do servidor. A classe `HubConnectionBuilder` cria um novo construtor para configurar a conexão do servidor. A função `withUrl` configura a URL do hub.

O SignalR habilita a troca de mensagens entre um cliente e um servidor. Cada mensagem tem um nome específico. Por exemplo, você pode ter mensagens com o nome `messageReceived` que executam a lógica responsável por exibir a nova mensagem na zona de mensagens. É possível escutar uma mensagem específica por meio da função `on`. Você pode escutar qualquer número de nomes de mensagem. Também é possível passar parâmetros para a mensagem, como o nome do autor e o conteúdo da mensagem recebida. Quando o cliente recebe a mensagem, um novo elemento `div` é criado com o nome do autor e o conteúdo da mensagem em seu atributo `innerHTML`. Ele é adicionado ao elemento principal `div` que exibe as mensagens.

3. Agora que o cliente pode receber mensagens, configure-o para enviá-las. Adicione o código destacado ao arquivo `src/index.ts`:

```

import "./css/main.css";
import * as signalR from "@aspnet/signalr";

const divMessages: HTMLDivElement = document.querySelector("#divMessages");
const tbMessage: HTMLInputElement = document.querySelector("#tbMessage");
const btnSend: HTMLButtonElement = document.querySelector("#btnSend");
const username = new Date().getTime();

const connection = new signalR.HubConnectionBuilder()
    .withUrl("/hub")
    .build();

connection.start().catch(err => document.write(err));

connection.on("messageReceived", (username: string, message: string) => {
    let messageContainer = document.createElement("div");

    messageContainer.innerHTML =
        `<div class="message-author">${username}</div><div>${message}</div>`;

    divMessages.appendChild(messageContainer);
    divMessages.scrollTop = divMessages.scrollHeight;
});

tbMessage.addEventListener("keyup", (e: KeyboardEvent) => {
    if (e.keyCode === 13) {
        send();
    }
});

btnSend.addEventListener("click", send);

function send() {
    connection.send("newMessage", username, tbMessage.value)
        .then(() => tbMessage.value = "");
}

```

Enviar uma mensagem por meio da conexão WebSockets exige uma chamada para o método `send`. O primeiro parâmetro do método é o nome da mensagem. Os dados da mensagem residem nos outros parâmetros. Neste exemplo, uma mensagem identificada como `newMessage` é enviada ao servidor. A mensagem é composta do nome de usuário e da entrada em uma caixa de texto. Se o envio for bem-sucedido, o valor da caixa de texto será limpo.

4. Adicione o método em destaque à classe `ChatHub` :

```

using Microsoft.AspNetCore.SignalR;
using System.Threading.Tasks;

namespace SignalRWebPack.Hubs
{
    public class ChatHub : Hub
    {
        public async Task NewMessage(string username, string message)
        {
            await Clients.All.SendAsync("messageReceived", username, message);
        }
    }
}

```

O código precedente transmite as mensagens recebidas para todos os usuários conectados quando o servidor as recebe. Não é necessário ter um método genérico `on` para receber todas as mensagens. Um método nomeado com o nome da mensagem é suficiente.

Neste exemplo, o cliente TypeScript envia uma mensagem identificada como `newMessage`. O método `NewMessage` de C# espera os dados enviados pelo cliente. Uma chamada é feita para o método `SendAsync` em `Clients.All`. As mensagens recebidas são enviadas a todos os clientes conectados ao hub.

## Testar o aplicativo

Confirme que o aplicativo funciona com as seguintes etapas.

- [Visual Studio](#)
- [Visual Studio Code](#)

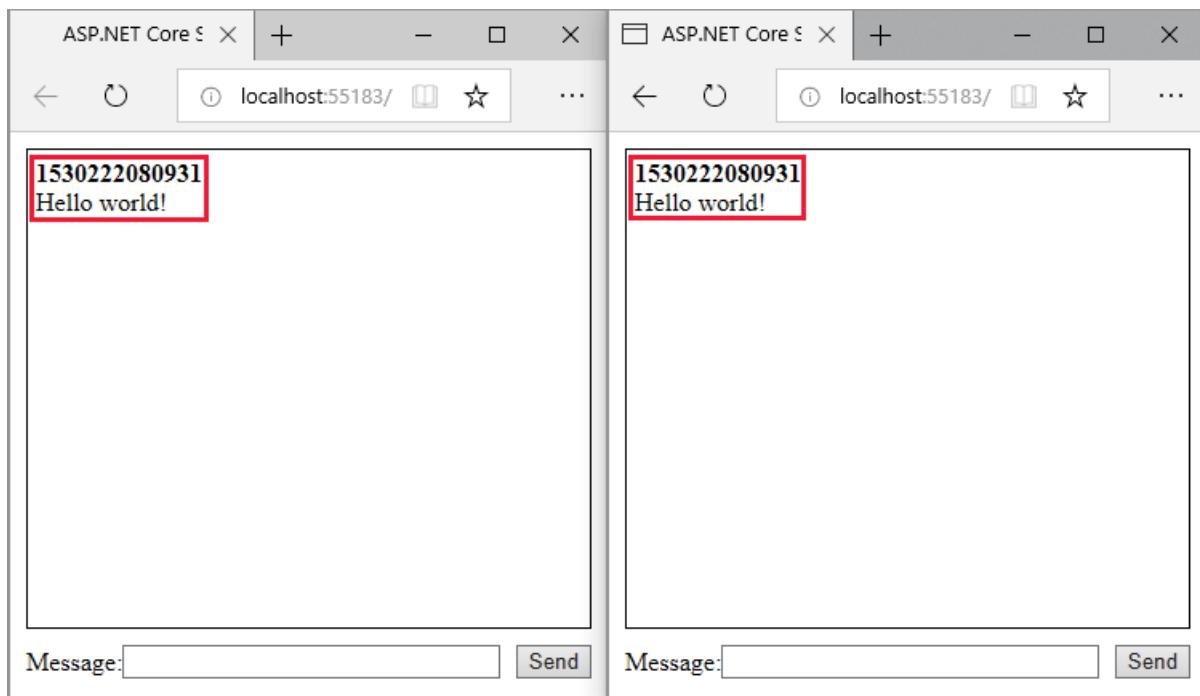
1. Execute o Webpack no modo de *versão*. Usando a janela **Console do Gerenciador de Pacotes**, execute o comando a seguir na raiz do projeto. Se você não estiver na raiz do projeto, insira `cd SignalRWebPack` antes de inserir o comando.

```
npm run release
```

Este comando suspende o fornecimento dos ativos do lado do cliente ao executar o aplicativo. Os ativos são colocados na pasta `wwwroot`.

O Webpack concluiu as seguintes tarefas:

- Limpou os conteúdos do diretório `wwwroot`.
  - Converteu o TypeScript para JavaScript, um processo conhecido como *transpilação*.
  - Reduziu o tamanho do arquivo JavaScript gerado, um processo conhecido como *minificação*.
  - Copiou os arquivos JavaScript, CSS e HTML processados do `src` para o diretório `wwwroot`.
  - Injetou os seguintes elementos no arquivo `wwwroot/index.html`:
    - Uma marca `<link>`, que referencia o arquivo `wwwroot/main.<hash>.css`. Essa marca é colocada imediatamente antes do fim da marca `</head>`.
    - Uma marca `<script>`, que referencia o arquivo minificado `wwwroot/main.<hash>.js`. Essa marca é colocada imediatamente antes do fim da marca `</body>`.
2. Selecione **Debug > Iniciar sem depuração** para iniciar o aplicativo em um navegador sem anexar o depurador. O arquivo `wwwroot/index.html` é fornecido em `http://localhost:<port_number>`.
  3. Abra outra instância do navegador (qualquer navegador). Cole a URL na barra de endereços.
  4. Escolha qualquer navegador, digite algo na caixa de texto **Mensagem** e clique no botão **Enviar**. O nome de usuário exclusivo e a mensagem são exibidas em ambas as páginas instantaneamente.



## Recursos adicionais

- [ASP.NET Core SignalR JavaScript cliente](#)
- [Usando os hubs de SignalR do ASP.NET Core](#)

# Usar os hubs no SignalR do ASP.NET Core

25/01/2019 • 13 minutes to read • [Edit Online](#)

Por [Rachel Appel](#) e [Kevin Griffin](#)

[Exibir ou baixar o código de exemplo \(como fazer o download\)](#)

## O que é um hub SignalR

A API de Hubs de SignalR permite chamar métodos em clientes conectados do servidor. No código do servidor, você define métodos que são chamados pelo cliente. O código do cliente, você define métodos que são chamados do servidor. SignalR cuida de tudo o que nos bastidores que possibilita a comunicação de servidor para cliente e servidor-cliente em tempo real.

## Configurar os hubs de SignalR

O middleware SignalR requer alguns serviços, que são configurados por meio da chamada `services.AddSignalR()`.

```
services.AddSignalR();
```

Ao adicionar a funcionalidade do SignalR para um aplicativo ASP.NET Core, configurar as rotas do SignalR chamando `app.UseSignalR` no `Startup.Configure` método.

```
app.UseSignalR(route =>
{
    route.MapHub<ChatHub>("/chathub");
});
```

## Criar e usar os hubs

Criar um hub, declarando uma classe que herda de `Hub` e adicione os métodos públicos a ele. Os clientes poderão chamar métodos que são definidos como `public`.

```
public class ChatHub : Hub
{
    public Task SendMessage(string user, string message)
    {
        return Clients.All.SendAsync("ReceiveMessage", user, message);
    }
}
```

Você pode especificar um tipo de retorno e parâmetros, incluindo tipos complexos e matrizes, como você faria em qualquer método em c#. O SignalR lida com a serialização e desserialização de objetos complexos e matrizes em seus valores de retorno e parâmetros.

## NOTE

Os hubs são transitórios:

- Não armazene o estado em uma propriedade na classe hub. Cada chamada de método de hub é executada em uma nova instância de hub.
- Use `await` ao chamar métodos assíncronos que dependem do hub de permanecer ativo. Por exemplo, um método, como `Clients.All.SendAsync(...)` pode falhar se ele for chamado sem `await` e o método de hub seja concluída antes de `SendAsync` for concluída.

## O objeto de contexto

O `Hub` classe tem um `Context` propriedade que contém as propriedades a seguir com informações sobre a conexão:

PROPRIEDADE	DESCRIÇÃO
<code>ConnectionId</code>	Obtém a ID exclusiva para a conexão, atribuído pelo SignalR. Há uma ID de conexão para cada conexão.
<code>UserIdentifier</code>	Obtém o <a href="#">identificador de usuário</a> . Por padrão, o SignalR usa o <code>ClaimTypes.NameIdentifier</code> do <code>ClaimsPrincipal</code> associado com a conexão como o identificador de usuário.
<code>User</code>	Obtém o <code>ClaimsPrincipal</code> associado ao usuário atual.
<code>Items</code>	Obtém uma coleção de chave/valor que pode ser usada para compartilhar dados dentro do escopo dessa conexão. Dados podem ser armazenados nessa coleção e ela será mantida para a conexão entre as invocações de método de hub diferentes.
<code>Features</code>	Obtém a coleção de recursos disponíveis sobre a conexão. Por enquanto, essa coleção não é necessária na maioria dos cenários, portanto, ele ainda não está documentado em detalhes.
<code>ConnectionAborted</code>	Obtém um <code>CancellationToken</code> que notifica quando a conexão será anulada.

`Hub.Context` também contém os seguintes métodos:

MÉTODO	DESCRIÇÃO
<code>GetHttpContext</code>	Retorna o <code>HttpContext</code> para a conexão, ou <code>null</code> se a conexão não está associado uma solicitação HTTP. Para conexões HTTP, você pode usar esse método para obter informações como cadeias de caracteres de consulta e cabeçalhos HTTP.
<code>Abort</code>	Anula a conexão.

## O objeto de clientes

O `Hub` classe tem um `Clients` propriedade que contém as seguintes propriedades para a comunicação entre

cliente e servidor:

PROPRIEDADE	DESCRIÇÃO
All	Chama um método em todos os clientes conectados
Caller	Chama um método no cliente que invocou o método de hub
Others	Chama um método em todos os clientes conectados, exceto o cliente que invocou o método

`Hub.Clients` também contém os seguintes métodos:

MÉTODO	DESCRIÇÃO
AllExcept	Chama um método em todos os clientes conectados, exceto para as conexões especificadas
Client	Chama um método em um cliente conectado específico
Clients	Chama um método em clientes conectados específicos
Group	Chama um método em todas as conexões no grupo especificado
GroupExcept	Chama um método em todas as conexões do grupo especificado, exceto as conexões especificadas
Groups	Chama um método em vários grupos de conexões
OthersInGroup	Chama um método em um grupo de conexões, excluindo o cliente que invocou o método de hub
User	Chama um método em todas as conexões associadas a um usuário específico
Users	Chama um método em todas as conexões associadas com os usuários especificados

Cada propriedade ou método nas tabelas anteriores retorna um objeto com um `SendAsync` método. O

`SendAsync` método permite que você forneça o nome e parâmetros do método de cliente para chamar.

## Enviar mensagens para os clientes

Para fazer chamadas para clientes específicos, use as propriedades do `clients` objeto. No exemplo a seguir, há três métodos de Hub:

- `SendMessage` envia uma mensagem para todos os clientes conectados usando `clients.All`.
- `SendMessageToCaller` envia uma mensagem de volta para o chamador usando `clients.Caller`.
- `SendMessageToGroups` envia uma mensagem a todos os clientes a `SignalR Users` grupo.

```

public Task SendMessage(string user, string message)
{
    return Clients.All.SendAsync("ReceiveMessage", user, message);
}

public Task SendMessageToCaller(string message)
{
    return Clients.Caller.SendAsync("ReceiveMessage", message);
}

public Task SendMessageToGroup(string message)
{
    return Clients.Group("SignalR Users").SendAsync("ReceiveMessage", message);
}

```

## Hubs com rigidez de tipos

Uma desvantagem de usar `SendAsync` é que ele se baseia em uma cadeia de caracteres mágica para especificar o método de cliente a ser chamado. Isso deixa o código aberto para erros de tempo de execução se o nome do método está incorreto ou ausente do cliente.

Uma alternativa ao uso `SendAsync` é tipar fortemente os `Hub` com `Hub<T>`. No exemplo a seguir, o `ChatHub` métodos de cliente foram extraídos por em uma interface chamada `IChatClient`.

```

public interface IChatClient
{
    Task ReceiveMessage(string user, string message);
    Task ReceiveMessage(string message);
}

```

Essa interface pode ser usada para refatorar anterior `ChatHub` exemplo.

```

public class StronglyTypedChatHub : Hub<IChatClient>
{
    public async Task SendMessage(string user, string message)
    {
        await Clients.All.ReceiveMessage(user, message);
    }

    public Task SendMessageToCaller(string message)
    {
        return Clients.Caller.ReceiveMessage(message);
    }
}

```

Usando `Hub<IChatClient>` habilita a verificação de tempo de compilação dos métodos do cliente. Isso evita problemas causados pelo uso de cadeias de caracteres mágicas desde `Hub<T>` só pode fornecer acesso aos métodos definidos na interface.

Usando fortemente tipado `Hub<T>` desabilita a capacidade de usar `SendAsync`. Todos os métodos definidos na interface ainda podem ser definidos como assíncronos. Na verdade, cada um desses métodos deve retornar um `Task`. Uma vez que ele é uma interface, não use o `async` palavra-chave. Por exemplo:

```

public interface IClient
{
    Task ClientMethod();
}

```

## NOTE

O `Async` sufixo não é removido do nome do método. A menos que o método de cliente é definido com `.on('MyMethodAsync')`, você não deve usar `MyMethodAsync` como um nome.

## Alterar o nome de um método de hub

Por padrão, um nome de método de hub do servidor é o nome do método do .NET. No entanto, você pode usar o `HubMethodName` atributo para alterar esse padrão e especificar manualmente um nome para o método. O cliente deve usar esse nome, em vez do nome de método do .NET, ao chamar o método.

```
[HubMethodName("SendMessageToUser")]
public Task DirectMessage(string user, string message)
{
    return Clients.User(user).SendAsync("ReceiveMessage", message);
}
```

## Manipular eventos para uma conexão

A API de Hubs de SignalR fornece o `OnConnectedAsync` e `OnDisconnectedAsync` métodos virtuais para gerenciar e controlar conexões. Substituir o `OnConnectedAsync` método virtual para executar ações quando um cliente se conecta ao Hub, como adicioná-lo a um grupo.

```
public override async Task OnConnectedAsync()
{
    await Groups.AddToGroupAsync(Context.ConnectionId, "SignalR Users");
    await base.OnConnectedAsync();
}
```

Substituir o `OnDisconnectedAsync` método virtual para executar ações quando um cliente se desconecta. Se o cliente se desconecta intencionalmente (chamando `connection.stop()`, por exemplo), o `exception` parâmetro será `null`. No entanto, se o cliente for desconectado devido a um erro (como uma falha de rede), o `exception` parâmetro conterá uma exceção que descreve a falha.

```
public override async Task OnDisconnectedAsync(Exception exception)
{
    await Groups.RemoveFromGroupAsync(Context.ConnectionId, "SignalR Users");
    await base.OnDisconnectedAsync(exception);
}
```

## Tratar erros

As exceções geradas em seus métodos de hub são enviadas ao cliente que invocou o método. No cliente JavaScript, o `invoke` método retorna um `promessa JavaScript`. Quando o cliente recebe um erro com um manipulador anexado à promessa usando `catch`, ele é chamado e passado como um JavaScript `Error` objeto.

```
connection.invoke("SendMessage", user, message).catch(err => console.error(err));
```

Se o seu Hub de lançar uma exceção, as conexões não estão fechadas. Por padrão, o SignalR retorna uma mensagem de erro genérica para o cliente. Por exemplo:

```
Microsoft.AspNetCore.SignalR.HubException: An unexpected error occurred invoking 'MethodName' on the server.
```

Exceções inesperadas geralmente contêm informações confidenciais, como o nome de um servidor de banco de dados em uma exceção acionada quando a conexão de banco de dados falha. O SignalR não expõe estas mensagens de erro detalhadas por padrão como medida de segurança. Consulte a [artigo de considerações de segurança](#) para obter mais informações sobre por que os detalhes da exceção são suprimidos.

Se você tiver um excepcional de condição você *fazer* deseja propagar para o cliente, você pode usar o `HubException` classe. Se você lançar uma `HubException` de seu método de hub do SignalR **será** enviar a mensagem inteira para o cliente, sem modificações.

```
public Task ThrowException()
{
    throw new HubException("This error will be sent to the client!");
}
```

#### NOTE

O SignalR envia apenas o `Message` propriedade da exceção para o cliente. O rastreamento de pilha e outras propriedades na exceção não estão disponíveis para o cliente.

## Recursos relacionados

- [Introdução ao SignalR do ASP.NET Core](#)
- [Cliente JavaScript](#)
- [Publicar no Azure](#)

# Enviar mensagens de fora de um hub

25/01/2019 • 3 minutes to read • [Edit Online](#)

Por [Mikael Mengistu](#)

O hub do SignalR é a abstração central para enviar mensagens para os clientes conectados ao servidor SignalR. Também é possível enviar mensagens de outros lugares no seu aplicativo usando o `IHubContext` service. Este artigo explica como acessar um SignalR `IHubContext` para enviar notificações para clientes externos um hub.

[Exibir ou baixar o código de exemplo \(como fazer o download\)](#)

## Obtenha uma instância de `IHubContext`

No SignalR do ASP.NET Core, você pode acessar uma instância de `IHubContext` por meio da injeção de dependência. Você pode injetar uma instância do `IHubContext` em um controlador, middleware ou outro serviço de injeção de dependência. Use a instância para enviar mensagens para os clientes.

### NOTE

Isso é diferente do ASP.NET 4.x SignalR que usado `GlobalHost` para fornecer acesso ao `IHubContext`. O ASP.NET Core tem uma estrutura de injeção de dependência que remove a necessidade de neste singleton global.

### Injetar uma instância do `IHubContext` em um controlador

Você pode injetar uma instância do `IHubContext` em um controlador, adicionando-o para seu construtor:

```
public class HomeController : Controller
{
    private readonly IHubContext<NotificationHub> _hubContext;

    public HomeController(IHubContext<NotificationHub> hubContext)
    {
        _hubContext = hubContext;
    }
}
```

Agora, com acesso a uma instância de `IHubContext`, você pode chamar métodos de hub, como se estivessem no próprio hub.

```
public async Task<IActionResult> Index()
{
    await _hubContext.Clients.All.SendAsync("Notify", $"Home page loaded at: {DateTime.Now}");
    return View();
}
```

### Obtenha uma instância de `IHubContext` no middleware

Acesso a `IHubContext` dentro do pipeline de middleware da seguinte forma:

```
app.Use(async (context, next) =>
{
    var hubContext = context.RequestServices
        .GetRequiredService<IHubContext<MyHub>>();
    //...
});
```

#### NOTE

Quando os métodos de hub são chamados de fora do `Hub` de classe, não há nenhum chamador associado com a invocação. Portanto, não há nenhum acesso para o `ConnectionId`, `Caller`, e `Others` propriedades.

### Injetar um HubContext fortemente tipados

Para injetar um `HubContext` fortemente tipadas, certifique-se de seu Hub herda de `Hub<T>`. Injetá-lo usando o `IHubContext<THub, T>` interface em vez de `IHubContext<THub>`.

```
public class ChatController : Controller
{
    public IHubContext<ChatHub, IChatClient> _strongChatHubContext { get; }

    public ChatController(IHubContext<ChatHub, IChatClient> chatHubContext)
    {
        _strongChatHubContext = chatHubContext;
    }

    public async Task SendMessage(string message)
    {
        await _strongChatHubContext.Clients.All.ReceiveMessage(message);
    }
}
```

## Recursos relacionados

- [Introdução](#)
- [Hubs](#)
- [Publicar no Azure](#)

# Gerenciar usuários e grupos no SignalR

02/02/2019 • 4 minutes to read • [Edit Online](#)

Por [Brennan Conroy](#)

O SignalR permite que mensagens sejam enviadas para todas as conexões associadas a um usuário específico, bem como grupos de conexões nomeados.

[Exibir ou baixar o código de exemplo \(como fazer o download\)](#)

## Usuários no SignalR

O SignalR permite enviar mensagens para todas as conexões associadas a um usuário específico. Por padrão, o SignalR usa o `ClaimTypes.NameIdentifier` do `ClaimsPrincipal` associado com a conexão como o identificador de usuário. Um único usuário pode ter várias conexões a um aplicativo do SignalR. Por exemplo, um usuário pode ser conectado em sua área de trabalho, bem como seu telefone. Cada dispositivo tem uma conexão SignalR separado, mas eles são todos associados ao mesmo usuário. Se uma mensagem é enviada para o usuário, todas as conexões associadas ao usuário recebem a mensagem. O identificador de usuário para uma conexão pode ser acessado pelo `context.UserIdentifier` propriedade em seu hub.

Enviar uma mensagem para um usuário específico, passando o identificador de usuário para o `User` funcionar no seu método de hub, conforme mostrado no exemplo a seguir:

### NOTE

O identificador de usuário diferencia maiusculas de minúsculas.

```
public Task SendPrivateMessage(string user, string message)
{
    return Clients.User(user).SendAsync("ReceiveMessage", message);
}
```

## Grupos no SignalR

Um grupo é uma coleção de conexões associado com um nome. As mensagens podem ser enviadas para todas as conexões em um grupo. Grupos são a maneira recomendada para enviar para uma conexão ou várias conexões, porque os grupos são gerenciados pelo aplicativo. Uma conexão pode ser um membro de vários grupos. Isso torna grupos ideal para algo como um aplicativo de bate-papo, onde cada sala pode ser representada como um grupo. Conexões podem ser adicionadas ou removidas de grupos por meio de `AddToGroupAsync` e `RemoveFromGroupAsync` métodos.

```
public async Task AddToGroup(string groupName)
{
    await Groups.AddToGroupAsync(Context.ConnectionId, groupName);

    await Clients.Group(groupName).SendAsync("Send", $"{Context.ConnectionId} has joined the group {groupName}.");
}

public async Task RemoveFromGroup(string groupName)
{
    await Groups.RemoveFromGroupAsync(Context.ConnectionId, groupName);

    await Clients.Group(groupName).SendAsync("Send", $"{Context.ConnectionId} has left the group {groupName}.");
}
```

Associação de grupo não é preservada quando uma conexão se reconecta. A conexão precisa ingressar novamente o grupo quando é restabelecida. Não é possível contar os membros de um grupo, uma vez que essas informações não estarão disponíveis se o aplicativo é dimensionado para vários servidores.

Para proteger o acesso a recursos durante o uso de grupos, use [autenticação e autorização](#) funcionalidade no ASP.NET Core. Se você adicionar apenas os usuários a um grupo quando as credenciais são válidas para esse grupo, as mensagens enviadas a esse grupo só irá para os usuários autorizados. No entanto, os grupos não são um recurso de segurança. Declarações de autenticação têm recursos que grupos não fizer isso, como a expiração e a revogação. Se a permissão do usuário para acessar o grupo for revogada, você precisa detectar que e removê-las manualmente.

#### NOTE

Nomes de grupo diferenciam maiusculas de minúsculas.

## Recursos relacionados

- [Introdução](#)
- [Hubs](#)
- [Publicar no Azure](#)

# Publicar um ASP.NET Core aplicativo SignalR em um aplicativo Web do Azure

25/01/2019 • 3 minutes to read • [Edit Online](#)

Aplicativo Web do Azure é um a computação em nuvem do Microsoft plataforma de serviço para hospedar aplicativos web, incluindo o ASP.NET Core.

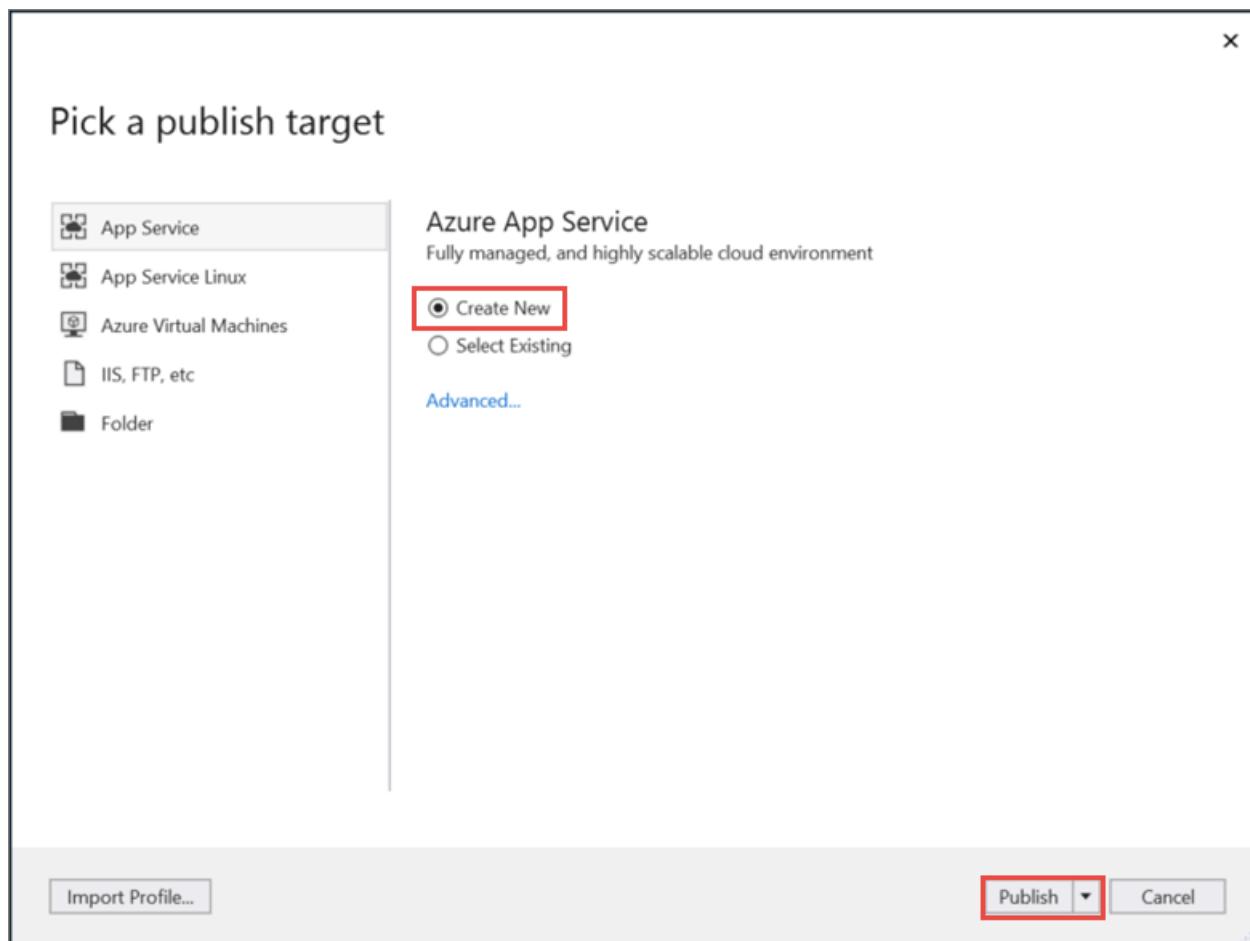
## NOTE

Este artigo refere-se a publicação de um aplicativo de SignalR do ASP.NET Core no Visual Studio. Visite [serviço SignalR para o Azure](#) para obter mais informações sobre como usar o SignalR no Azure.

## Publique o aplicativo

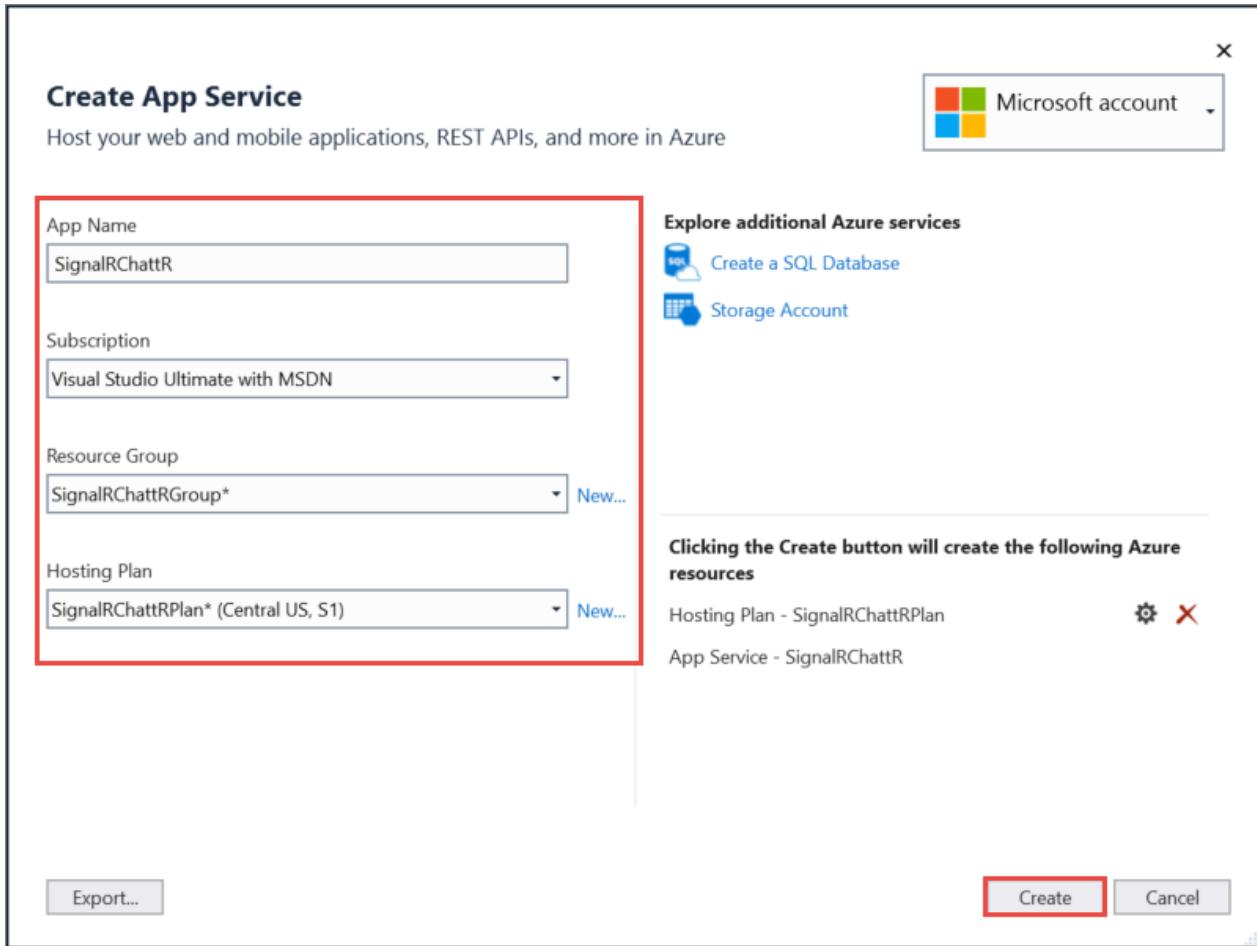
Visual Studio fornece ferramentas internas para publicação de um aplicativo da Web do Azure. Usuário do Visual Studio Code pode usar [CLI do Azure](#) comandos para publicar aplicativos no Azure. Este artigo aborda a publicação usando as ferramentas do Visual Studio. Para publicar um aplicativo usando a CLI do Azure, consulte [publicar um aplicativo ASP.NET Core no Azure com ferramentas de linha de comando](#).

Clique com botão direito no projeto no **Gerenciador de soluções** e selecione **publicar**. Confirme **criar novo** check-in a **escolher um destino de publicação** caixa de diálogo e selecione **publicar**.



Insira as seguintes informações na **criar serviço de aplicativo** caixa de diálogo e selecione **criar**.

ITEM	DESCRIÇÃO
<b>Nome do aplicativo</b>	Um nome exclusivo do aplicativo.
<b>Assinatura</b>	A assinatura do Azure que o aplicativo usa.
<b>Grupo de recursos</b>	O grupo de recursos relacionados ao qual pertence o aplicativo.
<b>Plano de hospedagem</b>	O plano de preços para o aplicativo web.



Visual Studio conclui as seguintes tarefas:

- Cria um perfil de publicação que contém as configurações de publicação.
- Cria ou usa um existente *aplicativo Web do Azure* com os detalhes fornecidos.
- Publica o aplicativo.
- Inicia um navegador, com o aplicativo web publicado carregado.

Observe o formato da URL para o aplicativo é `azurewebsites.NET {nome do aplicativo}.net`. Por exemplo, um aplicativo chamado `SignalRChattR` tem uma URL que se parece com `https://signalrchattr.azurewebsites.net`.

Se ocorrer um erro de HTTP 502.2, consulte [versão de visualização de implantar o ASP.NET Core no serviço de aplicativo do Azure](#) resolvê-lo.

## Configurar o aplicativo web de SignalR

Aplicativos do ASP.NET SignalR Core que são publicados como um aplicativo Web deve ter [afinidade ARR](#) habilitado. [WebSockets](#) deve ser habilitada, para permitir que o transporte de WebSockets para a função.

No portal do Azure, navegue até **configurações do aplicativo** para seu aplicativo web. Definir **WebSockets** à **On** e verifique se **afinidade ARR** é em **On**.

The screenshot shows the 'Application settings' section of the Azure App Service configuration. The left sidebar lists various settings like Authentication, Managed service identity, and WebJobs. The main area shows general settings for .NET Framework version (v4.7), PHP version (5.6), and Python version (Off). It also includes Java, Java minor version, Java web container, Platform (32-bit selected), and Web sockets (On). The 'Web sockets' setting is highlighted with a red box. Below it, 'Always On' is set to Off. Under 'Managed Pipeline Version', 'Integrated' is selected. A note about affinity cookies is present, and 'ARR Affinity' is set to On, also highlighted with a red box. Other settings shown include Auto Swap (Off) and Auto Swap Slot (Off).

WebSockets e outros transportes [são limitados com base no plano do serviço de aplicativo](#).

## Recursos relacionados

- [Publicar um aplicativo ASP.NET Core no Azure com ferramentas de linha de comando](#)
- [Publicar um aplicativo ASP.NET Core no Azure com o Visual Studio](#)
- [Hospedar e implantar aplicativos de visualização do ASP.NET Core no Azure](#)

# Considerações de design de API do SignalR

13/11/2018 • 4 minutes to read • [Edit Online](#)

Por [Andrew Stanton-Nurse](#)

Este artigo fornece diretrizes para a criação de APIs baseadas no SignalR.

## Usar parâmetros de objeto personalizado para garantir a compatibilidade com versões anteriores

Adicionar parâmetros a um método de hub do SignalR (no cliente ou servidor) é um *alteração significativa*. Isso significa que os clientes/servidores mais antigos receberá erros ao tentar invocar o método sem o número apropriado de parâmetros. No entanto, é a adição de propriedades para um parâmetro de objeto personalizado **não** uma alteração significativa. Isso pode ser usado para projetar APIs compatíveis que são resistentes a alterações no cliente ou servidor.

Por exemplo, considere uma API de servidor semelhante ao seguinte:

```
public async Task<string> GetTotalLength(string param1)
{
    return param1.Length;
}
```

O cliente JavaScript chama esse método usando `invoke` da seguinte maneira:

```
connection.invoke("GetTotalLength", "value1");
```

Se você adicionar posteriormente um segundo parâmetro para o método de servidor, os clientes mais antigos não fornecerá esse valor de parâmetro. Por exemplo:

```
public async Task<string> GetTotalLength(string param1, string param2)
{
    return param1.Length + param2.Length;
}
```

Quando o cliente antigo tenta invocar esse método, ele receberá um erro como este:

```
Microsoft.AspNetCore.SignalR.HubException: Failed to invoke 'GetTotalLength' due to an error on the server.
```

No servidor, você verá uma mensagem de log como esta:

```
System.IO.InvalidDataException: Invocation provides 1 argument(s) but target expects 2.
```

O cliente antigo enviados apenas um parâmetro, mas a API mais recente do servidor necessários dois parâmetros. O uso de objetos personalizados como parâmetros oferece mais flexibilidade. Vamos recriar a API original para usar um objeto personalizado:

```
public class TotalLengthRequest
{
    public string Param1 { get; set; }

}

public async Task GetTotalLength(TotalLengthRequest req)
{
    return req.Param1.Length;
}
```

Agora, o cliente usa um objeto para chamar o método:

```
connection.invoke("GetTotalLength", { param1: "value1" });
```

Em vez de adicionar um parâmetro, adicione uma propriedade para o `TotalLengthRequest` objeto:

```
public class TotalLengthRequest
{
    public string Param1 { get; set; }
    public string Param2 { get; set; }
}

public async Task GetTotalLength(TotalLengthRequest req)
{
    var length = req.Param1.Length;
    if (req.Param2 != null)
    {
        length += req.Param2.Length;
    }
    return length;
}
```

Quando o cliente antigo envia um único parâmetro, o extra `Param2` propriedade será deixada `null`. Você pode detectar uma mensagem enviada por um cliente mais antigo, verificando o `Param2` para `null` e aplicar um valor padrão. Um novo cliente pode enviar os dois parâmetros.

```
connection.invoke("GetTotalLength", { param1: "value1", param2: "value2" });
```

A mesma técnica funciona para métodos definidos no cliente. Você pode enviar um objeto personalizado do lado do servidor:

```
public async Task Broadcast(string message)
{
    await Clients.All.SendAsync("ReceiveMessage", new
    {
        Message = message
    });
}
```

No lado do cliente, você acessa o `Message` propriedade em vez de usar um parâmetro:

```
connection.on("ReceiveMessage", (req) => {
    appendMessageToChatWindow(req.message);
});
```

Se você decidir posteriormente adicionar o remetente da mensagem à carga, adicione uma propriedade no objeto:

```
public async Task Broadcast(string message)
{
    await Clients.All.SendAsync("ReceiveMessage", new
    {
        Sender = Context.User.Identity.Name,
        Message = message
    });
}
```

Os clientes mais antigos não esperando o `Sender` de valor, portanto eles vai ignorá-lo. Um novo cliente pode aceitá-lo com a atualização para a nova propriedade de leitura:

```
connection.on("ReceiveMessage", (req) => {
    let message = req.message;
    if (req.sender) {
        message = req.sender + ": " + message;
    }
    appendMessageToChatWindow(message);
});
```

Nesse caso, o novo cliente também é tolerante a falhas de um servidor antigo que não fornece o `Sender` valor. Uma vez que o servidor antigo não fornecerá a `Sender` valor, o cliente verifica para ver se ele existe antes de acessá-lo.

# Cliente de .NET do SignalR do ASP.NET Core

24/01/2019 • 4 minutes to read • [Edit Online](#)

A biblioteca de cliente .NET de SignalR do ASP.NET Core permite que você se comunicar com os hubs de SignalR em aplicativos .NET.

## NOTE

Xamarin tem requisitos especiais para a versão do Visual Studio. Para obter mais informações, consulte [cliente SignalR 2.1.1 no Xamarin](#).

[Exibir ou baixar código de exemplo \(como baixar\)](#)

O exemplo de código neste artigo é um aplicativo do WPF que usa o cliente .NET de SignalR do ASP.NET Core.

## Instalar o pacote de cliente .NET do SignalR

O `Microsoft.AspNetCore.SignalR.Client` pacote é necessário para clientes .NET conectar-se aos hubs de SignalR. Para instalar a biblioteca de cliente, execute o seguinte comando na **Package Manager Console** janela:

```
Install-Package Microsoft.AspNetCore.SignalR.Client
```

## Conectar a um hub

Para estabelecer uma conexão, cria uma `HubConnectionBuilder` e chamar `Build`. A URL do hub, protocolo, o tipo de transporte, nível de log, cabeçalhos e outras opções podem ser configuradas durante a criação de uma conexão. Configurar as opções necessárias, inserindo qualquer um dos `HubConnectionBuilder` métodos em `Build`. Iniciar a conexão com `StartAsync`.

```

using System;
using System.Threading.Tasks;
using System.Windows;
using Microsoft.AspNetCore.SignalR.Client;

namespace SignalRChatClient
{
    public partial class MainWindow : Window
    {
        HubConnection connection;
        public MainWindow()
        {
            InitializeComponent();

            connection = new HubConnectionBuilder()
                .WithUrl("http://localhost:53353/ChatHub")
                .Build();

            connection.Closed += async (error) =>
            {
                await Task.Delay(new Random().Next(0,5) * 1000);
                await connection.StartAsync();
            };
        }

        private async void connectButton_Click(object sender, RoutedEventArgs e)
        {
            connection.On<string, string>("ReceiveMessage", (user, message) =>
            {
                this.Dispatcher.Invoke(() =>
                {
                    var newMessage = $"{user}: {message}";
                    messagesList.Items.Add(newMessage);
                });
            });

            try
            {
                await connection.StartAsync();
                messagesList.Items.Add("Connection started");
                connectButton.IsEnabled = false;
                sendButton.IsEnabled = true;
            }
            catch (Exception ex)
            {
                messagesList.Items.Add(ex.Message);
            }
        }

        private async void sendButton_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                await connection.InvokeAsync("SendMessage",
                    userTextBox.Text, messageTextBox.Text);
            }
            catch (Exception ex)
            {
                messagesList.Items.Add(ex.Message);
            }
        }
    }
}

```

## Lidar com a conexão perdida

Use o `Closed` eventos para responder a uma conexão perdida. Por exemplo, você talvez queira automatizar a reconexão.

O `Closed` evento requer um delegado que retorna um `Task`, que permite que o código assíncrono executar sem usar `async void`. Para satisfazer a assinatura do delegado em um `Closed` manipulador de eventos que é executado de forma síncrona, retorna `Task.CompletedTask`:

```
connection.Closed += (error) => {
    // Do your close logic.
    return Task.CompletedTask;
};
```

O principal motivo para o suporte assíncrono é portanto, você pode reiniciar a conexão. Iniciar uma conexão é uma ação assíncrona.

Em um `Closed` manipulador que reinicia a conexão, considere aguardar algum atraso aleatório evitar sobrecarregar o servidor, conforme mostrado no exemplo a seguir:

```
connection.Closed += async (error) =>
{
    await Task.Delay(new Random().Next(0,5) * 1000);
    await connection.StartAsync();
};
```

## Chamar métodos de hub do cliente

`InvokeAsync` chama métodos no hub. Passe o nome do método de hub e quaisquer argumentos definidos no método de hub para `InvokeAsync`. O SignalR é assíncrono, portanto, use `async` e `await` ao fazer as chamadas.

```
await connection.InvokeAsync("SendMessage",
    userTextBox.Text, messageTextBox.Text);
```

## Chamar métodos de cliente do hub

Definir métodos de hub de chamadas usando `connection.on` depois de criar, mas antes de iniciar a conexão.

```
connection.On<string, string>("ReceiveMessage", (user, message) =>
{
    this.Dispatcher.Invoke(() =>
    {
        var newMessage = $"{user}: {message}";
        messagesList.Items.Add(newMessage);
    });
});
```

O código anterior no `connection.on` é executado quando o código do lado do servidor chama-o usando o `SendAsync` método.

```
public async Task SendMessage(string user, string message)
{
    await Clients.All.SendAsync("ReceiveMessage", user, message);
}
```

## Registro em log e tratamento de erros

Tratar erros com uma instrução try-catch. Inspecione o `Exception` objeto para determinar a ação apropriada a tomar depois de ocorrer um erro.

```
try
{
    await connection.InvokeAsync("SendMessage",
        userTextBox.Text, messageTextBox.Text);
}
catch (Exception ex)
{
    messagesList.Items.Add(ex.Message);
}
```

## Recursos adicionais

- [Hubs](#)
- [Cliente JavaScript](#)
- [Publicar no Azure](#)

# Cliente de Java do SignalR Core ASP.NET

05/12/2018 • 5 minutes to read • [Edit Online](#)

Por [Mikael Mengistu](#)

O cliente de Java permite se conectar a um servidor do SignalR do ASP.NET Core do código Java, incluindo aplicativos Android. Como o [cliente JavaScript](#) e o [cliente .NET](#), o cliente de Java permite que você receber e enviar mensagens a um hub em tempo real. O cliente de Java está disponível no ASP.NET Core 2.2 e posterior.

O aplicativo de console do Java de exemplo referenciado neste artigo usa o cliente de Java do SignalR.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Instalar o pacote de cliente Java do SignalR

O `1.0.0 signalr` arquivo JAR permite que os clientes se conectem aos hubs de SignalR. Para localizar o número de versão de arquivo JAR mais recente, consulte o [resultados da pesquisa Maven](#).

Se usando o Gradle, adicione a seguinte linha para o `dependencies` seção do seu `Build.gradle` arquivo:

```
implementation 'com.microsoft.signalr:signalr:1.0.0'
```

Se usando o Maven, adicione as seguintes linhas dentro de `<dependencies>` elemento da sua `POM.XML` arquivo:

```
<dependency>
  <groupId>com.microsoft.signalr</groupId>
  <artifactId>signalr</artifactId>
  <version>1.0.0</version>
</dependency>
```

## Conectar a um hub

Para estabelecer uma `HubConnection`, o `HubConnectionBuilder` deve ser usado. O nível de log e a URL do hub pode ser configurado durante a criação de uma conexão. Configurar as opções necessárias chamando o `HubConnectionBuilder` métodos antes `build`. Iniciar a conexão com `start`.

```
HubConnection hubConnection = HubConnectionBuilder.create(input)
    .build();
```

## Chamar métodos de hub do cliente

Uma chamada para `send` invoca um método de hub. Passe o nome do método de hub e quaisquer argumentos definidos no método de hub para `send`.

```
hubConnection.send("Send", input);
```

## Chamar métodos de cliente do hub

Use `hubConnection.on` para definir métodos no cliente que o hub pode chamar. Defina os métodos depois de criar,

mas antes de iniciar a conexão.

```
hubConnection.on("Send", (message) -> {
    System.out.println("New Message: " + message);
}, String.class);
```

## Adicionar registro em log

O cliente SignalR Java usa a [SLF4J](#) biblioteca para registro em log. É uma API de alto nível de log que permite aos usuários da biblioteca de escolher sua própria implementação de log específico, colocando em uma dependência de log específico. O trecho de código a seguir mostra como usar `java.util.logging` com o cliente de Java do SignalR.

```
implementation 'org.slf4j:slf4j-jdk14:1.7.25'
```

Se você não configurar o registro em log em suas dependências, SLF4J carrega um agente de operação não padrão com a seguinte mensagem de aviso:

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
```

Isso pode ser ignorado.

## Notas de desenvolvimento do Android

Com relação à compatibilidade do SDK do Android para os recursos de cliente do SignalR, considere os seguintes itens ao especificar sua versão do SDK do Android de destino:

- O cliente de Java do SignalR será executado no nível da API Android 16 e versões posteriores.
- Conectar-se por meio do serviço Azure SignalR exigirá o nível da API Android 20 e posterior porque a [serviço do Azure SignalR](#) requer o TLS 1.2 e não dá suporte a conjuntos de codificação baseado em SHA-1. Android adicionou suporte para [SHA-256 \(e superior\) conjuntos de codificação](#) no nível da API 20.

## Configurar a autenticação de token de portador

No cliente de Java do SignalR, você pode configurar um token de portador para usar para autenticação, fornecendo uma "access token fábrica" para o [HttpHubConnectionBuilder](#). Use [withAccessTokenFactory](#) para fornecer uma [RxJava único](#). Com uma chamada para [Single.defer](#), você pode escrever a lógica para gerar tokens de acesso do cliente.

```
HubConnection hubConnection = HubConnectionBuilder.create("YOUR HUB URL HERE")
    .withAccessTokenProvider(Single.defer(() -> {
        // Your logic here.
        return Single.just("An Access Token");
    })).build();
```

## Limitações conhecidas

- Há suporte para apenas o protocolo JSON.
- Há suporte para apenas o transporte de WebSockets.
- Streaming ainda não tem suporte.

## Recursos adicionais

- [Referência de API Java](#)
- [Usando os hubs de SignalR do ASP.NET Core](#)
- [ASP.NET Core SignalR JavaScript cliente](#)
- [Publicar um ASP.NET Core SignalR aplicativo ao aplicativo Web do Azure](#)

# ASP.NET Core SignalR JavaScript cliente

24/01/2019 • 7 minutes to read • [Edit Online](#)

Por [Rachel Appel](#)

A biblioteca de cliente JavaScript de SignalR do ASP.NET Core permite aos desenvolvedores chamar o código de hub do lado do servidor.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Instalar o pacote de cliente do SignalR

A biblioteca de cliente SignalR JavaScript é entregue como um [npm](#) pacote. Se você estiver usando o Visual Studio, execute `npm install` do **Package Manager Console** enquanto na pasta raiz. Para Visual Studio Code, execute o comando a partir de **Terminal integrado**.

```
npm init -y
npm install @aspnet/signalr
```

NPM instala o conteúdo do pacote na `node_modules\@aspnet\signalr\dist\browser` pasta. Criar uma nova pasta chamada `signalr` sob o `wwwroot\lib` pasta. Cópia de `signalr.js` do arquivo para o `wwwroot\lib\signalr` pasta.

## Usar o cliente SignalR JavaScript

Referência de cliente SignalR JavaScript no `<script>` elemento.

```
<script src="~/lib/signalr/signalr.js"></script>
```

## Conectar a um hub

O código a seguir cria e inicia uma conexão. Nome do hub é diferencia maiusculas de minúsculas.

```
const connection = new signalR.HubConnectionBuilder()
    .withUrl("/chatHub")
    .configureLogging(signalR.LogLevel.Information)
    .build();
```

## Conexões entre origens

Normalmente, os navegadores carregam as conexões do mesmo domínio que a página solicitada. No entanto, há ocasiões em que é necessária uma conexão para outro domínio.

Para impedir a leitura de dados confidenciais de outro site, um site mal-intencionado **conexões entre origens** estão desabilitados por padrão. Para permitir que uma solicitação entre origens, habilite-o no `Startup` classe.

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using SignalRChat.Hubs;

namespace SignalRChat
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        public void ConfigureServices(IServiceCollection services)
        {
            services.Configure<CookiePolicyOptions>(options =>
            {
                options.CheckConsentNeeded = context => true;
                options.MinimumSameSitePolicy = SameSiteMode.None;
            });

            services.AddMvc();

            services.AddCors(options => options.AddPolicy("CorsPolicy",
                builder =>
                {
                    builder.AllowAnyMethod().AllowAnyHeader()
                        .WithOrigins("http://localhost:55830")
                        .AllowCredentials();
                }));
        }

        public void AddSignalR();
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseBrowserLink();
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Error");
            app.UseHsts();
        }

        app.UseHttpsRedirection();
        app.UseStaticFiles();
        app.UseCookiePolicy();
        app.UseCors("CorsPolicy");
        app.UseSignalR(routes =>
        {
            routes.MapHub<ChatHub>("/chathub");
        });
        app.UseMvc();
    }
}
```

## Chamar métodos de hub do cliente

Clientes JavaScript chamam métodos públicos em hubs por meio de `invoke` método da `HubConnection`. O `invoke` método aceita dois argumentos:

- O nome do método de hub. No exemplo a seguir, o nome do método no hub é `SendMessage`.
- Quaisquer argumentos definidos no método de hub. No exemplo a seguir, é o nome do argumento `message`. O exemplo de código usa a sintaxe da função de seta tem suporte em versões atuais de todos os principais navegadores, exceto o Internet Explorer.

```
connection.invoke("SendMessage", user, message).catch(err => console.error(err.toString()));
```

## Chamar métodos de cliente do hub

Para receber mensagens do hub, definir um método usando o `na` método o `HubConnection`.

- O nome do método de cliente JavaScript. No exemplo a seguir, é o nome do método `ReceiveMessage`.
- O hub passa para o método de argumentos. No exemplo a seguir, o valor do argumento é `message`.

```
connection.on("ReceiveMessage", (user, message) => {
    const encodedMsg = user + " says " + message;
    const li = document.createElement("li");
    li.textContent = encodedMsg;
    document.getElementById("messagesList").appendChild(li);
});
```

O código anterior no `connection.on` é executado quando o código do lado do servidor chama-o usando o `SendAsync` método.

```
public async Task SendMessage(string user, string message)
{
    await Clients.All.SendAsync("ReceiveMessage", user, message);
}
```

O SignalR determina qual método de cliente para chamar, correspondendo o nome do método e argumentos definidos no `SendAsync` e `connection.on`.

### NOTE

Como uma prática recomendada, chame o `inicie` método na `HubConnection` depois `on`. Isso garante que os manipuladores são registrados antes de todas as mensagens são recebidas.

## Registro em log e tratamento de erros

Cadeia de um `catch` método até o final do `start` método para lidar com erros do lado do cliente. Use `console.error` para erros de saída para o console do navegador.

```
connection.start().catch(function (err) {
    return console.error(err.toString());
});
```

Configure o rastreamento de log do lado do cliente, passando um agente de log e o tipo de evento para

registerar em log quando a conexão é feita. As mensagens são registradas com o nível de log especificado e superior. Níveis de log disponíveis são da seguinte maneira:

- `signalR.LogLevel.Error` – Mensagens de erro. Logs `Error` somente mensagens.
- `signalR.LogLevel.Warning` – Mensagens de aviso sobre erros em potencial. Os logs `Warning`, e `Error` mensagens.
- `signalR.LogLevel.Information` – Mensagens de status sem erros. Os logs `Information`, `Warning`, e `Error` mensagens.
- `signalR.LogLevel.Trace` – Mensagens de rastreamento. Registra tudo, incluindo dados transportados entre cliente e o hub.

Use o [configureLogging](#) método `HubConnectionBuilder` para configurar o nível de log. As mensagens são registradas para o console do navegador.

```
const connection = new signalR.HubConnectionBuilder()
    .withUrl("/chatHub")
    .configureLogging(signalR.LogLevel.Information)
    .build();
```

## Os clientes de reconexão

O cliente JavaScript para o SignalR não reconectar-se automaticamente. Você deve escrever código que será reconectada seu cliente manualmente. O código a seguir demonstra uma abordagem típica de reconexão:

1. Uma função (nesse caso, o `start` função) é criado para iniciar a conexão.
2. Chame o `start` função em que a conexão `onclose` manipulador de eventos.

```
async function start() {
    try {
        await connection.start();
        console.log('connected');
    } catch (err) {
        console.log(err);
        setTimeout(() => start(), 5000);
    }
};

connection.onclose(async () => {
    await start();
});
```

Uma implementação real seria usar uma retirada exponencial ou repetir um número especificado de vezes antes de desistir.

## Recursos adicionais

- [Referência de API JavaScript](#)
- [Tutorial do JavaScript](#)
- [Tutorial de WebPack e TypeScript](#)
- [Hubs](#)
- [Cliente .NET](#)
- [Publicar no Azure](#)
- [Solicitações entre origens \(CORS\)](#)

# Hospedagem do ASP.NET SignalR Core e dimensionamento

25/01/2019 • 8 minutes to read • [Edit Online](#)

Por [Andrew Stanton-Nurse](#), [Brady Gaster](#), e [Tom Dykstra](#),

Este artigo explica as considerações para aplicativos de alto tráfego que usam o SignalR do ASP.NET Core de dimensionamento e hospedagem.

## Recursos de conexão TCP

O número de conexões TCP simultâneas que pode dar suporte a um servidor web é limitado. Os clientes HTTP padrão usam *efêmero* conexões. Essas conexões podem ser fechadas quando o cliente fica ocioso e reabrir posteriormente. Por outro lado, uma conexão SignalR é *persistente*. Conexões do SignalR permanecem aberta até mesmo quando o cliente vai ocioso. Em um aplicativo de alto tráfego que atende a muitos clientes, essas conexões persistentes podem causar servidores atingido o número máximo de conexões.

Conexões persistentes também consomem memória extra, para acompanhar cada conexão.

O uso intenso de recursos relacionados à conexão pelo SignalR pode afetar outros aplicativos web hospedados no mesmo servidor. Quando o SignalR é aberto e mantém as últimas conexões TCP disponíveis, outros aplicativos web no mesmo servidor também não tem mais conexões disponíveis para eles.

Se um servidor ficar sem conexões, você verá erros de soquete aleatória e erros de redefinição de conexão. Por exemplo:

```
An attempt was made to access a socket in a way forbidden by its access permissions...
```

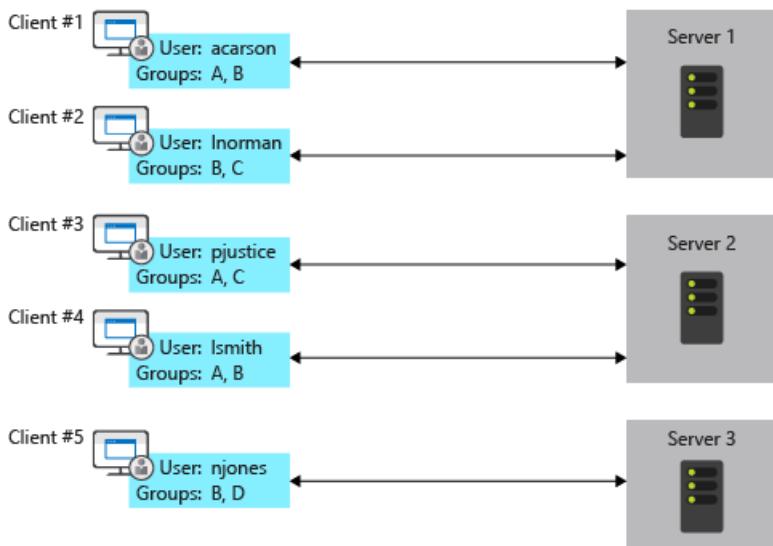
Para evitar o uso de recursos do SignalR causando erros em outros aplicativos da web, execute o SignalR em servidores diferentes do que seus outros aplicativos da web.

Para evitar o uso de recursos do SignalR causando erros em um aplicativo do SignalR, escala horizontalmente para limitar o número de conexões que um servidor deve manipular.

## Expansão do

Um aplicativo que usa o SignalR precisa manter o controle de todas as suas conexões, que cria problemas para um farm de servidores. Adicionar um servidor, e ele obtém novas conexões de outros servidores não conhecer. Por exemplo, o SignalR em cada servidor no diagrama a seguir desconhece as conexões nos outros servidores.

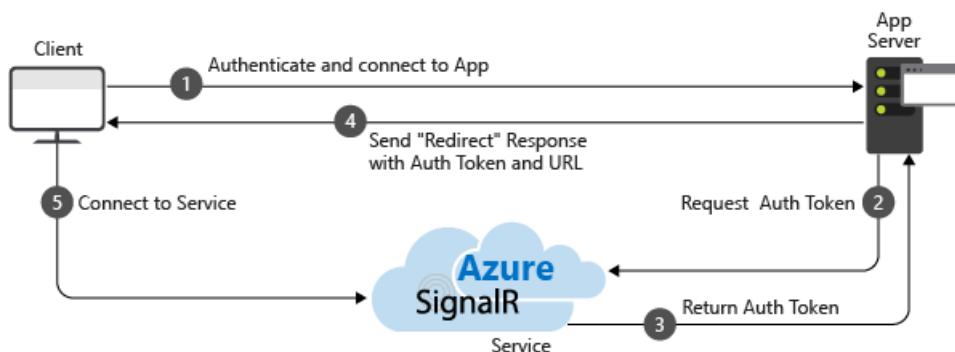
Quando quiser SignalR em um dos servidores enviar uma mensagem a todos os clientes, a mensagem é apenas vai para os clientes conectados a esse servidor.



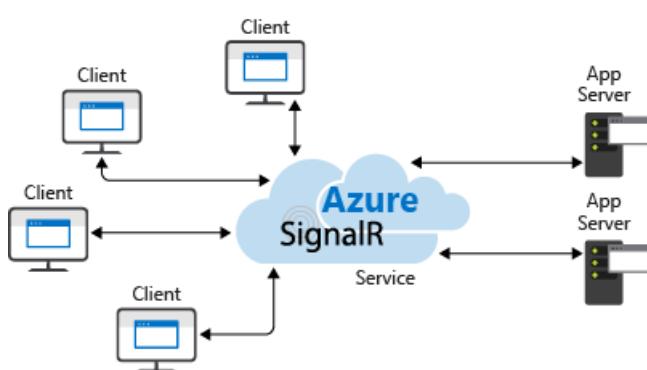
As opções para resolver esse problema são as [serviço do Azure SignalR](#) e [Redis backplane](#).

## Serviço Azure SignalR

O serviço do Azure SignalR é um proxy em vez de um backplane. Cada vez que um cliente inicia uma conexão ao servidor, o cliente é redirecionado para se conectar ao serviço. Esse processo é ilustrado no diagrama a seguir:



O resultado é que o serviço gerencia todas as conexões de cliente, enquanto cada servidor precisa de apenas um pequeno número constante de conexões para o serviço, conforme mostrado no diagrama a seguir:



Essa abordagem de expansão tem várias vantagens em relação a alternativa de backplane do Redis:

- Sessões adesivas, também conhecidas como [afinidade do cliente](#), não é necessário, pois os clientes imediatamente são redirecionados para o serviço do Azure SignalR quando eles se conectam.
- Um aplicativo pode escalar horizontalmente de SignalR com base no número de mensagens enviadas, enquanto o serviço do Azure SignalR é dimensionado automaticamente para lidar com qualquer número de conexões. Por exemplo, pode haver milhares de clientes, mas se apenas algumas mensagens por segundo são enviadas, o aplicativo SignalR não será necessário escalar horizontalmente em vários servidores apenas para lidar com as conexões em si.

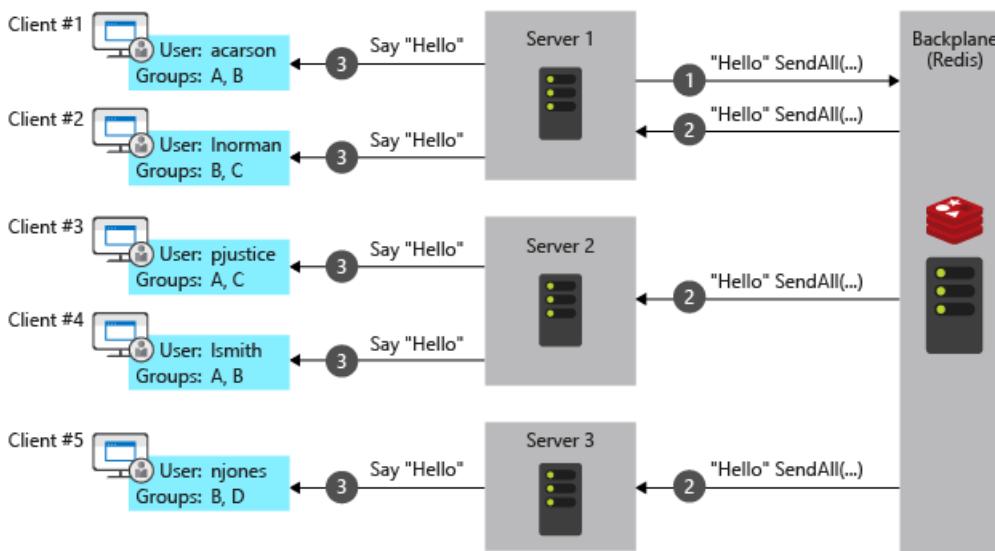
- Um aplicativo de SignalR não usar significativamente mais recursos de conexão que um aplicativo web sem SignalR.

Por esses motivos, recomendamos que o serviço do Azure SignalR para todos os aplicativos do SignalR do ASP.NET Core hospedados no Azure, incluindo o serviço de aplicativo, as VMs e contêineres.

Para obter mais informações, consulte o [documentação do serviço do Azure SignalR](#).

## Backplane de redis

**Redis** é um repositório de chave-valor na memória que dá suporte a um sistema de mensagens com um modelo de publicação/assinatura. O backplane SignalR Redis usa o recurso de publicação/assinatura para encaminhar mensagens para outros servidores. Quando um cliente faz uma conexão, as informações de conexão são passadas ao backplane. Quando um servidor deseja enviar uma mensagem a todos os clientes, ele envia ao backplane. Backplane sabe clientes tudo conectados e quais servidores que eles estão. Ele envia a mensagem a todos os clientes por meio de seus respectivos servidores. Esse processo é ilustrado no diagrama a seguir:



O backplane do Redis é a abordagem recomendada de escalabilidade horizontal para aplicativos hospedados em sua própria infraestrutura. Azure SignalR Service não é uma opção prática para uso em produção com aplicativos no local devido à latência de conexão entre seu data center e um data center do Azure.

As vantagens de serviço do Azure SignalR observadas anteriormente são as desvantagens para o backplane do Redis:

- Sessões adesivas, também conhecidas como [afinidade do cliente](#), é necessário. Depois que uma conexão é iniciada em um servidor, a conexão deve permanecer nesse servidor.
- Um aplicativo de SignalR deve escalar horizontalmente com base no número de clientes, mesmo se algumas mensagens estão sendo enviadas.
- Um aplicativo de SignalR usa significativamente mais recursos de conexão que um aplicativo web sem SignalR.

## Próximas etapas

Para obter mais informações, consulte os seguintes recursos:

- [Documentação do SignalR Service do Azure](#)
- [Configurar um backplane de Redis](#)

# Configurar um backplane de Redis para expansão do SignalR do ASP.NET Core

25/01/2019 • 6 minutes to read • [Edit Online](#)

Por [Andrew Stanton-Nurse](#), [Brady Gaster](#), e [Tom Dykstra](#),

Este artigo explica os aspectos de SignalR específicas de configuração de um [Redis](#) servidor a ser usado para dimensionar um aplicativo do SignalR do ASP.NET Core.

## Configurar um backplane de Redis

- Implante um servidor do Redis.

### IMPORTANT

Para uso em produção, um backplane de Redis é recomendável somente quando ele é executado no mesmo data center que o aplicativo do SignalR. Caso contrário, latência de rede degrada o desempenho. Se seu aplicativo SignalR está em execução na nuvem do Azure, é recomendável o serviço do Azure SignalR em vez de um backplane de Redis. Você pode usar o serviço de Cache Redis do Azure para desenvolvimento e ambientes de teste.

Para obter mais informações, consulte os seguintes recursos:

- [Hospedagem de produção do ASP.NET SignalR Core e dimensionamento](#)
  - [Documentação do redis](#)
  - [Documentação do Cache Redis do Azure](#)
- No aplicativo do SignalR, instale o `Microsoft.AspNetCore.SignalR.Redis` pacote do NuGet. (Há também um `Microsoft.AspNetCore.SignalR.StackExchangeRedis` empacotar, mas que um é para o ASP.NET Core 2.2 e posterior.)
  - No `Startup.ConfigureServices` método, chame `AddRedis` depois `AddSignalR`:

```
services.AddSignalR().AddRedis("<your_Redis_connection_string>");
```

- Defina as opções conforme necessário:

A maioria das opções pode ser definido na cadeia de conexão ou nos [ConfigurationOptions](#) objeto. As opções especificadas em `ConfigurationOptions` substituirão as definido na cadeia de conexão.

O exemplo a seguir mostra como definir opções no `ConfigurationOptions` objeto. Este exemplo adiciona um prefixo de canal para que vários aplicativos podem compartilhar a mesma instância do Redis, conforme explicado na etapa a seguir.

```
services.AddSignalR()
    .AddRedis(connectionString, options => {
        options.Configuration.ChannelPrefix = "MyApp";
    });

```

No código anterior, `options.Configuration` é inicializada com tudo o que foi especificado na cadeia de conexão.

- No aplicativo do SignalR, instale um dos seguintes pacotes NuGet:
  - `Microsoft.AspNetCore.SignalR.StackExchangeRedis` -Depende do stackexchange. Redis 2.X.X. Este é o pacote recomendado para o ASP.NET Core 2.2 e posterior.
  - `Microsoft.AspNetCore.SignalR.Redis` -Depende do 1.X.X stackexchange. Redis. Este pacote não enviará no ASP.NET Core 3.0.
- No `Startup.ConfigureServices` método, chame `AddStackExchangeRedis` depois `AddSignalR` :

```
services.AddSignalR().AddStackExchangeRedis("<your_Redis_connection_string>");
```

- Defina as opções conforme necessário:

A maioria das opções pode ser definido na cadeia de conexão ou nos `ConfigurationOptions` objeto. As opções especificadas em `ConfigurationOptions` substituirão as definido na cadeia de conexão.

O exemplo a seguir mostra como definir opções no `ConfigurationOptions` objeto. Este exemplo adiciona um prefixo de canal para que vários aplicativos podem compartilhar a mesma instância do Redis, conforme explicado na etapa a seguir.

```
services.AddSignalR()
    .AddStackExchangeRedis(connectionString, options => {
        options.Configuration.ChannelPrefix = "MyApp";
    });
}
```

No código anterior, `options.Configuration` é inicializada com tudo o que foi especificado na cadeia de conexão.

Para obter informações sobre as opções do Redis, consulte o [StackExchange Redis documentação](#).

- Se você estiver usando um servidor de Redis para vários aplicativos do SignalR, use um prefixo de canal diferente para cada aplicativo do SignalR.

Configurar um prefixo de canal isola a um aplicativo do SignalR de outras pessoas que usam os prefixos de canal diferente. Se você não atribuir prefixos diferentes, uma mensagem enviada de um aplicativo para todos os seus próprios clientes irão para todos os clientes de todos os aplicativos que usam o servidor Redis como um backplane.

- Configure seu servidor farm balanceamento de carga para sessões adesivas. Aqui estão alguns exemplos de documentação sobre como fazer isso:

- [IIS](#)
- [HAProxy](#)
- [Nginx](#)
- [pfSense](#)

## Erros do servidor de redis

Quando um servidor Redis fica inativo, o SignalR gera exceções que indicam as mensagens não entregues. Algumas mensagens de exceção típico:

- *Mensagem de gravação com falha*
- *Falha ao invocar o método de hub 'MethodName'*
- *Falha na Conexão ao Redis*

O SignalR não armazena em buffer as mensagens para enviá-las quando o servidor de volta a funcionar. Todas as

mensagens enviadas enquanto o servidor Redis estiver inativo serão perdidas.

O SignalR se reconecta automaticamente quando o servidor Redis estiver disponível novamente.

### Comportamento personalizado para falhas de conexão

Aqui está um exemplo que mostra como manipular eventos de falha de conexão do Redis.

```
services.AddSignalR()
    .AddRedis(o =>
{
    o.ConnectionFactory = async writer =>
    {
        var config = new ConfigurationOptions
        {
            AbortOnConnectFail = false
        };
        config.EndPoints.Add(IPAddress.Loopback, 0);
        config.SetDefaultPorts();
        var connection = await ConnectionMultiplexer.ConnectAsync(config, writer);
        connection.ConnectionFailed += (_, e) =>
        {
            Console.WriteLine("Connection to Redis failed.");
        };

        if (!connection.IsConnected)
        {
            Console.WriteLine("Did not connect to Redis.");
        }

        return connection;
    };
});
```

```
services.AddSignalR()
    .AddMessagePackProtocol()
    .AddStackExchangeRedis(o =>
{
    o.ConnectionFactory = async writer =>
    {
        var config = new ConfigurationOptions
        {
            AbortOnConnectFail = false
        };
        config.EndPoints.Add(IPAddress.Loopback, 0);
        config.SetDefaultPorts();
        var connection = await ConnectionMultiplexer.ConnectAsync(config, writer);
        connection.ConnectionFailed += (_, e) =>
        {
            Console.WriteLine("Connection to Redis failed.");
        };

        if (!connection.IsConnected)
        {
            Console.WriteLine("Did not connect to Redis.");
        }

        return connection;
    };
});
```

## Clustering

O clustering é um método para alcançar alta disponibilidade por meio de vários servidores do Redis. Clustering

não é oficialmente suportado, mas pode funcionar.

## Próximas etapas

Para obter mais informações, consulte os seguintes recursos:

- [Hospedagem de produção do ASP.NET SignalR Core e dimensionamento](#)
- [Documentação do redis](#)
- [Documentação do StackExchange Redis](#)
- [Documentação do Cache Redis do Azure](#)

# Host SignalR do ASP.NET Core em serviços em segundo plano

05/02/2019 • 5 minutes to read • [Edit Online](#)

Por [Brady Gaster](#)

Este artigo fornece orientações para:

- Hospedagem de Hubs de SignalR usando um processo de trabalho em segundo plano hospedado com o ASP.NET Core.
- Enviando mensagens para clientes de dentro de um núcleo do .NET de conectados [BackgroundService](#).

[Exibir ou baixar o código de exemplo \(como fazer o download\)](#)

## Conectar o SignalR durante a inicialização

Os Hubs de SignalR do ASP.NET Core no contexto de um processo de trabalho em segundo plano de hospedagem é idêntico à hospedagem de um Hub em um aplicativo web ASP.NET Core. No `Startup.ConfigureServices` chamar um método, `services.AddSignalR` adiciona os serviços necessários para a camada de injeção de dependência de núcleo do ASP.NET (DI) para dar suporte ao SignalR. Na `Startup.Configure`, o `UseSignalR` método é chamado para ligar os pontos de extremidade de Hub no pipeline de solicitação do ASP.NET Core.

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddSignalR();
        services.AddHostedService<Worker>();
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.UseSignalR((routes) =>
        {
            routes.MapHub<ClockHub>("/hubs/clock");
        });
    }
}
```

No exemplo anterior, o `ClockHub` classe implementa o `Hub<T>` classe para criar um Hub com rigidez de tipos. O `ClockHub` tiver sido configurado na `Startup` classe responder às solicitações no ponto de extremidade `/hubs/clock`.

Para obter mais informações sobre os Hubs com rigidez de tipos, consulte [usando os hubs de SignalR do ASP.NET Core](#).

#### NOTE

Essa funcionalidade não está limitada para o `Hub<T>` classe. Qualquer classe que herda de `Hub`, como `DynamicHub`, também funcionará.

```
public class ClockHub : Hub<IClock>
{
    public async Task SendTimeToClients(DateTime dateTime)
    {
        await Clients.All.ShowTime(dateTime);
    }
}
```

A interface usada pelo fortemente tipados `ClockHub` é o `IClock` interface.

```
public interface IClock
{
    Task ShowTime(DateTime currentTime);
}
```

## Chamar um SignalR Hub de um serviço em segundo plano

Durante a inicialização, o `Worker` classe, uma `BackgroundService`, está conectada usando `AddHostedService`.

```
services.AddHostedService<Worker>();
```

Uma vez que o SignalR também está conectado durante o `Startup` fase, em que cada Hub está anexado a um ponto de extremidade individual no pipeline de solicitação HTTP do ASP.NET Core, cada Hub é representado por um `IHubContext<T>` no servidor. Usando o ASP.NET Core injeção de dependência recursos, outras classes instanciadas por meio da camada de hospedagem, como `BackgroundService` classes, classes de controlador MVC ou modelos de página do Razor, podem obter referências para os Hubs do lado do servidor, aceitando as instâncias de `IHubContext<ClockHub, IClock>` durante a construção.

```
public class Worker : BackgroundService
{
    private readonly ILogger<Worker> _logger;
    private readonly IHubContext<ClockHub, IClock> _clockHub;

    public Worker(ILogger<Worker> logger, IHubContext<ClockHub, IClock> clockHub)
    {
        _logger = logger;
        _clockHub = clockHub;
    }

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            _logger.LogInformation($"Worker running at: {DateTime.Now}");
            await _clockHub.Clients.All.ShowTime(DateTime.Now);
            await Task.Delay(1000);
        }
    }
}
```

Como o `ExecuteAsync` método é chamado de forma iterativa no serviço do plano de fundo, o servidor data e hora

atuais são enviadas para os clientes conectados usando o `ClockHub`.

## Reagir a eventos do SignalR com serviços em segundo plano

Como um aplicativo de página única usando o cliente JavaScript para um aplicativo de área de trabalho do .NET ou SignalR pode fazer usando o usando o [Cliente de .NET do SignalR do ASP.NET Core](#), um `BackgroundService` ou `IHostedService` implementação também pode ser usada para conectar-se aos Hubs de SignalR e responder a eventos.

O `ClockHubClient` classe implementa ambos o `IClock` interface e o `IHostedService` interface. Dessa forma, ele pode ser conectado durante `Startup` para executar continuamente e responder a eventos do Hub do servidor.

```
public partial class ClockHubClient : IClock, IHostedService
{
}
```

Durante a inicialização, o `ClockHubClient` cria uma instância de um `HubConnection` e conecta os `IClock.ShowTime` método como o manipulador para o Hub `ShowTime` eventos.

```
private readonly ILogger<ClockHubClient> _logger;
private HubConnection _connection;

public ClockHubClient(ILogger<ClockHubClient> logger)
{
    _logger = logger;

    _connection = new HubConnectionBuilder()
        .WithUrl(Strings.HubUrl)
        .Build();

    _connection.On<DateTime>(Strings.Events.TimeSent,
        dateTime => _ = ShowTime(dateTime));
}

public Task ShowTime(DateTime currentTime)
{
    _logger.LogInformation($"{currentTime.ToShortTimeString()}");

    return Task.CompletedTask;
}
```

No `IHostedService.StartAsync` implementação, o `HubConnection` é iniciado de forma assíncrona.

```
public async Task StartAsync(CancellationToken cancellationToken)
{
    // Loop is here to wait until the server is running
    while (true)
    {
        try
        {
            await _connection.StartAsync(cancellationToken);

            break;
        }
        catch
        {
            await Task.Delay(1000);
        }
    }
}
```

Durante o `IHostedService.StopAsync` método, o `HubConnection` é descartado de forma assíncrona.

```
public Task StopAsync(CancellationToken cancellationToken)
{
    return _connection.DisposeAsync();
}
```

## Recursos adicionais

- [Introdução](#)
- [Hubs](#)
- [Publicar no Azure](#)
- [Hubs com rigidez de tipos](#)

# Configuração do ASP.NET SignalR Core

08/02/2019 • 19 minutes to read • [Edit Online](#)

## Opções de serialização JSON/MessagePack

SignalR do ASP.NET Core dá suporte a dois protocolos para codificação de mensagens: [JSON](#) e [MessagePack](#). Cada protocolo tem opções de configuração de serialização.

Serialização JSON pode ser configurada no servidor usando o [AddJsonProtocol](#) método de extensão, que pode ser adicionado após [AddSignalR](#) em seu `Startup.ConfigureServices` método. O `AddJsonProtocol` leva um delegado que recebe um `options` objeto. O [PayloadSerializerSettings](#) propriedade desse objeto é um [JSON.NET JsonSerializerSettings](#) objeto que pode ser usado para configurar a serialização de argumentos e valores de retorno. Consulte a [documentação do JSON.NET](#) para obter mais detalhes.

Por exemplo, para configurar o serializador para usar nomes de propriedade "PascalCase", em vez dos nomes de "camelCase" padrão, use o seguinte código:

```
services.AddSignalR()
    .AddJsonProtocol(options => {
        options.PayloadSerializerSettings.ContractResolver =
            new DefaultContractResolver();
    });
}
```

No cliente do .NET, o mesmo `AddJsonProtocol` método de extensão existe no [HubConnectionBuilder](#). O `Microsoft.Extensions.DependencyInjection` namespace deve ser importado para resolver o método de extensão:

```
// At the top of the file:
using Microsoft.Extensions.DependencyInjection;

// When constructing your connection:
var connection = new HubConnectionBuilder()
    .AddJsonProtocol(options => {
        options.PayloadSerializerSettings.ContractResolver =
            new DefaultContractResolver();
    })
    .Build();
```

### NOTE

Não é possível configurar a serialização JSON no cliente JavaScript neste momento.

## Opções de serialização MessagePack

MessagePack serialização pode ser configurada fornecendo um delegado para o [AddMessagePackProtocol](#) chamar. Ver [MessagePack no SignalR](#) para obter mais detalhes.

### NOTE

Não é possível configurar a serialização de MessagePack no cliente JavaScript neste momento.

## Configurar opções de servidor

A tabela a seguir descreve as opções de configuração hubs do SignalR:

OPÇÃO	VALOR PADRÃO	DESCRIÇÃO
<code>ClientTimeoutInterval</code>	30 segundos	O servidor irá considerar o cliente desconectado se ele ainda não recebeu uma mensagem (incluindo keep-alive) nesse intervalo. Ele pode demorar mais que esse intervalo de tempo limite para o cliente, na verdade, ser marcado como desconectado devido a como isso é implementado. O valor recomendado é double o <code>KeepAliveInterval</code> valor.
<code>HandshakeTimeout</code>	15 segundos	Se o cliente não envia uma mensagem de handshake inicial dentro deste intervalo de tempo, a conexão será fechada. Isso é uma configuração avançada que deve ser modificada apenas se os erros de tempo limite de handshake estiverem ocorrendo devido à latência de rede graves. Para obter mais detalhes sobre o processo de handshake, consulte a <a href="#">especificação de protocolo de Hub do SignalR</a> .
<code>KeepAliveInterval</code>	15 segundos	Se o servidor não enviou uma mensagem dentro deste intervalo, uma mensagem de ping é enviada automaticamente para manter a conexão aberta. Ao alterar <code>KeepAliveInterval</code> , altere o <code>ServerTimeout</code> / <code>serverTimeoutInMilliseconds</code> configuração no cliente. Recomendado <code>ServerTimeout</code> / <code>serverTimeoutInMilliseconds</code> valor é double o <code>KeepAliveInterval</code> valor.
<code>SupportedProtocols</code>	Todos os protocolos	Protocolos com suporte por esse hub. Por padrão, todos os protocolos registrados no servidor são permitidos, mas protocolos podem ser removidos dessa lista para desabilitar os protocolos específicos para hubs individuais.
<code>EnableDetailedErrors</code>	<code>false</code>	Se <code>true</code> detalhada mensagens de exceção são retornadas aos clientes quando uma exceção é gerada em um método de Hub. O padrão é <code>false</code> , conforme essas mensagens de exceção podem conter informações confidenciais.

Opções podem ser configuradas para todos os hubs, fornecendo um delegado de opções para o `AddSignalR` chamar em `Startup.ConfigureServices`.

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddSignalR(hubOptions =>
    {
        hubOptions.EnableDetailedErrors = true;
        hubOptions.KeepAliveInterval = TimeSpan.FromMinutes(1);
    });
}

```

Opções para um único hub substituem as opções de globais fornecidas no `AddSignalR` e pode ser configurado usando `AddHubOptions<T>`:

```

services.AddSignalR().AddHubOptions<MyHub>(options =>
{
    options.EnableDetailedErrors = true;
});

```

## Opções avançadas de configuração de HTTP

Use `HttpConnectionDispatcherOptions` para definir configurações avançadas relacionadas a transportes e gerenciamento de buffer de memória. Essas opções são configuradas passando um delegado para `MapHub<T>` em `Startup.Configure`.

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseSignalR((configure) =>
    {
        var desiredTransports =
            HttpTransportType.WebSockets |
            HttpTransportType.LongPolling;

        configure.MapHub<MyHub>("/myhub", (options) =>
        {
            options.Transports = desiredTransports;
        });
    });
}

```

A tabela a seguir descreve opções para configurar opções avançadas de HTTP do SignalR do ASP.NET Core:

OPÇÃO	VALOR PADRÃO	DESCRIÇÃO
<code>ApplicationMaxBufferSize</code>	32 KB	O número máximo de bytes recebidos do cliente que os buffers de servidor. Aumentar esse valor permite que o servidor receber mensagens maiores, mas pode afetar negativamente o consumo de memória.
<code>AuthorizationData</code>	Dados coletados automaticamente a partir de <code>Authorize</code> atributos aplicados à classe Hub.	Uma lista dos <code>IAuthorizeData</code> objetos usados para determinar se um cliente está autorizado a conectar-se ao hub.

OPÇÃO	VALOR PADRÃO	DESCRIÇÃO
<code>TransportMaxBufferSize</code>	32 KB	O número máximo de bytes enviados pelo aplicativo que os buffers de servidor. Aumentar esse valor permite que o servidor enviar mensagens maiores, mas pode afetar negativamente o consumo de memória.
<code>Transports</code>	Todos os transportes estão habilitados.	Um bitmask de <code>HttpTransportType</code> valores que possam restringir os transportes de um cliente pode usar para se conectar.
<code>LongPolling</code>	Veja a seguir.	Opções adicionais específicas para o transporte de sondagem longa.
<code>WebSockets</code>	Veja a seguir.	Opções adicionais específicas para o transporte de WebSockets.

O transporte de sondagem longa tem opções adicionais que podem ser configuradas usando o `LongPolling` propriedade:

OPÇÃO	VALOR PADRÃO	DESCRIÇÃO
<code>PollTimeout</code>	90 segundos	A quantidade máxima de tempo o servidor aguarda uma mensagem para enviar ao cliente antes de encerrar uma solicitação de pesquisa única. Diminuir esse valor faz com que o cliente emitir novas solicitações de sondagem com mais frequência.

O transporte de WebSocket tem opções adicionais que podem ser configuradas usando o `WebSockets` propriedade:

OPÇÃO	VALOR PADRÃO	DESCRIÇÃO
<code>CloseTimeout</code>	5 segundos	Depois de fechar o servidor, se o cliente não conseguir fechar dentro deste intervalo de tempo, a conexão é encerrada.
<code>SubProtocolSelector</code>	<code>null</code>	Um delegado que pode ser usado para definir o <code>Sec-WebSocket-Protocol</code> cabeçalho para um valor personalizado. O delegado recebe os valores solicitados pelo cliente como entrada e deve retornar o valor desejado.

## Configurar opções do cliente

Opções de cliente podem ser configuradas na `HubConnectionBuilder` (disponível em clientes .NET e JavaScript), do tipo, bem como no `HubConnection` em si.

### Configurar o registro em log

O log está configurado no cliente do .NET usando o `ConfigureLogging` método. Registro em log provedores e

filtros pode ser registrado da mesma forma como estão no servidor. Consulte a [entrar no ASP.NET Core](#) documentação para obter mais informações.

#### NOTE

Para registrar os provedores de log, você deve instalar os pacotes necessários. Consulte a [provedores de log internos](#) seção do docs para obter uma lista completa.

Por exemplo, para habilitar o log de Console, instale o `Microsoft.Extensions.Logging.Console` pacote do NuGet. Chamar o `AddConsole` método de extensão:

```
var connection = new HubConnectionBuilder()
    .WithUrl("https://example.com/myhub")
    .ConfigureLogging(logging => {
        logging.SetMinimumLevel(LogLevel.Information);
        logging.AddConsole();
    })
    .Build();
```

No cliente JavaScript, um semelhante `configureLogging` existe um método. Forneça um `LogLevel` valor que indica o nível mínimo de mensagens de log para produzir. Os logs são gravados na janela de console do navegador.

```
let connection = new signalR.HubConnectionBuilder()
    .withUrl("/myhub")
    .configureLogging(signalR.LogLevel.Information)
    .build();
```

#### NOTE

Para desabilitar o registro em log totalmente, especifique `signalR.LogLevel.None` no `configureLogging` método.

Níveis de log disponíveis para o cliente JavaScript estão listados abaixo. Definindo o nível de log para um desses valores habilita o log de mensagens no **ou superior** desse nível.

NÍVEL	DESCRIÇÃO
<code>None</code>	Nenhuma mensagem será registrada.
<code>Critical</code>	Mensagens que indicam uma falha em todo o aplicativo.
<code>Error</code>	Mensagens que indicam uma falha na operação atual.
<code>Warning</code>	Mensagens que indicam um problema de não-fatais.
<code>Information</code>	Mensagens informativas.
<code>Debug</code>	Mensagens de diagnóstico útil para depuração.
<code>Trace</code>	Mensagens de diagnóstico muito detalhadas projetadas para diagnosticar problemas específicos.

## Configure transportes permitidos

Os transportes usados pelo SignalR podem ser configurados na `WithUrl` chamar (`withUrl` em JavaScript). Um bitwise OR dos valores de `HttpTransportType` pode ser usado para restringir o cliente para usar somente os transportes especificados. Todos os transportes são habilitados por padrão.

Por exemplo, para desabilitar o transporte de eventos do Server-Sent, mas permita conexões WebSocket e sondagem longa:

```
var connection = new HubConnectionBuilder()
    .WithUrl("https://example.com/myhub", HttpTransportType.WebSockets | HttpTransportType.LongPolling)
    .Build();
```

No cliente JavaScript, transportes são configurados, definindo o `transport` campo no objeto de opções fornecido ao `withUrl`:

```
let connection = new signalR.HubConnectionBuilder()
    .withUrl("/myhub", { transport: signalR.HttpTransportType.WebSockets |
        signalR.HttpTransportType.LongPolling })
    .build();
```

## Configurar a autenticação do portador

Para fornecer dados de autenticação junto com as solicitações de SignalR, use o `AccessTokenProvider` opção (`accessTokenFactory` em JavaScript) para especificar uma função que retorna o token de acesso desejado. No cliente do .NET, esse token de acesso é passado como um HTTP "Autenticação do portador" token (usando o `Authorization` cabeçalho com um tipo de `Bearer`). No cliente JavaScript, o token de acesso é usado como um token de portador **exceto** em alguns casos em que o navegador APIs restringir a capacidade de aplicar cabeçalhos (especificamente, em solicitações de eventos do Server-sent e WebSockets). Nesses casos, o token de acesso é fornecido como um valor de cadeia de caracteres de consulta `access_token`.

No cliente do .NET, o `AccessTokenProvider` opção pode ser especificada usando o delegado de opções no `WithUrl`:

```
var connection = new HubConnectionBuilder()
    .WithUrl("https://example.com/myhub", options => {
        options.AccessTokenProvider = async () => {
            // Get and return the access token.
        };
    })
    .Build();
```

No cliente JavaScript, o token de acesso é configurado definindo a `accessTokenFactory` campo no objeto de opções no `withUrl`:

```
let connection = new signalR.HubConnectionBuilder()
    .withUrl("/myhub", {
        accessTokenFactory: () => {
            // Get and return the access token.
            // This function can return a JavaScript Promise if asynchronous
            // logic is required to retrieve the access token.
        }
    })
    .build();
```

## Configurar tempo limite e opções de keep alive

Opções adicionais para configurar o tempo limite e o comportamento de keep alive estão disponíveis no `HubConnection` objeto em si:

OPÇÃO DE .NET	OPÇÃO DE JAVASCRIPT	VALOR PADRÃO	DESCRIÇÃO
<code>ServerTimeout</code>	<code>serverTimeoutInMilliseconds</code>	30 segundos (30.000 milissegundos)	Tempo limite para a atividade do servidor. Se o servidor não enviou uma mensagem nesse intervalo, o cliente considera o servidor desconectado e gatilhos do <code>Closed</code> evento ( <code>onclose</code> em JavaScript). Esse valor deve ser grande o suficiente para uma mensagem de ping a serem enviados do servidor e recebida pelo cliente dentro do intervalo de tempo limite. O valor recomendado é um número pelo menos duas vezes o servidor <code>KeepAliveInterval</code> valor, para dar tempo para pings de chegada.
<code>HandshakeTimeout</code>	Não é configurável	15 segundos	Tempo limite para o handshake inicial do servidor. Se o servidor não enviar uma resposta de handshake nesse intervalo, o cliente cancela o handshake e gatilhos do <code>Closed</code> evento ( <code>onclose</code> em JavaScript). Isso é uma configuração avançada que deve ser modificada apenas se os erros de tempo limite de handshake estiverem ocorrendo devido à latência de rede graves. Para obter mais detalhes sobre o processo de Handshake, consulte a <a href="#">especificação de protocolo de Hub do SignalR</a> .

No cliente do .NET, os valores de tempo limite são especificados como `TimeSpan` valores. No cliente JavaScript, os valores de tempo limite são especificados como um número que indica a duração em milissegundos.

### Configurar opções adicionais

Opções adicionais podem ser configuradas na `WithUrl` (`withUrl` em JavaScript) método em

`HubConnectionBuilder` :

OPÇÃO DE .NET	OPÇÃO DE JAVASCRIPT	VALOR PADRÃO	DESCRIÇÃO
<code>AccessTokenProvider</code>	<code>accessTokenFactory</code>	<code>null</code>	Uma função que retorna uma cadeia de caracteres que é fornecida como um token de autenticação de portador em solicitações HTTP.

OPÇÃO DE .NET	OPÇÃO DE JAVASCRIPT	VALOR PADRÃO	DESCRIÇÃO
SkipNegotiation	skipNegotiation	false	Defina isso como <code>true</code> para ignorar a etapa de negociação. <b>Só tem suporte quando o transporte de WebSockets é o único transporte habilitado.</b> Essa configuração não pode ser habilitada ao usar o serviço do Azure SignalR.
ClientCertificates	Não é configurável *	Vazio	Uma coleção de certificados TLS para enviar para autenticar solicitações.
Cookies	Não é configurável *	Vazio	Uma coleção de cookies HTTP a ser enviado com cada solicitação HTTP.
Credentials	Não é configurável *	Vazio	Credenciais devem ser enviados com cada solicitação HTTP.
CloseTimeout	Não é configurável *	5 segundos	WebSockets. A quantidade máxima de tempo que o cliente aguardará depois de fechamento para o servidor confirmar a solicitação de fechamento. Se o servidor não reconhece o fechamento dentro desse período, o cliente se desconecta.
Headers	Não é configurável *	Vazio	Um dicionário de cabeçalhos HTTP adicionais para enviar com todas as solicitações HTTP.

OPÇÃO DE .NET	OPÇÃO DE JAVASCRIPT	VALOR PADRÃO	DESCRIÇÃO
<code>HttpMessageHandlerFactory</code>	Não é configurável *	<code>null</code>	Um delegado que pode ser usado para configurar ou substituir o <code>HttpMessageHandler</code> usado para enviar solicitações HTTP. Não é usado para conexões de WebSocket. Esse delegado deve retornar um valor não nulo e recebe o valor padrão como um parâmetro. Modificar as configurações nesse valor padrão e retorná-lo ou retornar um novo <code>HttpMessageHandler</code> instância. <b>Ao substituir o manipulador Certifique-se de copiar as configurações que você deseja impedir que o manipulador fornecido, caso contrário, as opções configuradas (como cabeçalhos e Cookies) não se aplicam ao novo manipulador.</b>
<code>Proxy</code>	Não é configurável *	<code>null</code>	Um proxy HTTP a ser usado ao enviar solicitações HTTP.
<code>UseDefaultCredentials</code>	Não é configurável *	<code>false</code>	Defina esse valor booleano para enviar as credenciais padrão para solicitações HTTP e WebSockets. Isso permite que o uso da autenticação do Windows.
<code>WebSocketConfiguration</code>	Não é configurável *	<code>null</code>	Um delegado que pode ser usado para configurar opções adicionais de WebSocket. Recebe uma instância do <code>ClientWebSocketOptions</code> que pode ser usado para configurar as opções.

Opções marcadas com um asterisco (\*) não são configuráveis no cliente JavaScript, devido a limitações no navegador APIs.

No cliente do .NET, essas opções podem ser modificadas pelo delegado opções fornecido para `WithUrl`:

```
var connection = new HubConnectionBuilder()
    .WithUrl("https://example.com/myhub", options => {
        options.Headers["Foo"] = "Bar";
        options.Cookies.Add(new Cookie(/* ... */));
        options.ClientCertificates.Add(/* ... */);
    })
    .Build();
```

No cliente JavaScript, essas opções podem ser fornecidas em um objeto JavaScript fornecido para `withUrl`:

```
let connection = new signalR.HubConnectionBuilder()
    .withUrl("/myhub", {
        skipNegotiation: true,
        transport: signalR.HttpTransportType.WebSockets
    })
    .build();
```

## Recursos adicionais

- [Introdução ao SignalR para ASP.NET Core](#)
- [Usando os hubs de SignalR do ASP.NET Core](#)
- [ASP.NET Core SignalR JavaScript cliente](#)
- [Cliente de .NET do SignalR do ASP.NET Core](#)
- [Usar o protocolo de MessagePack Hub no SignalR do ASP.NET Core](#)
- [Plataformas com suporte do SignalR do ASP.NET Core](#)

# Autenticação e autorização no SignalR do ASP.NET Core

02/02/2019 • 11 minutes to read • [Edit Online](#)

Por [Andrew Stanton-Nurse](#)

[Exibir ou baixar o código de exemplo \(como fazer o download\)](#)

## Autenticar usuários que se conectam a um hub SignalR

O SignalR pode ser usado com [autenticação do ASP.NET Core](#) para associar um usuário a cada conexão. Em um hub, os dados de autenticação podem ser acessados do `HubConnectionContext.User` propriedade. A autenticação permite que o hub para chamar métodos em todas as conexões associadas a um usuário (consulte [gerenciar usuários e grupos no SignalR](#) para obter mais informações). Várias conexões podem estar associadas a um único usuário.

### Autenticação de cookie

Em um aplicativo baseado em navegador, a autenticação de cookie permite que suas credenciais de usuário existentes fluir automaticamente para conexões do SignalR. Ao usar o cliente de navegador, nenhuma configuração adicional é necessária. Se o usuário está conectado ao seu aplicativo, a conexão do SignalR herda automaticamente essa autenticação.

Os cookies são uma maneira de específicas do navegador para enviar os tokens de acesso, mas os clientes sem navegador podem enviá-los. Ao usar o `cliente .NET`, o `Cookies` propriedade pode ser configurada no `.WithUrl` chamada para fornecer um cookie. No entanto, usando a autenticação de cookie do cliente .NET requer que o aplicativo para fornecer uma API para trocar dados de autenticação para um cookie.

### Autenticação de token de portador

O cliente pode fornecer um token de acesso em vez de usar um cookie. O servidor valida o token e usa-o para identificar o usuário. Essa validação é feita somente quando a conexão é estabelecida. Durante a vida da conexão, o servidor não automaticamente revalidar para verificar a revogação do token.

No servidor de autenticação de token de portador é configurada usando o [middleware de portador de JWT](#).

No cliente JavaScript, o token pode ser fornecido usando o `accessTokenFactory` opção.

```
this.connection = new signalR.HubConnectionBuilder()
    .withUrl("/hubs/chat", { accessTokenFactory: () => this.loginToken })
    .build();
```

No cliente do .NET, há um semelhante [AccessTokenProvider](#) propriedade que pode ser usada para configurar o token:

```
var connection = new HubConnectionBuilder()
    .WithUrl("https://example.com/myhub", options =>
    {
        options.AccessTokenProvider = () => Task.FromResult(_myAccessToken);
    })
    .Build();
```

## NOTE

A função de token de acesso que você fornecer é chamada antes de **cada** solicitação HTTP feita pelo SignalR. Se você precisa renovar o token para manter a conexão ativa (porque ele pode expirar durante a conexão), faça isso dentro dessa função e retornar o token atualizado.

APIs da web padrão, os tokens de portador são enviados em um cabeçalho HTTP. No entanto, o SignalR é não é possível definir esses cabeçalhos em navegadores quando usando alguns transportes. Ao usar WebSockets e eventos do Server-Sent, o token é transmitido como um parâmetro de cadeia de caracteres de consulta. Para suportar isso no servidor, a configuração adicional é necessária:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();

    services.AddAuthentication(options =>
    {
        // Identity made Cookie authentication the default.
        // However, we want JWT Bearer Auth to be the default.
        options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
        options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    })
    .AddJwtBearer(options =>
    {
        // Configure JWT Bearer Auth to expect our security key
        options.TokenValidationParameters =
            new TokenValidationParameters
            {
                LifetimeValidator = (before, expires, token, param) =>
                {
                    return expires > DateTime.UtcNow;
                },
                ValidateAudience = false,
                ValidateIssuer = false,
                ValidateActor = false,
                ValidateLifetime = true,
                IssuerSigningKey = SecurityKey
            };

        // We have to hook the OnMessageReceived event in order to
        // allow the JWT authentication handler to read the access
        // token from the query string when a WebSocket or
        // Server-Sent Events request comes in.
        options.Events = new JwtBearerEvents
        {
            OnMessageReceived = context =>
            {
                var accessToken = context.Request.Query["access_token"];

                // If the request is for our hub...
                var path = context.HttpContext.Request.Path;
                if (!string.IsNullOrEmpty(accessToken) &&
                    (path.StartsWithSegments("/hubs/chat")))
                {
                    // Read the token out of the query string
                    context.Token = accessToken;
                }
                return Task.CompletedTask;
            }
        };
    });
}
```

```

    };

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    services.AddSignalR();

    // Change to use Name as the user identifier for SignalR
    // WARNING: This requires that the source of your JWT token
    // ensures that the Name claim is unique!
    // If the Name claim isn't unique, users could receive messages
    // intended for a different user!
    services.AddSingleton<IUserIdProvider, NameUserIdProvider>();

    // Change to use email as the user identifier for SignalR
    // services.AddSingleton<IUserIdProvider, EmailBasedUserIdProvider>();

    // WARNING: use *either* the NameUserIdProvider *or* the
    // EmailBasedUserIdProvider, but do not use both.
}

```

## Cookies versus os tokens de portador

Como os cookies são específicos para navegadores, enviá-las de outros tipos de clientes adiciona complexidade em comparação comparada enviar tokens de portador. Por esse motivo, a autenticação de cookie não é recomendada, a menos que o aplicativo precisa apenas para autenticar usuários de cliente de navegador. Autenticação de token de portador é a abordagem recomendada ao usar os clientes que não seja o cliente do navegador.

## Autenticação do Windows

Se [autenticação do Windows](#) é configurado em seu aplicativo, o SignalR pode usar essa identidade para proteger hubs. No entanto, para enviar mensagens para usuários individuais, você precisará adicionar um provedor de ID de usuário personalizado. Isso ocorre porque o sistema de autenticação do Windows não fornece a declaração "Identificador de nome" que usa o SignalR para determinar o nome de usuário.

Adicione uma nova classe que implementa `IUserIdProvider` e recuperar uma das declarações de usuário a ser usado como o identificador. Por exemplo, para usar a declaração de "Nome" (que é o nome de usuário do Windows na forma `[Domain]\[Username]` ), crie a seguinte classe:

```

public class NameUserIdProvider : IUserIdProvider
{
    public string GetUserId(HubConnectionContext connection)
    {
        return connection.User?.Identity?.Name;
    }
}

```

Em vez de `ClaimTypes.Name`, você pode usar qualquer valor entre o `User` (como o identificador do SID do Windows, etc.).

### NOTE

O valor escolhido deve ser exclusivo entre todos os usuários em seu sistema. Caso contrário, uma mensagem para um usuário poderia acabar indo para um usuário diferente.

Registrar esse componente em seu `startup.ConfigureServices` método.

```
public void ConfigureServices(IServiceCollection services)
{
    // ... other services ...

    services.AddSignalR();
    services.AddSingleton<IUserIdProvider, NameUserIdProvider>();
}
```

No cliente do .NET, a autenticação do Windows deve ser habilitada definindo o [UseDefaultCredentials](#) propriedade:

```
var connection = new HubConnectionBuilder()
    .WithUrl("https://example.com/myhub", options =>
{
    options.UseDefaultCredentials = true;
})
.Build();
```

Autenticação do Windows só tem suporte pelo cliente do navegador ao usar o Microsoft Internet Explorer ou Microsoft Edge.

### Uso de declarações para personalizar o tratamento de identidade

Um aplicativo que autentica usuários pode derivar as IDs de usuário do SignalR de declarações de usuário. Para especificar como o SignalR cria IDs de usuário, implementar [IUserIdProvider](#) e registrar a implementação.

O código de exemplo demonstra como você usaria declarações para selecionar o endereço de email do usuário como a propriedade de identificação.

#### NOTE

O valor escolhido deve ser exclusivo entre todos os usuários em seu sistema. Caso contrário, uma mensagem para um usuário poderia acabar indo para um usuário diferente.

```
public class EmailBasedUserIdProvider : IUserIdProvider
{
    public virtual string GetUserId(HubConnectionContext connection)
    {
        return connection.User?.FindFirst(ClaimTypes.Email)?.Value;
    }
}
```

O registro de conta adiciona uma declaração com o tipo [ClaimsTypes.Email](#) no banco de dados de identidade do ASP.NET.

```
// create a new user
var user = new ApplicationUser { UserName = Input.Email, Email = Input.Email };
var result = await _userManager.CreateAsync(user, Input.Password);

// add the email claim and value for this user
await _userManager.AddClaimAsync(user, new Claim(ClaimTypes.Email, Input.Email));
```

Registrar esse componente em seu [Startup.ConfigureServices](#).

```
services.AddSingleton<IUserIdProvider, EmailBasedUserIdProvider>();
```

## Autorizar usuários para acesso hubs e métodos de hub

Por padrão, todos os métodos em um hub podem ser chamados por um usuário não autenticado. Para exigir autenticação, se aplicam a [autorizar](#) de atributo para o hub:

```
[Authorize]  
public class ChatHub: Hub  
{  
}
```

Você pode usar os argumentos de construtor e propriedades do `[Authorize]` atributo para restringir o acesso somente para usuários específicos de correspondência [políticas de autorização](#). Por exemplo, se você tiver uma política de autorização personalizada chamada `MyAuthorizationPolicy` pode garantir que somente os usuários dessa política de correspondência podem acessar o hub usando o seguinte código:

```
[Authorize("MyAuthorizationPolicy")]  
public class ChatHub: Hub  
{  
}
```

Métodos de hub individuais podem ter o `[Authorize]` atributo aplicado também. Se o usuário atual não corresponder a política aplicada ao método, um erro será retornado ao chamador:

```
[Authorize]  
public class ChatHub: Hub  
{  
    public async Task Send(string message)  
    {  
        // ... send a message to all users ...  
    }  
  
    [Authorize("Administrators")]  
    public void BanUser(string userName)  
    {  
        // ... ban a user from the chat room (something only Administrators can do) ...  
    }  
}
```

## Recursos adicionais

- [Autenticação de Token de portador no ASP.NET Core](#)

# Considerações de segurança no SignalR do ASP.NET Core

24/01/2019 • 9 minutes to read • [Edit Online](#)

Por [Andrew Stanton-Nurse](#)

Este artigo fornece informações sobre como proteger o SignalR.

## Compartilhamento de recursos entre origens

[Recursos entre origens \(CORS\) compartilhamento](#) pode ser usado para permitir conexões do SignalR entre origens no navegador. Se o código JavaScript é hospedado em um domínio diferente do aplicativo do SignalR, [middleware CORS](#) deve estar habilitado para permitir que o JavaScript para se conectar ao aplicativo SignalR. Permitir solicitações entre origens de domínios que confiáveis ou controle. Por exemplo:

- O site é hospedado em `http://www.example.com`
- Seu aplicativo SignalR é hospedado em `http://signalr.example.com`

CORS devem ser configurado no aplicativo do SignalR para permitir somente a origem `www.example.com`.

Para obter mais informações sobre como configurar o CORS, consulte [habilitar solicitações de entre origens \(CORS\)](#). O SignalR **requer** as seguintes políticas CORS:

- Permitir que as origens esperadas específicas. Permitir qualquer origem é possível, mas está **não** segura ou recomendada.
- Métodos HTTP `GET` e `POST` devem ser permitidos.
- Credenciais devem ser habilitadas, mesmo quando a autenticação não é usada.

Por exemplo, a seguinte política CORS permite que um cliente de navegador do SignalR hospedado no `https://example.com` para acessar o aplicativo de SignalR hospedado em `https://signalr.example.com`:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    // ... other middleware ...

    // Make sure the CORS middleware is ahead of SignalR.
    app.UseCors(builder =>
    {
        builder.WithOrigins("https://example.com")
            .AllowAnyHeader()
            .WithMethods("GET", "POST")
            .AllowCredentials();
    });

    // ... other middleware ...

    app.UseSignalR(routes =>
    {
        routes.MapHub<ChatHub>("/chatHub");
    });

    // ... other middleware ...
}
```

#### NOTE

O SignalR não é compatível com o recurso interno de CORS no serviço de aplicativo do Azure.

## WebSocket Origin Restriction

As proteções fornecidas pelo CORS não se aplicam ao WebSockets. Para a restrição de origem sobre WebSockets, leia [restrição de origem de WebSockets](#).

As proteções fornecidas pelo CORS não se aplicam ao WebSockets. Navegadores **não**:

- Executam solicitações de simulação de CORS.
- Respeitam as restrições especificadas em cabeçalhos `Access-Control` ao fazer solicitações de WebSocket.

No entanto, os navegadores enviam o cabeçalho `Origin` ao emitir solicitações de WebSocket. Os aplicativos devem ser configurados para validar esses cabeçalhos e garantir que apenas WebSockets provenientes de origens esperadas sejam permitidos.

No ASP.NET Core 2.1 e posterior, validação de cabeçalho pode ser obtida usando um middleware personalizado colocado **antes de** `UseSignalR` **e o middleware de autenticação** em `Configure`:

```

// In Startup, add a static field listing the allowed Origin values:
private static readonly HashSet<string> _allowedOrigins = new HashSet<string>()
{
    // Add allowed origins here. For example:
    "https://www.mysite.com",
    "https://mysite.com",
};

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    // ... other middleware ...

    // Validate Origin header on WebSocket requests to prevent unexpected cross-site
    // WebSocket requests.
    app.Use((context, next) =>
    {
        // Check for a WebSocket request.
        if (string.Equals(context.Request.Headers["Upgrade"], "websocket"))
        {
            var origin = context.Request.Headers["Origin"];

            // If there is an origin header, and the origin header doesn't match
            // an allowed value:
            if (!string.IsNullOrEmpty(origin) && !_allowedOrigins.Contains(origin))
            {
                // The origin is not allowed, reject the request
                context.Response.StatusCode = StatusCodes.Status403Forbidden;
                return Task.CompletedTask;
            }
        }

        // The request is a valid Origin or not a WebSocket request, so continue.
        return next();
    });

    // ... other middleware ...

    app.UseSignalR(routes =>
    {
        routes.MapHub<ChatHub>("/chatHub");
    });

    // ... other middleware ...
}

```

#### NOTE

O cabeçalho `Origin` é controlado pelo cliente e, como o cabeçalho `Referer`, pode ser falsificado. Esses cabeçalhos devem **não** ser usado como um mecanismo de autenticação.

## Log de token de acesso

Ao usar WebSockets ou Server-Sent eventos, o cliente de navegador envia o token de acesso na cadeia de caracteres de consulta. Receber o token de acesso por meio da cadeia de caracteres de consulta é geralmente tão seguro quanto usar o padrão `Authorization` cabeçalho. Você sempre deve usar HTTPS para garantir uma conexão segura de ponta a ponta entre o cliente e o servidor. A URL para cada solicitação, incluindo a cadeia de caracteres de consulta de log de muitos servidores web. Registro em log as URLs pode registrar o token de acesso. ASP.NET Core registra a URL para cada solicitação, por padrão, o que incluirá a cadeia de caracteres de consulta. Por exemplo:

```
info: Microsoft.AspNetCore.Hosting.Internal.WebHost[1]
      Request starting HTTP/1.1 GET http://localhost:5000/myhub?access_token=1234
```

Se você tiver dúvidas sobre o log de dados com os logs do servidor, você pode desabilitar esse log inteiramente, configurando o `Microsoft.AspNetCore.Hosting` agente para o `Warning` nível ou superior (essas mensagens são gravadas em `Info` nível). Consulte a documentação sobre [filtragem de Log](#) para obter mais informações. Se você ainda quiser registrar determinadas informações de solicitação, você poderá [escrever um middleware](#) para registrar os dados necessários e filtrar o `access_token` valor de cadeia de caracteres de consulta (se presente).

## Exceções

Mensagens de exceção são geralmente consideradas dados confidenciais que não devem ser revelados para um cliente. Por padrão, o SignalR não envia os detalhes de uma exceção gerada por um método de hub para o cliente. Em vez disso, o cliente recebe uma mensagem genérica indicando que ocorreu um erro. Entrega de mensagem de exceção para o cliente pode ser substituída (por exemplo, no desenvolvimento ou teste) com `EnableDetailedErrors`. Mensagens de exceção não devem ser expostas ao cliente em aplicativos de produção.

## Gerenciamento de buffer

O SignalR usa buffers por conexão para gerenciar mensagens de entrada e saídas. Por padrão, o SignalR limita esses buffers para 32 KB. A mensagem maior que um cliente ou servidor pode enviar é 32 KB. O máximo de memória consumido por uma conexão de mensagens é 32 KB. Se as mensagens são sempre menores que 32 KB, você pode reduzir o limite, que:

- Impede que um cliente que está sendo capaz de enviar uma mensagem maior.
- O servidor nunca será necessário alocar buffers grandes para aceitar mensagens.

Se suas mensagens forem maiores que 32 KB, você pode aumentar o limite. Aumentar esse limite significa:

- O cliente pode fazer com que o servidor para alocar buffers de memória grandes.
- Alocação de servidor de buffers grandes pode reduzir o número de conexões simultâneas.

Há limites para mensagens de entrada e saídas, ambos podem ser configuradas sobre o

`HttpConnectionDispatcherOptions` objeto configurado no `MapHub`:

- `ApplicationMaxBufferSize` representa o número máximo de bytes do cliente que os buffers de servidor. Se o cliente tenta enviar uma mensagem maior que esse limite, a conexão poderá ser fechada.
- `TransportMaxBufferSize` representa o número máximo de bytes que o servidor pode enviar. Se o servidor tenta enviar uma mensagem (incluindo valores de retorno de métodos de hub) maiores que esse limite, uma exceção será lançada.

Definir o limite como `0` desabilita o limite. Remover o limite permite que um cliente enviar uma mensagem de qualquer tamanho. Os clientes mal-intencionados enviando mensagens extensas podem provocar alocação de memória em excesso. Uso de memória em excesso pode reduzir significativamente o número de conexões simultâneas.

# Usar o protocolo de MessagePack Hub no SignalR do ASP.NET Core

07/02/2019 • 4 minutes to read • [Edit Online](#)

Por [Brennan Conroy](#)

Este artigo pressupõe que o leitor esteja familiarizado com os tópicos abordados [começar](#).

## O que é MessagePack?

[MessagePack](#) é um formato de serialização binária que é rápido e compacto. Ela é útil quando o desempenho e a largura de banda são uma preocupação porque cria mensagens menores em comparação comparadas [JSON](#). Porque ele é um formato binário, as mensagens são ilegíveis ao examinar os logs e rastreamentos de rede, a menos que os bytes são passados por meio de um analisador MessagePack. O SignalR tem suporte interno para o formato MessagePack e fornece APIs para o cliente e servidor usar.

## Configurar MessagePack no servidor

Para habilitar o protocolo do Hub MessagePack no servidor, instale o

`Microsoft.AspNetCore.SignalR.Protocols.MessagePack` pacote em seu aplicativo. No arquivo Startup.cs, adicione `AddMessagePackProtocol` para o `AddSignalR` chamada para habilitar o suporte de MessagePack no servidor.

### NOTE

JSON é habilitado por padrão. Adicionar MessagePack habilita o suporte para JSON e MessagePack clientes.

```
services.AddSignalR()
    .AddMessagePackProtocol();
```

Para personalizar como MessagePack formatará a seus dados, `AddMessagePackProtocol` aceita um delegado para configurar as opções. No delegado, o `FormatterResolvers` propriedade pode ser usada para configurar as opções de serialização MessagePack. Para obter mais informações sobre como funcionam os resolvedores, visite a biblioteca, MessagePack em [MessagePack-CSharp](#). Atributos podem ser usados nos objetos que você deseja serializar para definir como eles devem ser tratados.

```
services.AddSignalR()
    .AddMessagePackProtocol(options =>
{
    options.FormatterResolvers = new List<MessagePack.IFormatterResolver>()
    {
        MessagePack.Resolvers.StandardResolver.Instance
    };
});
```

## Configurar MessagePack no cliente

#### NOTE

JSON é habilitado por padrão para os clientes com suporte. Os clientes só podem dar suporte a um único protocolo. Adicionar suporte MessagePack qualquer substituirá anteriormente protocolos configurados.

### Cliente .NET

Para habilitar MessagePack no cliente .NET, instale o `Microsoft.AspNetCore.SignalR.Protocols.MessagePack` pacote e chame `AddMessagePackProtocol` em `HubConnectionBuilder`.

```
var hubConnection = new HubConnectionBuilder()
    .WithUrl("/chatHub")
    .AddMessagePackProtocol()
    .Build();
```

#### NOTE

Isso `AddMessagePackProtocol` chamada aceita um delegado para configurar opções de como o servidor.

### Cliente JavaScript

MessagePack suporte para o cliente JavaScript é fornecido pelo `@aspnet/signalr-protocol-msgpack` pacote npm.

```
npm install @aspnet/signalr-protocol-msgpack
```

Depois de instalar o pacote npm, o módulo pode ser usado diretamente por meio de um carregador de módulo de JavaScript ou importado para o navegador fazendo referência a `node_modules\@aspnet\signalr-protocol-msgpack\dist\browser\signalr-protocol-msgpack.js` arquivo. Em um navegador, o `msgpack5` biblioteca também deve ser referenciada. Use um `<script>` marca para criar uma referência. A biblioteca pode ser encontrada em `node_modules\msgpack5\dist\msgpack5.js`.

#### NOTE

Ao usar o `<script>` elemento, a ordem é importante. Se `signalr-protocol-msgpack.js` é referenciada antes `msgpack5.js`, ocorre um erro quando tentar se conectar com MessagePack. `SignalR.js` também é necessária antes `signalr-protocol-msgpack.js`.

```
<script src="~/lib/signalr/signalr.js"></script>
<script src="~/lib/msgpack5/msgpack5.js"></script>
<script src="~/lib/signalr/signalr-protocol-msgpack.js"></script>
```

Adicionando `.withHubProtocol(new signalR.protocols.msgpack.MessagePackHubProtocol())` para o `HubConnectionBuilder` irá configurar o cliente para usar o protocolo MessagePack ao se conectar a um servidor.

```
const connection = new signalR.HubConnectionBuilder()
    .withUrl("/chatHub")
    .withHubProtocol(new signalR.protocols.msgpack.MessagePackHubProtocol())
    .build();
```

**NOTE**

Neste momento, não há nenhuma opção de configuração para o protocolo MessagePack no cliente JavaScript.

## Recursos relacionados

- [Introdução](#)
- [Cliente .NET](#)
- [Cliente JavaScript](#)

# Usar o streaming em SignalR do ASP.NET Core

25/01/2019 • 7 minutes to read • [Edit Online](#)

Por [Brennan Conroy](#)

SignalR do ASP.NET Core dá suporte a streaming valores de retorno dos métodos de servidor. Isso é útil para cenários em que os fragmentos de dados serão apresentadas ao longo do tempo. Quando um valor de retorno é transmitido ao cliente, cada fragmento é enviado ao cliente assim que ele se torna disponível, em vez de aguardar que todos os dados fiquem disponíveis.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Configurar o hub

Um método de hub automaticamente se torna um método de hub streaming quando ele retorna um `ChannelReader<T>` ou um `Task<ChannelReader<T>>`. Abaixo está um exemplo que mostra os conceitos básicos do fluxo de dados para o cliente. Sempre que um objeto é gravado o `channelReader` esse objeto é enviado imediatamente para o cliente. No final, o `channelReader` estiver concluído para dizer ao cliente o fluxo está fechado.

### NOTE

- Gravar o `ChannelReader` em um thread em segundo plano e retorne o `ChannelReader` assim que possível. Outras chamadas de hub serão bloqueadas até que um `ChannelReader` é retornado.
- Encapsular sua lógica em uma `try ... catch` e conclua o `Channel` em `catch` e fora de `catch` para garantir que o hub de invocação de método é concluída corretamente.

```
public class StreamHub : Hub
{
    public ChannelReader<int> Counter(int count, int delay)
    {
        var channel = Channel.CreateUnbounded<int>();

        // We don't want to await WriteItemsAsync, otherwise we'd end up waiting
        // for all the items to be written before returning the channel back to
        // the client.
        _ = WriteItemsAsync(channel.Writer, count, delay);

        return channel.Reader;
    }

    private async Task WriteItemsAsync(
        ChannelWriter<int> writer,
        int count,
        int delay)
    {
        try
        {
            for (var i = 0; i < count; i++)
            {
                await writer.WriteAsync(i);
                await Task.Delay(delay);
            }
        }
        catch (Exception ex)
        {
            writer.TryComplete(ex);
        }

        writer.TryComplete();
    }
}
```

```

public class StreamHub : Hub
{
    public ChannelReader<int> Counter(
        int count,
        int delay,
        CancellationToken cancellationToken)
    {
        var channel = Channel.CreateUnbounded<int>();

        // We don't want to await WriteItemsAsync, otherwise we'd end up waiting
        // for all the items to be written before returning the channel back to
        // the client.
        _ = WriteItemsAsync(channel.Writer, count, delay, cancellationToken);

        return channel.Reader;
    }

    private async Task WriteItemsAsync(
        ChannelWriter<int> writer,
        int count,
        int delay,
        CancellationToken cancellationToken)
    {
        try
        {
            for (var i = 0; i < count; i++)
            {
                // Check the cancellation token regularly so that the server will stop
                // producing items if the client disconnects.
                cancellationToken.ThrowIfCancellationRequested();
                await writer.WriteAsync(i);

                // Use the cancellationToken in other APIs that accept cancellation
                // tokens so the cancellation can flow down to them.
                await Task.Delay(delay, cancellationToken);
            }
        }
        catch (Exception ex)
        {
            writer.TryComplete(ex);
        }

        writer.TryComplete();
    }
}

```

No ASP.NET Core 2.2 ou posterior, os métodos de Hub de streaming podem aceitar um `CancellationToken` parâmetro que será acionado quando o cliente cancela a assinatura do fluxo. Use esse token para interromper a operação do servidor e liberar quaisquer recursos se o cliente se desconecta antes do final do fluxo.

## Cliente .NET

O `StreamAsChannelAsync` método no `HubConnection` é usado para invocar um método de transmissão. Passe o nome do método de hub e argumentos definidos no método de hub para `StreamAsChannelAsync`. O parâmetro genérico em `StreamAsChannelAsync<T>` especifica o tipo de objetos retornados pelo método de transmissão. Um `ChannelReader<T>` é retornada da invocação de fluxo e representa o fluxo no cliente. Para ler dados, um padrão comum é executar um loop sobre `WaitToReadAsync` e chamar `TryRead` quando os dados estiverem disponíveis. O loop terminará quando o fluxo foi fechado pelo servidor ou o token de cancelamento passado para `StreamAsChannelAsync` é cancelada.

```

// Call "Cancel" on this CancellationTokenSource to send a cancellation message to
// the server, which will trigger the corresponding token in the Hub method.
var cancellationTokenSource = new CancellationTokenSource();
var channel = await hubConnection.StreamAsChannelAsync<int>(
    "Counter", 10, 500, cancellationTokenSource.Token);

// Wait asynchronously for data to become available
while (await channel.WaitToReadAsync())
{
    // Read all currently available data synchronously, before waiting for more data
    while (channel.TryRead(out var count))
    {
        Console.WriteLine($"{count}");
    }
}

Console.WriteLine("Streaming completed");

```

```

var channel = await hubConnection
    .StreamAsChannelAsync<int>("Counter", 10, 500, CancellationToken.None);

// Wait asynchronously for data to become available
while (await channel.WaitToReadAsync())
{
    // Read all currently available data synchronously, before waiting for more data
    while (channel.TryRead(out var count))
    {
        Console.WriteLine($"{count}");
    }
}

Console.WriteLine("Streaming completed");

```

## Cliente JavaScript

Os clientes JavaScript chamar métodos de streaming em hubs usando `connection.stream`. O `stream` método aceita dois argumentos:

- O nome do método de hub. No exemplo a seguir, o nome do método de hub é `Counter`.
- Argumentos definidos no método de hub. No exemplo a seguir, os argumentos são: uma contagem do número de itens de fluxo para receber e o atraso entre itens de fluxo.

`connection.stream` Retorna um `IStreamResult` que contém um `subscribe` método. Passar uma `IStreamSubscriber` para `subscribe` e defina as `next`, `error`, e `complete` retornos de chamada para receber as notificações a `stream` invocação.

```
connection.stream("Counter", 10, 500)
    .subscribe({
        next: (item) => {
            var li = document.createElement("li");
            li.textContent = item;
            document.getElementById("messagesList").appendChild(li);
        },
        complete: () => {
            var li = document.createElement("li");
            li.textContent = "Stream completed";
            document.getElementById("messagesList").appendChild(li);
        },
        error: (err) => {
            var li = document.createElement("li");
            li.textContent = err;
            document.getElementById("messagesList").appendChild(li);
        },
    });
});
```

Para terminar o fluxo do cliente, chame o `dispose` método em de `ISubscription` que é retornado do `subscribe` método.

Para terminar o fluxo do cliente, chame o `dispose` método em de `ISubscription` que é retornado do `subscribe` método. Chamar esse método fará com que o `CancellationToken` parâmetro do método de Hub (se você tiver fornecido um) a ser cancelada.

## Recursos relacionados

- [Hubs](#)
- [Cliente .NET](#)
- [Cliente JavaScript](#)
- [Publicar no Azure](#)

# Diferenças entre o SignalR do ASP.NET e o SignalR do ASP.NET Core

25/01/2019 • 8 minutes to read • [Edit Online](#)

SignalR do ASP.NET Core não é compatível com clientes ou servidores para ASP.NET SignalR. Este artigo fornece detalhes sobre os recursos que foram removidos ou alterados no SignalR do ASP.NET Core.

## Como identificar a versão do SignalR

	ASP.NET SIGNALR	SIGNALR DO ASP.NET CORE
Pacote do NuGet Server	<a href="#">Microsoft.AspNet.SignalR</a>	<a href="#">Microsoft.AspNetCore.App</a> (.NET Core) <a href="#">Microsoft</a> (.NET Framework)
Pacotes NuGet de cliente	<a href="#">Microsoft.AspNet.SignalR.Client</a> <a href="#">Microsoft.AspNet.SignalR.JS</a>	<a href="#">Microsoft.AspNetCore.SignalR.Client</a>
Pacote npm de cliente	signalr	@aspnet/signalr
Cliente de Java	<a href="#">Repositório GitHub</a> (preferido)	Pacote do Maven <a href="#">com.microsoft.signalr</a>
Tipo de aplicativo de servidor	ASP.NET (System. Web) ou a auto-hospedagem de OWIN	ASP.NET Core
Plataformas de servidor com suporte	.NET framework 4.5 ou posterior	.NET Framework 4.6.1 ou posterior .NET core 2.1 ou posterior

## Diferenças de recursos

### Reconexão automática

Reconexão automática não têm suporte no SignalR do ASP.NET Core. Se o cliente for desconectado, o usuário explicitamente deve iniciar uma nova conexão, se eles desejam se reconectar. No ASP.NET SignalR, SignalR tentará reconectar-se ao servidor se a conexão for interrompida.

### Suporte de protocolo

SignalR do ASP.NET Core dá suporte a JSON, bem como um novo protocolo binário, com base em [MessagePack](#). Além disso, os protocolos personalizados podem ser criados.

### Transportes

Não há suporte para o transporte de quadro para sempre SignalR do ASP.NET Core.

## Diferenças no servidor

As bibliotecas do lado do servidor SignalR do ASP.NET Core são incluídas na [metapacote do Microsoft](#) que faz parte do pacote a **aplicativo Web ASP.NET Core** modelo Razor e MVC projetos.

SignalR do ASP.NET Core é um middleware do ASP.NET Core, portanto, ele deve ser configurado por meio da chamada [AddSignalR](#) em `startup.ConfigureServices`.

```
services.AddSignalR()
```

Para configurar o roteamento, mapear as rotas para os hubs de dentro de [UseSignalR](#) chamada de método no `Startup.Configure` método.

```
app.UseSignalR(routes =>
{
    routes.MapHub<ChatHub>("/hub");
});
```

## Sessões temporárias

O modelo de expansão do SignalR do ASP.NET permite que os clientes para se reconectar e enviar mensagens para qualquer servidor no farm. No SignalR do ASP.NET Core, o cliente deve interagir com o mesmo servidor durante a conexão. Para escala horizontal usando Redis, isso significa que as sessões temporárias são necessárias. Para o uso de expansão [serviço do Azure SignalR](#), sessões temporárias não são necessárias porque o serviço lida com as conexões aos clientes.

## Hub único por conexão

O SignalR do ASP.NET Core, o modelo de conexão foi simplificado. As conexões são feitas diretamente a um único hub, em vez de uma única conexão está sendo usado para compartilhar o acesso a vários hubs.

## Streaming

ASP.NET SignalR Core agora dá suporte à [dados de streaming](#) do hub para o cliente.

## Estado

A capacidade de passar o estado arbitrário entre clientes e o hub (geralmente chamado de HubState) foi removida, bem como suporte para mensagens de progresso. Não há nenhum equivalente de proxies de hub no momento.

## Remoção de PersistentConnection

No SignalR do ASP.NET Core, o [PersistentConnection](#) classe foi removida.

## GlobalHost

O ASP.NET Core tem dentro da estrutura de injeção de dependência (DI). Serviços podem usar a DI para acessar o `HubContext`. O `GlobalHost` objeto que é usado no ASP.NET SignalR para obter um `HubContext` não existe no SignalR do ASP.NET Core.

## HubPipeline

SignalR do ASP.NET Core não tem suporte para `HubPipeline` módulos.

# Diferenças no cliente

## TypeScript

O cliente SignalR do ASP.NET Core é escrito em [TypeScript](#). Você pode escrever em JavaScript ou TypeScript ao usar o [cliente JavaScript](#).

## O cliente JavaScript é hospedado em npm

Nas versões anteriores, o cliente JavaScript foi obtido por meio de um pacote do NuGet no Visual Studio. Para as versões de núcleo, o `@aspnet/signalr` pacote npm contém as bibliotecas de JavaScript. Este pacote não está incluído na **aplicativo Web ASP.NET Core** modelo. Usar npm para obter e instalar o `@aspnet/signalr` pacote npm.

```
npm init -y
npm install @aspnet/signalr
```

## jQuery

A dependência no jQuery foi removida, no entanto, projetos ainda podem usar jQuery.

## Suporte do Internet Explorer

SignalR do ASP.NET Core requer o Microsoft Internet Explorer 11 ou posterior (o SignalR do ASP.NET com suporte Microsoft Internet Explorer 8 e posterior).

## Sintaxe de método de cliente JavaScript

A sintaxe de JavaScript foi alterado da versão anterior do SignalR. Em vez de usar o `$connection` de objeto, criar uma conexão usando o [HubConnectionBuilder API](#).

```
const connection = new signalR.HubConnectionBuilder()
    .withUrl("/hub")
    .build();
```

Use o [em](#) método para especificar que o hub pode chamar métodos do cliente.

```
connection.on("ReceiveMessage", (user, message) => {
    const msg = message.replace(/&/g, "&").replace(/</g, "<").replace(/>/g, ">");
    const encodedMsg = user + " says " + msg;
    log(encodedMsg);
});
```

Depois de criar o método do cliente, inicie a conexão de hub. Cadeia de um [catch](#) método para fazer logon ou lidar com erros.

```
connection.start().catch(err => console.error(err.toString()));
```

## Proxies de Hub

Os proxies de Hub não automaticamente são gerados. Em vez disso, o nome do método é passado para o [invocar](#) API como uma cadeia de caracteres.

## .NET e outros clientes

O `Microsoft.AspNetCore.SignalR.Client` pacote NuGet contém as bibliotecas de cliente .NET para o SignalR do ASP.NET Core.

Use o [HubConnectionBuilder](#) para criar e compilar uma instância de uma conexão a um hub.

```
connection = new HubConnectionBuilder()
    .WithUrl("url")
    .Build();
```

## Diferenças de expansão

SignalR do ASP.NET oferece suporte a SQL Server e o Redis. SignalR do ASP.NET Core dá suporte ao serviço do Azure SignalR e Redis.

## ASP.NET

- [Expansão do SignalR com o barramento de serviço do Azure](#)

- [Expansão do SignalR com Redis](#)
- [Expansão do SignalR com o SQL Server](#)

#### **ASP.NET Core**

- [Serviço Azure SignalR](#)

## Recursos adicionais

- [Hubs](#)
- [Cliente JavaScript](#)
- [Cliente .NET](#)
- [Plataformas compatíveis](#)

# Suporte ao WebSockets no ASP.NET Core

30/01/2019 • 13 minutes to read • [Edit Online](#)

Por [Tom Dykstra](#) e [Andrew Stanton-Nurse](#)

Este artigo explica como começar a usar o WebSockets no ASP.NET Core. [WebSocket \(RFC 6455\)](#) é um protocolo que permite canais de comunicação persistentes bidirecionais em conexões TCP. Ele é usado em aplicativos que se beneficiam de comunicação rápida e em tempo real, como chat, painel e aplicativos de jogos.

[Exibir ou baixar um código de exemplo \(como baixar\)](#). Confira a seção [Próximas etapas](#) para obter mais informações.

## Pré-requisitos

- ASP.NET Core 1.1 ou posterior
- Qualquer sistema operacional compatível com o ASP.NET Core:
  - Windows 7/Windows Server 2008 ou posterior
  - Linux
  - macOS
- Se o aplicativo é executado no Windows com o IIS:
  - Windows 8/Windows Server 2012 ou posterior
  - IIS 8/IIS 8 Express
  - WebSockets precisam ser habilitados (confira a seção [Suporte para IIS/IIS Express](#)).
- Se o aplicativo é executado no [HTTP.sys](#):
  - Windows 8/Windows Server 2012 ou posterior
- Para saber quais são os navegadores compatíveis, confira <https://caniuse.com/#feat=websockets>.

## Quando usar WebSockets

Use WebSockets para trabalhar diretamente com uma conexão de soquete. Por exemplo, use WebSockets para obter o melhor desempenho possível em um jogo em tempo real.

[SignalR do ASP.NET Core](#) é uma biblioteca que simplifica a adição da funcionalidade da Web em tempo real aos aplicativos. Ele usa WebSockets sempre que possível.

## Como usar WebSockets

- Instale o pacote [Microsoft.AspNetCore.WebSockets](#).
- Configure o middleware.
- Aceite solicitações do WebSocket.
- Envie e receba mensagens.

### Configurar o middleware

Adicione o middleware do WebSockets no método `Configure` da classe `Startup`:

```
app.UseWebSockets();
```

```
app.UseWebSockets();
```

As seguintes configurações podem ser definidas:

- `KeepAliveInterval` – a frequência para enviar quadros "ping" ao cliente para garantir que os proxies mantenham a conexão aberta. O padrão é dois minutos.
- `ReceiveBufferSize` – o tamanho do buffer usado para receber dados. Os usuários avançados podem precisar alterar isso para ajuste de desempenho com base no tamanho dos dados. O padrão é 4 KB.

As seguintes configurações podem ser definidas:

- `KeepAliveInterval` – a frequência para enviar quadros "ping" ao cliente para garantir que os proxies mantenham a conexão aberta. O padrão é dois minutos.
- `ReceiveBufferSize` – o tamanho do buffer usado para receber dados. Os usuários avançados podem precisar alterar isso para ajuste de desempenho com base no tamanho dos dados. O padrão é 4 KB.
- `AllowedOrigins` – Uma lista de valores de cabeçalho de origem permitidos para solicitações do WebSocket. Por padrão, todas as origens são permitidas. Consulte "Restrição de origem do WebSocket" abaixo para obter detalhes.

```
var webSocketOptions = new WebSocketOptions()
{
    KeepAliveInterval = TimeSpan.FromSeconds(120),
    ReceiveBufferSize = 4 * 1024
};

app.UseWebSockets(webSocketOptions);
```

```
var webSocketOptions = new WebSocketOptions()
{
    KeepAliveInterval = TimeSpan.FromSeconds(120),
    ReceiveBufferSize = 4 * 1024
};
app.UseWebSockets(webSocketOptions);
```

## Aceitar solicitações do WebSocket

Em algum lugar e em um momento futuro no ciclo de vida da solicitação (posteriormente no método `Configure` ou em uma ação do MVC, por exemplo), verifique se é uma solicitação do WebSocket e aceite-a.

O exemplo a seguir é de uma fase posterior do método `Configure`:

```
app.Use(async (context, next) =>
{
    if (context.Request.Path == "/ws")
    {
        if (context.WebSockets.IsWebSocketRequest)
        {
            WebSocket webSocket = await context.WebSockets.AcceptWebSocketAsync();
            await Echo(context, webSocket);
        }
        else
        {
            context.Response.StatusCode = 400;
        }
    }
    else
    {
        await next();
    }
});
```

```
app.Use(async (context, next) =>
{
    if (context.Request.Path == "/ws")
    {
        if (context.WebSockets.IsWebSocketRequest)
        {
            WebSocket webSocket = await context.WebSockets.AcceptWebSocketAsync();
            await Echo(context, webSocket);
        }
        else
        {
            context.Response.StatusCode = 400;
        }
    }
    else
    {
        await next();
    }
});
```

Uma solicitação do WebSocket pode entrar em qualquer URL, mas esse código de exemplo aceita apenas solicitações de `/ws`.

### Enviar e receber mensagens

O método `AcceptWebSocketAsync` atualiza a conexão TCP para uma conexão WebSocket e fornece um objeto `WebSocket`. Use o objeto `WebSocket` para enviar e receber mensagens.

O código mostrado anteriormente que aceita a solicitação do WebSocket passa o objeto `WebSocket` para um método `Echo`. O código recebe uma mensagem e envia de volta imediatamente a mesma mensagem. As mensagens são enviadas e recebidas em um loop até que o cliente feche a conexão:

```

private async Task Echo(HttpContext context, WebSocket webSocket)
{
    var buffer = new byte[1024 * 4];
    WebSocketReceiveResult result = await webSocket.ReceiveAsync(new ArraySegment<byte>(buffer),
CancellationToken.None);
    while (!result.CloseStatus.HasValue)
    {
        await webSocket.SendAsync(new ArraySegment<byte>(buffer, 0, result.Count), result.MessageType,
result.EndOfMessage, CancellationToken.None);

        result = await webSocket.ReceiveAsync(new ArraySegment<byte>(buffer), CancellationToken.None);
    }
    await webSocket.CloseAsync(result.CloseStatus.Value, result.CloseStatusDescription,
CancellationToken.None);
}

```

```

private async Task Echo(HttpContext context, WebSocket webSocket)
{
    var buffer = new byte[1024 * 4];
    WebSocketReceiveResult result = await webSocket.ReceiveAsync(new ArraySegment<byte>(buffer),
CancellationToken.None);
    while (!result.CloseStatus.HasValue)
    {
        await webSocket.SendAsync(new ArraySegment<byte>(buffer, 0, result.Count), result.MessageType,
result.EndOfMessage, CancellationToken.None);

        result = await webSocket.ReceiveAsync(new ArraySegment<byte>(buffer), CancellationToken.None);
    }
    await webSocket.CloseAsync(result.CloseStatus.Value, result.CloseStatusDescription,
CancellationToken.None);
}

```

Ao aceitar a conexão WebSocket antes de iniciar o loop, o pipeline de middleware é encerrado. Ao fechar o soquete, o pipeline é desenrolado. Ou seja, a solicitação deixa de avançar no pipeline quando o WebSocket é aceito. Quando o loop é concluído e o soquete é fechado, a solicitação continua a avançar no pipeline.

### Tratar desconexões de cliente

O servidor não é informado automaticamente quando o cliente se desconecta devido à perda de conectividade. O servidor recebe uma mensagem de desconexão apenas quando o cliente a envia, e isso é possível somente quando há conexão com a Internet. Se quiser tomar alguma medida quando isso ocorrer, defina um tempo limite, caso não receba nada do cliente após uma determinada janela de tempo.

Se o cliente não envia mensagens com frequência, e você prefere não definir um tempo limite apenas porque a conexão fica ociosa, peça ao cliente para usar um temporizador a fim de enviar uma mensagem de ping a cada X segundos. No servidor, se uma mensagem não chegar dentro de 2\*X segundos após a mensagem anterior, encerre a conexão e informe que o cliente se desconectou. Aguarde o dobro do intervalo de tempo esperado a fim de deixar um tempo extra para atrasos na rede, que possam atrasar a mensagem de ping.

### Restrição de origem do WebSocket

As proteções fornecidas pelo CORS não se aplicam ao WebSockets. Navegadores **não**:

- Executam solicitações de simulação de CORS.
- Respeitam as restrições especificadas em cabeçalhos `Access-Control` ao fazer solicitações de WebSocket.

No entanto, os navegadores enviam o cabeçalho `Origin` ao emitir solicitações de WebSocket. Os aplicativos devem ser configurados para validar esses cabeçalhos e garantir que apenas WebSockets provenientes de origens esperadas sejam permitidos.

Se você estiver hospedando o servidor em "https://server.com" e hospedando seu cliente em "https://client.com",

adicone "https://client.com" à lista `AllowedOrigins` para o WebSockets verificar.

```
var webSocketOptions = new WebSocketOptions()
{
    KeepAliveInterval = TimeSpan.FromSeconds(120),
    ReceiveBufferSize = 4 * 1024
};
webSocketOptions.AllowedOrigins.Add("https://client.com");
webSocketOptions.AllowedOrigins.Add("https://www.client.com");

app.UseWebSockets(webSocketOptions);
```

#### NOTE

O cabeçalho `Origin` é controlado pelo cliente e, como o cabeçalho `Referer`, pode ser falsificado. **Não** use esses cabeçalhos como um mecanismo de autenticação.

## Suporte ao IIS/IIS Express

O Windows Server 2012 ou posterior e o Windows 8 ou posterior com o IIS/IIS Express 8 ou posterior são compatíveis com o protocolo WebSocket.

#### NOTE

WebSockets estão sempre habilitados ao usar o IIS Express.

### Habilitar WebSockets no IIS

Para habilitar o suporte para o protocolo WebSocket no Windows Server 2012 ou posterior:

#### NOTE

Estas etapas não são necessárias ao usar o IIS Express

1. Use o assistente **Adicionar Funções e Recursos** por meio do menu **Gerenciar** ou do link no **Gerenciador do Servidor**.
2. Selecione **Instalação baseada em função ou em recurso**. Selecione **Avançar**.
3. Selecione o servidor apropriado (o servidor local é selecionado por padrão). Selecione **Avançar**.
4. Expanda **Servidor Web (IIS)** na árvore **Funções**, expanda **Servidor Web** e, em seguida, expanda **Desenvolvimento de Aplicativos**.
5. Selecione o **Protocolo WebSocket**. Selecione **Avançar**.
6. Se não forem necessários recursos adicionais, selecione **Avançar**.
7. Clique em **Instalar**.
8. Quando a instalação for concluída, selecione **Fstrar** para sair do assistente.

Para habilitar o suporte para o protocolo WebSocket no Windows 8 ou posterior:

#### NOTE

Estas etapas não são necessárias ao usar o IIS Express

1. Navegue para **Painel de Controle > Programas > Programas e Recursos > Ativar ou desativar recursos do Windows** (lado esquerdo da tela).

2. Abra os seguintes nós: **Serviços de Informações da Internet** > **Serviços da World Wide Web** >

### Recursos de Desenvolvimento de Aplicativos.

3. Selecione o recurso **Protocolo WebSocket**. Selecione **OK**.

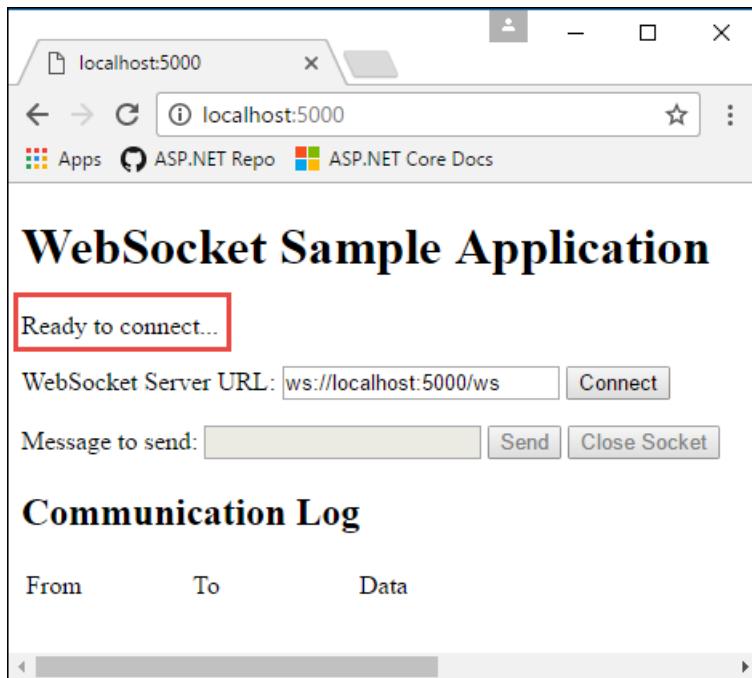
#### Desabilite o WebSocket ao usar o socket.io no Node.js

Se você estiver usando o suporte do WebSocket no [socket.io](#) no [Node.js](#), desabilite o módulo do WebSocket do IIS padrão usando o elemento `WebSocket` em `web.config` ou em `applicationHost.config`. Se essa etapa não for executada, o módulo do WebSocket do IIS tentará manipular a comunicação do WebSocket em vez do Node.js e o aplicativo.

```
<system.webServer>
  <WebSocket enabled="false" />
</system.webServer>
```

## Próximas etapas

O [aplicativo de exemplo](#) que acompanha este artigo é um aplicativo de eco. Ele tem uma página da Web que faz conexões WebSocket e o servidor reenvia para o cliente todas as mensagens recebidas. Execute o aplicativo em um prompt de comando (ele não está configurado para execução no Visual Studio com o IIS Express) e navegue para `http://localhost:5000`. A página da Web exibe o status de conexão no canto superior esquerdo:



Selecione **Conectar** para enviar uma solicitação WebSocket para a URL exibida. Insira uma mensagem de teste e selecione **Enviar**. Quando terminar, selecione **Figar Soquete**. A seção **Log de Comunicação** relata cada ação de abertura, envio e fechamento conforme ela ocorre.

A screenshot of a Microsoft Edge browser window. The address bar shows 'localhost:5000'. Below the address bar, there are links for 'Apps', 'ASP.NET Repo', and 'ASP.NET Core Docs'. The main content area has a title 'WebSocket Sample Application' and a status message 'Closed'. It includes a text input field with 'ws://localhost:5000/ws' and a 'Connect' button. Below that is a text input field with 'second test message' and buttons for 'Send' and 'Close Socket'. A section titled 'Communication Log' displays a table of messages:

From	To	Data
Connection opened		
Client	Server	first test message
Server	Client	first test message
Client	Server	second test message
Server	Client	second test message

Below the table, it says 'Connection closed. Code: 1000. Reason: Closing from client'.

# Testes de unidade de páginas do Razor no ASP.NET Core

30/10/2018 • 17 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

ASP.NET Core dá suporte a testes de unidade de aplicativos de páginas do Razor. Testes dos dados (DAL) da camada de acesso e modelos de página ajudam a garantir:

- Partes de um aplicativo páginas Razor trabalhem independentemente ou em conjunto como uma unidade durante a construção do aplicativo.
- Classes e métodos limitaram escopos de responsabilidade.
- Existe documentação adicional sobre como o aplicativo deve se comportar.
- As regressões, que são trazidas por atualizações para o código de erros, são encontradas durante a implantação e compilação automatizada.

Este tópico pressupõe que você tenha uma compreensão básica dos aplicativos das páginas do Razor e testes de unidade. Se você estiver familiarizado com conceitos de teste ou de aplicativos de páginas do Razor, consulte os tópicos a seguir:

- [Introdução a Páginas do Razor](#)
- [Introdução a Páginas do Razor](#)
- [Teste de unidade em C# no .NET Core usando dotnet test e xUnit](#)

[Exibir ou baixar código de exemplo \(como baixar\)](#)

O projeto de exemplo é composto de dois aplicativos:

APLICATIVO	PASTA DO PROJETO	DESCRIÇÃO
Aplicativo de mensagens	<i>src/RazorPagesTestSample</i>	Permite que um usuário adicione, exclua uma, excluir todas as e analisar as mensagens.
Aplicativo de teste	<i>tests/RazorPagesTestSample.Tests</i>	Usado para o aplicativo de mensagem de teste de unidade: acesso a dados (DAL) de camada e o modelo de página de índice.

Os testes podem ser executados usando os recursos de teste interno de um IDE, como [Visual Studio](#). Se usando [Visual Studio Code](#) ou a linha de comando, execute o seguinte comando em um prompt de comando na *tests/RazorPagesTestSample.Tests* pasta:

```
dotnet test
```

## Organização de aplicativo de mensagem

O aplicativo de mensagem é um sistema de mensagem de páginas do Razor simples com as seguintes características:

- A página de índice do aplicativo (*Pages* e *Pages/Index.cshtml.cs*) fornece uma interface do usuário e a página de métodos de modelo para controlar a adição, exclusão e análise de mensagens (palavras médias por mensagem).
- Uma mensagem é descrita pela `Message` classe (*Data/Message.cs*) com duas propriedades: `Id` (chave) e `Text` (mensagem). O `Text` propriedade é necessária e é limitada a 200 caracteres.
- As mensagens são armazenadas usando [banco de dados do Entity Framework na memória](#).
- O aplicativo contém uma camada de acesso de dados (DAL) na sua classe de contexto do banco de dados, `AppDbContext` (*Data/AppDbContext.cs*). Os métodos DAL são marcados como `virtual`, que permite que os métodos para uso em testes de simulação.
- Se o banco de dados está vazio na inicialização do aplicativo, o repositório de mensagens é inicializado com três mensagens. Eles *propagado mensagens* também são usados em testes.

<sup>†</sup>O tópico EF [testar com InMemory](#), explica como usar um banco de dados na memória para testes com MSTest. Este tópico usa o [xUnit](#) estrutura de teste. Conceitos de teste e teste implementações em estruturas de teste diferentes são semelhantes, mas não idênticos.

Embora o aplicativo não usa o padrão de repositório e não é um exemplo efetivação do [padrão de unidade de trabalho \(UoW\)](#), páginas do Razor dá suporte a esses padrões de desenvolvimento. Para obter mais informações, consulte [Projetando a camada de persistência de infraestrutura](#) e [lógica do controlador de teste](#) (o exemplo implementa o padrão de repositório).

## Organização de aplicativo de teste

O aplicativo de teste é um aplicativo de console dentro de *tests/RazorPagesTestSample.Tests* pasta.

PASTA DO APlicATIVO DE TESTE	DESCRÍÇÃO
<i>UnitTests</i>	<ul style="list-style-type: none"> <li><i>DataAccessLayerTest.cs</i> contém os testes de unidade para o DAL.</li> <li><i>IndexPageTests.cs</i> contém os testes de unidade para o modelo de página de índice.</li> </ul>
<i>Utilitários</i>	Contém o <code>TestingDbContextOptions</code> método usado para criar o novo banco de dados opções de contexto para cada teste de unidade da DAL, de modo que o banco de dados é redefinido como sua condição de linha de base para cada teste.

É a estrutura de teste [xUnit](#). É o objeto de objetos fictícios [Moq](#).

## Testes de unidade de dados (DAL) da camada de acesso

O aplicativo de mensagem tem uma DAL com quatro métodos contidos na `AppDbContext` classe (*src/RazorPagesTestSample/Data/AppDbContext.cs*). Cada método tem um ou dois testes de unidade no aplicativo de teste.

MÉTODO DAL	FUNÇÃO
<code>GetMessagesAsync</code>	Obtém uma <code>List&lt;Message&gt;</code> do banco de dados classificado pelo <code>Text</code> propriedade.
<code>AddMessageAsync</code>	Adiciona um <code>Message</code> no banco de dados.

MÉTODO DAL	FUNÇÃO
<code>DeleteAllMessagesAsync</code>	Exclui todos os <code>Message</code> entradas do banco de dados.
<code>DeleteMessageAsync</code>	Exclui um único <code>Message</code> do banco de dados por <code>Id</code> .

Testes de unidade da DAL requerem `DbContextOptions` ao criar um novo `AppDbContext` para cada teste. Uma abordagem para criar o `DbContextOptions` para cada teste é usar um `DbContextOptionsBuilder`:

```
var optionsBuilder = new DbContextOptionsBuilder<AppDbContext>()
    .UseInMemoryDatabase("InMemoryDb");

using (var db = new AppDbContext(optionsBuilder.Options))
{
    // Use the db here in the unit test.
}
```

O problema com essa abordagem é que cada teste recebe o banco de dados no estado em que o teste anterior a deixou. Isso pode ser um problema ao tentar escrever testes de unidade atômica que não interfiram uns aos outros. Para forçar o `AppDbContext` para usar um novo contexto de banco de dados para cada teste, forneça um `DbContextOptions` instância com base em um novo provedor de serviço. O aplicativo de teste mostra como fazer isso usando seus `Utilities` método de classe `TestingDbContextOptions` (`tests/RazorPagesTestSample.Tests/Utilities/Utilities.cs`):

```
public static DbContextOptions<AppDbContext> TestDbContextOptions()
{
    // Create a new service provider to create a new in-memory database.
    var serviceProvider = new ServiceCollection()
        .AddEntityFrameworkInMemoryDatabase()
        .BuildServiceProvider();

    // Create a new options instance using an in-memory database and
    // IServiceProvider that the context should resolve all of its
    // services from.
    var builder = new DbContextOptionsBuilder<AppDbContext>()
        .UseInMemoryDatabase("InMemoryDb")
        .UseInternalServiceProvider(serviceProvider);

    return builder.Options;
}
```

Usando o `DbContextOptions` na unidade de DAL testes permite que cada teste executado atomicamente com uma instância de banco de dados atualizado:

```
using (var db = new AppDbContext(Utilities.TestingDbContextOptions()))
{
    // Use the db here in the unit test.
}
```

Cada método de teste na `DataAccessLayerTest` classe (`UnitTests/DataAccessLayerTest.cs`) segue um padrão semelhante Arrange, Act-Assert:

1. Organizar: O banco de dados está configurado para o teste de e/ou o resultado esperado é definido.
2. O ACT: O teste é executado.
3. Assert: Asserções são feitas para determinar se o resultado do teste é um sucesso.

Por exemplo, o `DeleteMessageAsync` método é responsável por remover uma única mensagem identificada pelo seu `Id` (`src/RazorPagesTestSample/Data/AppDbContext.cs`):

```
public async virtual Task DeleteMessageAsync(int id)
{
    var message = await Messages.FindAsync(id);

    if (message != null)
    {
        Messages.Remove(message);
        await SaveChangesAsync();
    }
}
```

Há dois testes para esse método. Um teste verifica que o método exclui uma mensagem quando a mensagem estiver presente no banco de dados. Os outros testes de método que o banco de dados não serão alteradas se a mensagem `Id` para exclusão não existe. O `DeleteMessageAsync_MessageIsDeleted_WhenMessageIsFound` método é mostrado abaixo:

```
[Fact]
public async Task DeleteMessageAsync_MessageIsDeleted_WhenMessageIsFound()
{
    using (var db = new AppDbContext(Utilities.TestDbContextOptions()))
    {
        // Arrange
        var seedMessages = AppDbContext.GetSeedingMessages();
        await db.AddRangeAsync(seedMessages);
        await db.SaveChangesAsync();
        var recId = 1;
        var expectedMessages =
            seedMessages.Where(message => message.Id != recId).ToList();

        // Act
        await db.DeleteMessageAsync(recId);

        // Assert
        var actualMessages = await db.Messages.AsNoTracking().ToListAsync();
        Assert.Equal(
            expectedMessages.OrderBy(x => x.Id),
            actualMessages.OrderBy(x => x.Id),
            new Utilities.MessageComparer());
    }
}
```

Primeiro, o método executa a etapa Arranjar, onde ocorre a preparação para a etapa atuar. As mensagens de propagação são obtidas e mantidas em `seedMessages`. As mensagens de propagação são salvos no banco de dados. A mensagem com um `Id` de `1` é definido para exclusão. Quando o `DeleteMessageAsync` método é executado, as mensagens esperadas devem ter todas as mensagens, exceto aquele com um `Id` de `1`. O `expectedMessages` variável representa esse resultado esperado.

```
// Arrange
var seedMessages = AppDbContext.GetSeedingMessages();
await db.AddRangeAsync(seedMessages);
await db.SaveChangesAsync();
var recId = 1;
var expectedMessages =
    seedMessages.Where(message => message.Id != recId).ToList();
```

O método funciona: O `DeleteMessageAsync` método é executado passando a `recId` de `1`:

```
// Act
await db.DeleteMessageAsync(recId);
```

Por fim, o método obtém o `Messages` do contexto e o compara a `expectedMessages` declarando que os dois são iguais:

```
// Assert
var actualMessages = await db.Messages.AsNoTracking().ToListAsync();
Assert.Equal(
    expectedMessages.OrderBy(m => m.Id).Select(m => m.Text),
    actualMessages.OrderBy(m => m.Id).Select(m => m.Text));
```

Para comparar os dois `List<Message>` são os mesmos:

- As mensagens são ordenadas por `Id`.
- Pares de mensagens são comparados no `Text` propriedade.

Um método de teste semelhante, `DeleteMessageAsync_NoMessageIsDeleted_WhenMessageNotFound` verifica o resultado da tentativa de excluir uma mensagem que não existe. Nesse caso, as mensagens esperadas no banco de dados devem ser iguais para as mensagens reais após o `DeleteMessageAsync` método é executado. Não deve haver nenhuma alteração para o conteúdo do banco de dados:

```
[Fact]
public async Task DeleteMessageAsync_NoMessageIsDeleted_WhenMessageNotFound()
{
    using (var db = new AppDbContext(Utilities.TestDbContextOptions()))
    {
        // Arrange
        var expectedMessages = AppDbContext.GetSeedingMessages();
        await db.AddRangeAsync(expectedMessages);
        await db.SaveChangesAsync();
        var recId = 4;

        // Act
        await db.DeleteMessageAsync(recId);

        // Assert
        var actualMessages = await db.Messages.AsNoTracking().ToListAsync();
        Assert.Equal(
            expectedMessages.OrderBy(m => m.Id).Select(m => m.Text),
            actualMessages.OrderBy(m => m.Id).Select(m => m.Text));
    }
}
```

## Testes de unidade dos métodos do modelo de página

Outro conjunto de testes de unidade é responsável por testes de métodos do modelo de página. No aplicativo de mensagem, os modelos de página de índice são encontrados na `IndexModel` classe `src/RazorPagesTestSample/Pages/Index.cshtml.cs`.

MÉTODO DE MODELO DE PÁGINA	FUNÇÃO
<code>OnGetAsync</code>	Obtém as mensagens de DAL para a interface do usuário usando o <code>GetMessagesAsync</code> método.

MÉTODO DE MODELO DE PÁGINA	FUNÇÃO
OnPostAddMessageAsync	Se o <code>ModelState</code> é válido, as chamadas <code>AddMessageAsync</code> para adicionar uma mensagem para o banco de dados.
OnPostDeleteAllMessagesAsync	Chamadas <code>DeleteAllMessagesAsync</code> para excluir todas as mensagens no banco de dados.
OnPostDeleteMessageAsync	Executa <code>DeleteMessageAsync</code> para excluir uma mensagem com o <code>Id</code> especificado.
OnPostAnalyzeMessagesAsync	Se uma ou mais mensagens estão no banco de dados, calcula o número médio de palavras por mensagem.

Os métodos de modelo de página são testados usando testes de sete na `IndexPageTests` classe (`tests/RazorPagesTestSample.Tests/UnitTests/IndexPageTests.cs`). Os testes de usam o padrão de Act Assert familiar. Esses testes se concentram em:

- Determinando se os métodos seguem o comportamento correto quando o `ModelState` é inválido.
- Confirmar os métodos produzem correto `IActionResult`.
- Verificando-se de que as atribuições de valor de propriedade são feitas corretamente.

Geralmente, esse grupo de testes de simular os métodos da DAL para produzir os dados esperados para a etapa atuar em que um método de modelo de página é executado. Por exemplo, o `GetMessagesAsync` método da `AppDbContext` é simulado para produzir uma saída. Quando este método é executado um método de modelo de página, a simulação retorna o resultado. Os dados não vem do banco de dados. Isso cria condições de teste previsível e confiável para uso a DAL nos testes de modelo de página.

O `OnGetAsync_PopulatesThePageModel_WithAListOfMessages` teste mostra como o `GetMessagesAsync` método é simulado para o modelo de página:

```
var mockAppDbContext = new Mock<AppDbContext>(optionsBuilder.Options);
var expectedMessages = AppDbContext.GetSeedingMessages();
mockAppDbContext.Setup(
    db => db.GetMessagesAsync().Returns(Task.FromResult(expectedMessages)));
var pageModel = new IndexModel(mockAppDbContext.Object);
```

Quando o `OnGetAsync` método é executado na etapa Act, ela chama o modelo de página `GetMessagesAsync` método.

Etapa atuar de teste de unidade (`tests/RazorPagesTestSample.Tests/UnitTests/IndexPageTests.cs`):

```
// Act
await pageModel.OnGetAsync();
```

`IndexPage` modelo de página `OnGetAsync` método (`src/RazorPagesTestSample/Pages/Index.cshtml.cs`):

```
public async Task OnGetAsync()
{
    Messages = await _db.GetMessagesAsync();
}
```

O `GetMessagesAsync` método no DAL não retornar o resultado para esta chamada de método. A versão fictícia do método retorna o resultado.

No `Assert` etapa, as mensagens reais (`actualMessages`) são atribuídos a partir de `Messages` propriedade do modelo de página. Uma verificação de tipo também é executada quando as mensagens são atribuídas. As mensagens esperadas e reais são comparadas por seus `Text` propriedades. O teste declara que os dois `List<Message>` instâncias contêm as mesmas mensagens.

```
// Assert
var actualMessages = Assert.IsAssignableFrom<List<Message>>(pageModel.Messages);
Assert.Equal(
    expectedMessages.OrderBy(m => m.Id).Select(m => m.Text),
    actualMessages.OrderBy(m => m.Id).Select(m => m.Text));
```

Outros testes neste grupo Criar página de objetos de modelo que incluem o `DefaultHttpContext`, o `ModelStateDictionary`, um `ActionContext` para estabelecer a `PageContext`, um  `ViewDataDictionary` e um `PageContext`. Elas são úteis na realização de testes. Por exemplo, o aplicativo de mensagens estabelece uma `ModelState` erro com `AddModelError` para verificar se válido `PageResult` é retornado quando `OnPostAddMessageAsync` é executado:

```
[Fact]
public async Task OnPostAddMessageAsync_ReturnsAPageResult_WhenModelStateIsInvalid()
{
    // Arrange
    var optionsBuilder = new DbContextOptionsBuilder<AppDbContext>()
        .UseInMemoryDatabase("InMemoryDb");
    var mockAppDbContext = new Mock<AppDbContext>(optionsBuilder.Options);
    var expectedMessages = AppDbContext.GetSeedingMessages();
    mockAppDbContext.Setup(db => db.GetMessagesAsync()).Returns(Task.FromResult(expectedMessages));
    var httpContext = new DefaultHttpContext();
    var ModelState = new ModelStateDictionary();
    var actionContext = new ActionContext(httpContext, new RouteData(), new PageActionDescriptor(),
    ModelState);
    var modelMetadataProvider = new EmptyModelMetadataProvider();
    var ViewData = new ViewDataDictionary(modelMetadataProvider, ModelState);
    var tempData = new TempDataDictionary(httpContext, Mock.Of<ITempDataProvider>());
    var pageContext = new PageContext(actionContext)
    {
        ViewData = ViewData
    };
    var pageModel = new IndexModel(mockAppDbContext.Object)
    {
        PageContext = pageContext,
        TempData = tempData,
        Url = new UrlHelper(actionContext)
    };
    pageModel.ModelState.AddModelError("Message.Text", "The Text field is required.");

    // Act
    var result = await pageModel.OnPostAddMessageAsync();

    // Assert
    Assert.IsType<PageResult>(result);
}
```

## Recursos adicionais

- [Teste de unidade em C# no .NET Core usando dotnet test e xUnit](#)
- [Controladores de teste](#)
- [Seu código de teste de unidade \(Visual Studio\)](#)
- [Testes de integração](#)
- [xUnit.net](#)

- [Guia de Introdução xUnit.net \(.NET Core/ASP.NET Core\)](#)
- [Moq](#)
- [Guia de início rápido Moq](#)

# Lógica do controlador de teste no ASP.NET Core

10/11/2018 • 19 minutes to read • [Edit Online](#)

Por [Steve Smith](#)

Controladores desempenham um papel central em qualquer aplicativo ASP.NET Core MVC. Assim, você precisa estar confiante de que os controladores se comportarão conforme o esperado. Testes automatizados podem detectar erros antes do aplicativo ser implantado em um ambiente de produção.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Testes de unidade da lógica do controlador

Os [testes de unidade](#) envolvem o teste de uma parte de um aplicativo isoladamente em relação à sua infraestrutura e às suas dependências. Quando a unidade está testando a lógica do controlador, somente o conteúdo de uma única ação é testada, não o comportamento de suas dependências ou da estrutura em si.

Configure testes de unidade de ações do controlador para se concentrarem no comportamento do controlador. Um teste de unidade do controlador evita cenários como [filtros](#), [roteamento](#) ou [model binding](#). Os testes que abrangem as interações entre os componentes que respondem coletivamente a uma solicitação são manipulados pelos *testes de integração*. Para obter mais informações sobre os testes de integração, consulte [Testes de integração no ASP.NET Core](#).

Se você estiver escrevendo filtros e rotas personalizados, realize testes de unidade neles de forma isolada, não como parte dos testes em uma ação do controlador específica.

Para demonstrar testes de unidade do controlador, examine o controlador a seguir no aplicativo de exemplo. O controlador Home exibe uma lista de sessões de debate e permite que novas sessões sejam criadas com uma solicitação POST:

```

public class HomeController : Controller
{
    private readonly IBrainstormSessionRepository _sessionRepository;

    public HomeController(IBrainstormSessionRepository sessionRepository)
    {
        _sessionRepository = sessionRepository;
    }

    public async Task<IActionResult> Index()
    {
        var sessionList = await _sessionRepository.ListAsync();

        var model = sessionList.Select(session => new StormSessionViewModel()
        {
            Id = session.Id,
            DateCreated = session.DateCreated,
            Name = session.Name,
            IdeaCount = session.Ideas.Count
        });

        return View(model);
    }

    public class NewSessionModel
    {
        [Required]
        public string SessionName { get; set; }
    }

    [HttpPost]
    public async Task<IActionResult> Index(NewSessionModel model)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }
        else
        {
            await _sessionRepository.AddAsync(new BrainstormSession()
            {
                DateCreated = DateTimeOffset.Now,
                Name = model.SessionName
            });
        }

        return RedirectToAction(actionName: nameof(Index));
    }
}

```

O controlador anterior:

- Segue o [Princípio de Dependências Explícitas](#).
- Espera [DI \(injeção de dependência\)](#) para fornecer uma instância de `IBrainstormSessionRepository`.
- Pode ser testado com um serviço `IBrainstormSessionRepository` fictício usando uma estrutura de objeto fictício, como [Moq](#). Um *objeto fictício* é um objeto fabricado com um conjunto predeterminado de comportamentos de propriedade e de método usado para teste. Para saber mais, consulte [Introdução aos testes de integração](#).

O método `HTTP GET Index` não tem nenhum loop ou branch e chama apenas um método. O teste de unidade para esta ação:

- Imita o serviço `IBrainstormSessionRepository` usando o método `GetTestSessions`. `GetTestSessions` cria duas

sessões de debate fictícias com datas e nomes de sessão.

- Executa o método `Index`.
- Faz declarações sobre o resultado retornado pelo método:
  - Um `ViewResult` é retornado.
  - O `ViewDataDictionary.Model` é um `StormSessionViewModel`.
  - Há duas sessões de debate armazenadas no `ViewDataDictionary.Model`.

```
[Fact]
public async Task Index_ReturnsAViewResult_WithAListOfBrainstormSessions()
{
    // Arrange
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.ListAsync())
        .ReturnsAsync(GetTestSessions());
    var controller = new HomeController(mockRepo.Object);

    // Act
    var result = await controller.Index();

    // Assert
    var viewResult = Assert.IsType<ViewResult>(result);
    var model = Assert.IsAssignableFrom<IEnumerable<StormSessionViewModel>>(
        viewResult.ViewData.Model);
    Assert.Equal(2, model.Count());
}
```

```
private List<BrainstormSession> GetTestSessions()
{
    var sessions = new List<BrainstormSession>();
    sessions.Add(new BrainstormSession()
    {
        DateCreated = new DateTime(2016, 7, 2),
        Id = 1,
        Name = "Test One"
    });
    sessions.Add(new BrainstormSession()
    {
        DateCreated = new DateTime(2016, 7, 1),
        Id = 2,
        Name = "Test Two"
    });
    return sessions;
}
```

Os testes método `HTTP POST Index` do controlador Home verificam se:

- Quando `ModelState.IsValid` é `false`, o método de ação retorna uma *400 Solicitação inválida ViewResult* com os dados apropriados.
- Quando `ModelState.IsValid` é `true`:
  - O método `Add` no repositório é chamado.
  - Um `RedirectToActionResult` é retornado com os argumentos corretos.

O estado de modelo inválido é testado por meio da adição de erros usando `AddModelError`, conforme mostrado no primeiro teste abaixo:

```

[Fact]
public async Task IndexPost_ReturnsBadRequestResult_WhenModelStateIsInvalid()
{
    // Arrange
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.ListAsync())
        .ReturnsAsync(GetTestSessions());
    var controller = new HomeController(mockRepo.Object);
    controller.ModelState.AddModelError("SessionName", "Required");
    var newSession = new HomeController.NewSessionModel();

    // Act
    var result = await controller.Index(newSession);

    // Assert
    var badRequestResult = Assert.IsType<BadRequestObjectResult>(result);
    Assert.IsType<SerializableError>(badRequestResult.Value);
}

[Fact]
public async Task IndexPost_ReturnsARedirectAndAddsSession_WhenModelStateIsValid()
{
    // Arrange
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.AddAsync(It.IsAny<BrainstormSession>()))
        .Returns(Task.CompletedTask)
        .Verifiable();
    var controller = new HomeController(mockRepo.Object);
    var newSession = new HomeController.NewSessionModel()
    {
        SessionName = "Test Name"
    };

    // Act
    var result = await controller.Index(newSession);

    // Assert
    var redirectToActionResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Null(redirectToActionResult.ControllerName);
    Assert.Equal("Index", redirectToActionResult.ActionName);
    mockRepo.Verify();
}

```

Quando `ModelState` não for válido, o mesmo `ViewResult` será retornado para uma solicitação GET. O teste não tenta passar um modelo inválido. Passar um modelo inválido não é uma abordagem válida, visto que o model binding não está em execução (embora um [teste de integração](#) use model binding). Nesse caso, o model binding não está sendo testado. Esses testes de unidade estão testando apenas o código no método de ação.

O segundo teste verifica se, quando o `ModelState` é válido:

- Um novo `BrainstormSession` é adicionado (por meio do repositório).
- O método retorna um `RedirectToActionResult` com as propriedades esperadas.

Chamadas fictícias que não são chamadas são normalmente ignoradas, mas a chamada a `Verifiable` no final da chamada de instalação permite a validação fictícia no teste. Isso é realizado com a chamada a `mockRepo.Verify`, que não será aprovada no teste se o método esperado não tiver sido chamado.

#### NOTE

A biblioteca do Moq usada neste exemplo possibilita a combinação de simulações verificáveis ou "estritas" com simulações não verificáveis (também chamadas de simulações "flexíveis" ou stubs). Saiba mais sobre como [personalizar o comportamento de Simulação com o Moq](#).

[SessionController](#) no exemplo de aplicativo exibe informações relacionadas a uma sessão de debate específica. O controlador inclui lógica para lidar com valores `id` inválidos (há dois cenários `return` no exemplo a seguir para abordar esses cenários). A última instrução `return` retorna um novo `StormSessionViewModel` para a exibição (*Controllers/SessionController.cs*):

```
public class SessionController : Controller
{
    private readonly IBrainstormSessionRepository _sessionRepository;

    public SessionController(IBrainstormSessionRepository sessionRepository)
    {
        _sessionRepository = sessionRepository;
    }

    public async Task<IActionResult> Index(int? id)
    {
        if (!id.HasValue)
        {
            return RedirectToAction(actionName: nameof(Index),
                controllerName: "Home");
        }

        var session = await _sessionRepository.GetByIdAsync(id.Value);
        if (session == null)
        {
            return Content("Session not found.");
        }

        var viewModel = new StormSessionViewModel()
        {
            DateCreated = session.DateCreated,
            Name = session.Name,
            Id = session.Id
        };

        return View(viewModel);
    }
}
```

Os testes de unidade incluem um teste para cada cenário `return` na ação `Index` do controlador `Session`:

```

[Fact]
public async Task IndexReturnsARedirectToIndexHomeWhenIdIsNull()
{
    // Arrange
    var controller = new SessionController(sessionRepository: null);

    // Act
    var result = await controller.Index(id: null);

    // Assert
    var redirectToActionResult =
        Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Home", redirectToActionResult.ControllerName);
    Assert.Equal("Index", redirectToActionResult.ActionName);
}

[Fact]
public async Task IndexReturnsContentWithSessionNotFoundWhenSessionNotFound()
{
    // Arrange
    int testSessionId = 1;
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync((BrainstormSession)null);
    var controller = new SessionController(mockRepo.Object);

    // Act
    var result = await controller.Index(testSessionId);

    // Assert
    var contentResult = Assert.IsType<ContentResult>(result);
    Assert.Equal("Session not found.", contentResult.Content);
}

[Fact]
public async Task IndexReturnsViewResultWithStormSessionViewModel()
{
    // Arrange
    int testSessionId = 1;
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync(GetTestSessions().FirstOrDefault(
            s => s.Id == testSessionId));
    var controller = new SessionController(mockRepo.Object);

    // Act
    var result = await controller.Index(testSessionId);

    // Assert
    var viewResult = Assert.IsType<ViewResult>(result);
    var model = Assert.IsType<StormSessionViewModel>(
        viewResult.ViewData.Model);
    Assert.Equal("Test One", model.Name);
    Assert.Equal(2, model.DateCreated.Day);
    Assert.Equal(testSessionId, model.Id);
}

```

Mudando para o controlador Ideas, o aplicativo expõe a funcionalidade como uma API Web na rota `api/ideas`:

- Uma lista de ideias (`IdeaDTO`) associada com uma sessão de debate é retornada pelo método `ForSession`.
- O método `Create` adiciona novas ideias a uma sessão.

```

[HttpGet("forsession/{sessionId}")]
public async Task<IActionResult> ForSession(int sessionId)
{
    var session = await _sessionRepository.GetByIdAsync(sessionId);
    if (session == null)
    {
        return NotFound(sessionId);
    }

    var result = session.Ideas.Select(idea => new IdeaDTO()
    {
        Id = idea.Id,
        Name = idea.Name,
        Description = idea.Description,
        DateCreated = idea.DateCreated
    }).ToList();

    return Ok(result);
}

[HttpPost("create")]
public async Task<IActionResult> Create([FromBody]NewIdeaModel model)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var session = await _sessionRepository.GetByIdAsync(model.SessionId);
    if (session == null)
    {
        return NotFound(model.SessionId);
    }

    var idea = new Idea()
    {
        DateCreated = DateTimeOffset.Now,
        Description = model.Description,
        Name = model.Name
    };
    session.AddIdea(idea);

    await _sessionRepository.UpdateAsync(session);

    return Ok(session);
}

```

Evite retornar entidades de domínio de negócios diretamente por meio de chamadas à API. Entidades de domínio:

- Geralmente incluem mais dados do que o cliente necessita.
- Acopla desnecessariamente o modelo de domínio interno do aplicativo à API exposta publicamente.

É possível executar o mapeamento entre entidades de domínio e os tipos retornados ao cliente:

- Manualmente com um LINQ `Select`, como o aplicativo de exemplo usa. Para saber mais, consulte [LINQ \(Consulta Integrada à Linguagem\)](#).
- Automaticamente com uma biblioteca, como [AutoMapper](#).

Em seguida, o aplicativo de exemplo demonstra os testes de unidade para os métodos de API `Create` e `ForSession` do controlador Ideas.

O aplicativo de exemplo contém dois testes `ForSession`. O primeiro teste determina se `ForSession` retorna um

`NotFoundObjectResult` (HTTP não encontrado) para uma sessão inválida:

```
[Fact]
public async Task ForSession_ReturnsHttpNotFound_ForInvalidSession()
{
    // Arrange
    int testSessionId = 123;
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync((BrainstormSession)null);
    var controller = new IdeasController(mockRepo.Object);

    // Act
    var result = await controller.ForSession(testSessionId);

    // Assert
    var notFoundObjectResult = Assert.IsType<NotFoundObjectResult>(result);
    Assert.Equal(testSessionId, notFoundObjectResult.Value);
}
```

O segundo teste `ForSession` determina se `ForSession` retorna uma lista de ideias de sessão (`<List<IdeaDTO>>`) para uma sessão válida. As verificações também examinam a primeira ideia para confirmar se sua propriedade `Name` está correta:

```
[Fact]
public async Task ForSession_ReturnsIdeasForSession()
{
    // Arrange
    int testSessionId = 123;
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync(GetTestSession());
    var controller = new IdeasController(mockRepo.Object);

    // Act
    var result = await controller.ForSession(testSessionId);

    // Assert
    var okResult = Assert.IsType<OkObjectResult>(result);
    var returnValue = Assert.IsType<List<IdeaDTO>>(okResult.Value);
    var idea = returnValue.FirstOrDefault();
    Assert.Equal("One", idea.Name);
}
```

Para testar o comportamento do método `Create` quando o `ModelState` é inválido, o aplicativo de exemplo adiciona um erro de modelo ao controlador como parte do teste. Não tente testar a validação de modelos ou o model binding em testes de unidade—teste apenas o comportamento do método de ação quando houver um `ModelState` inválido:

```

[Fact]
public async Task Create_ReturnsBadRequest_GivenInvalidModel()
{
    // Arrange & Act
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    var controller = new IdeasController(mockRepo.Object);
    controller.ModelState.AddModelError("error", "some error");

    // Act
    var result = await controller.Create(model: null);

    // Assert
    Assert.IsType<BadRequestObjectResult>(result);
}

```

O segundo teste de `Create` depende do repositório retornar `null`, portanto, o repositório fictício é configurado para retornar `null`. Não é necessário criar um banco de dados de teste (na memória ou de outro tipo) e construir uma consulta que retornará esse resultado. O teste pode ser realizado em uma única instrução, como mostrado no código de exemplo:

```

[Fact]
public async Task Create_ReturnsHttpNotFound_ForInvalidSession()
{
    // Arrange
    int testSessionId = 123;
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync((BrainstormSession)null);
    var controller = new IdeasController(mockRepo.Object);

    // Act
    var result = await controller.Create(new NewIdeaModel());

    // Assert
    Assert.IsType<NotFoundObjectResult>(result);
}

```

O terceiro teste `Create`, `Create_ReturnsNewlyCreatedIdeaForSession`, verifica se o método `UpdateAsync` do repositório é chamado. A simulação é chamada com `Verifiable` e o método `Verify` do repositório fictício é chamado para confirmar se o método verificável é executado. Não é responsabilidade do teste de unidade garantir que o método `updateAsync` salve os dados—isso pode ser realizado com um teste de integração.

```

[Fact]
public async Task Create_ReturnsNewlyCreatedIdeaForSession()
{
    // Arrange
    int testSessionId = 123;
    string testName = "test name";
    string testDescription = "test description";
    var testSession = GetTestSession();
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync(testSession);
    var controller = new IdeasController(mockRepo.Object);

    var newIdea = new NewIdeaModel()
    {
        Description = testDescription,
        Name = testName,
        SessionId = testSessionId
    };
    mockRepo.Setup(repo => repo.UpdateAsync(testSession))
        .Returns(Task.CompletedTask)
        .Verifiable();

    // Act
    var result = await controller.Create(newIdea);

    // Assert
    var okResult = Assert.IsType<OkObjectResult>(result);
    var returnSession = Assert.IsType<BrainstormSession>(okResult.Value);
    mockRepo.Verify();
    Assert.Equal(2, returnSession.Ideas.Count());
    Assert.Equal(testName, returnSession.Ideas.LastOrDefault().Name);
    Assert.Equal(testDescription, returnSession.Ideas.LastOrDefault().Description);
}

```

## Testar ActionResult<T>

No ASP.NET Core 2.1 ou posterior, o [ActionResult<T>](#) ([ActionResult<TValue>](#)) permite que você retorne um tipo derivado de `ActionResult` ou retorne um tipo específico.

O aplicativo de exemplo inclui um método que retorna um `List<IdeaDTO>` para uma sessão `id` determinada. Se a sessão `id` não existir, o controlador retornará [NotFound](#):

```

[HttpGet("forsessionactionresult/{sessionId}")]
[ProducesResponseType(200)]
[ProducesResponseType(404)]
public async Task<ActionResult<List<IdeaDTO>>> ForSessionActionResult(int sessionId)
{
    var session = await _sessionRepository.GetByIdAsync(sessionId);

    if (session == null)
    {
        return NotFound(sessionId);
    }

    var result = session.Ideas.Select(idea => new IdeaDTO()
    {
        Id = idea.Id,
        Name = idea.Name,
        Description = idea.Description,
        DateCreated = idea.DateCreated
    }).ToList();

    return result;
}

```

Dois testes do controlador `ForSessionActionResult` estão incluídos no `ApiIdeasControllerTests`.

O primeiro teste confirma se o controlador retorna um `ActionResult`, mas não uma lista de ideias inexistente para uma sessão `id` inexistente:

- O tipo `ActionResult<List<IdeaDTO>>` é `ActionResult`.
- O `Result` é um `NotFoundObjectResult`.

```

[Fact]
public async Task ForSessionActionResult_ReturnsNotFoundObjectResultForNonexistentSession()
{
    // Arrange
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    var controller = new IdeasController(mockRepo.Object);
    var nonExistentSessionId = 999;

    // Act
    var result = await controller.ForSessionActionResult(nonExistentSessionId);

    // Assert
    var actionResult = Assert.IsType<ActionResult<List<IdeaDTO>>>(result);
    Assert.IsType<NotFoundObjectResult>(actionResult.Result);
}

```

Para uma sessão `id` válida, o segundo teste confirma se o método retorna:

- Um `ActionResult` com um tipo `List<IdeaDTO>`.
- O `ActionResult<T>.Value` é um tipo `List<IdeaDTO>`.
- O primeiro item na lista é uma ideia válida que corresponde à ideia armazenada na sessão fictícia (obtida chamando `GetTestSession`).

```

[Fact]
public async Task ForSessionActionResult_ReturnsIdeasForSession()
{
    // Arrange
    int testSessionId = 123;
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync(GetTestSession());
    var controller = new IdeasController(mockRepo.Object);

    // Act
    var result = await controller.ForSessionActionResult(testSessionId);

    // Assert
    var actionResult = Assert.IsType<ActionResult<List<IdeaDTO>>>(result);
    var returnValue = Assert.IsType<List<IdeaDTO>>(actionResult.Value);
    var idea = returnValue.FirstOrDefault();
    Assert.Equal("One", idea.Name);
}

```

O aplicativo de exemplo também inclui um método para criar um novo `Idea` para uma determinada sessão. O controlador retorna:

- `BadRequest` para um modelo inválido.
- `NotFound` se a sessão não existir.
- `CreatedAtAction` quando a sessão for atualizada com a nova ideia.

```

[HttpPost("createactionresult")]
[ProducesResponseType(201)]
[ProducesResponseType(400)]
[ProducesResponseType(404)]
public async Task<ActionResult<BrainstormSession>> CreateActionResult([FromBody]NewIdeaModel model)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var session = await _sessionRepository.GetByIdAsync(model.SessionId);

    if (session == null)
    {
        return NotFound(model.SessionId);
    }

    var idea = new Idea()
    {
        DateCreated = DateTimeOffset.Now,
        Description = model.Description,
        Name = model.Name
    };
    session.AddIdea(idea);

    await _sessionRepository.UpdateAsync(session);

    return CreatedAtAction(nameof(CreateActionResult), new { id = session.Id }, session);
}

```

Três testes `CreateActionResult` estão incluídos no `ApiIdeasControllerTests`.

O primeiro texto confirma que um `BadRequest` é retornado para um modelo inválido.

```

[Fact]
public async Task CreateActionResult_ReturnsBadRequest_GivenInvalidModel()
{
    // Arrange & Act
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    var controller = new IdeasController(mockRepo.Object);
    controller.ModelState.AddModelError("error", "some error");

    // Act
    var result = await controller.CreateActionResult(model: null);

    // Assert
    var actionResult = Assert.IsType<ActionResult<BrainstormSession>>(result);
    Assert.IsType<BadRequestObjectResult>(actionResult.Result);
}

```

O segundo teste verifica se um `NotFound` será retornado se a sessão não existir.

```

[Fact]
public async Task CreateActionResult_ReturnsNotFoundObjectResultForNonexistentSession()
{
    // Arrange
    var nonExistentSessionId = 999;
    string testName = "test name";
    string testDescription = "test description";
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    var controller = new IdeasController(mockRepo.Object);

    var newIdea = new NewIdeaModel()
    {
        Description = testDescription,
        Name = testName,
        SessionId = nonExistentSessionId
    };

    // Act
    var result = await controller.CreateActionResult(newIdea);

    // Assert
    var actionResult = Assert.IsType<ActionResult<BrainstormSession>>(result);
    Assert.IsType<NotFoundObjectResult>(actionResult.Result);
}

```

Para uma sessão `id` válida, o teste final confirmará se:

- O método retorna um `ActionResult` com um tipo `BrainstormSession`.
- O `ActionResult<T>.Result` é um `CreatedAtActionResult`. `CreatedAtActionResult` é semelhante à resposta `201 Criado` com um cabeçalho `Location`.
- O `ActionResult<T>.Value` é um tipo `BrainstormSession`.
- A chamada fictícia para atualizar a sessão, `UpdateAsync(testSession)`, foi invocada. A chamada de método `Verifiable` é verificada por meio da execução de `mockRepo.Verify()` nas declarações.
- Dois objetos `Idea` são retornados para a sessão.
- O último item (o `Idea` adicionado pela chamada fictícia a `UpdateAsync`) corresponde ao `newIdea` adicionado à sessão no teste.

```

[Fact]
public async Task CreateActionResult_ReturnsNewlyCreatedIdeaForSession()
{
    // Arrange
    int testSessionId = 123;
    string testName = "test name";
    string testDescription = "test description";
    var testSession = GetTestSession();
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync(testSession);
    var controller = new IdeasController(mockRepo.Object);

    var newIdea = new NewIdeaModel()
    {
        Description = testDescription,
        Name = testName,
        SessionId = testSessionId
    };
    mockRepo.Setup(repo => repo.UpdateAsync(testSession))
        .Returns(Task.CompletedTask)
        .Verifiable();

    // Act
    var result = await controller.CreateActionResult(newIdea);

    // Assert
    var actionResult = Assert.IsType<ActionResult<BrainstormSession>>(result);
    var createdAtIndexResult = Assert.IsType<CreatedAtActionResult>(actionResult.Result);
    var returnValue = Assert.IsType<BrainstormSession>(createdAtIndexResult.Value);
    mockRepo.Verify();
    Assert.Equal(2, returnValue.Ideas.Count());
    Assert.Equal(testName, returnValue.Ideas.LastOrDefault().Name);
    Assert.Equal(testDescription, returnValue.Ideas.LastOrDefault().Description);
}

```

## Recursos adicionais

- [Testes de integração no ASP.NET Core](#)
- [Crie e execute testes de unidade com o Visual Studio.](#)
- [Princípio de Dependências Explícitas](#)

# Testes de integração no ASP.NET Core

12/01/2019 • 33 minutes to read • [Edit Online](#)

Por [Luke Latham](#) e [Steve Smith](#)

Testes de integração Certifique-se de que os componentes do aplicativo funcionam corretamente em um nível que inclui a infraestrutura de suporte do aplicativo, como o banco de dados, o sistema de arquivos e a rede. ASP.NET Core dá suporte a testes de integração usando uma estrutura de teste de unidade com um host de teste da web e um servidor de teste na memória.

Este tópico pressupõe uma compreensão básica dos testes de unidade. Se estiver familiarizado com os conceitos de teste, consulte o [testes de unidade no .NET Core e .NET Standard](#) tópico e seu conteúdo vinculado.

## [Exibir ou baixar código de exemplo \(como baixar\)](#)

O aplicativo de exemplo é um aplicativo páginas Razor e pressupõe uma compreensão básica de páginas do Razor. Se estiver familiarizado com as páginas Razor, consulte os tópicos a seguir:

- [Introdução a Páginas do Razor](#)
- [Introdução a Páginas do Razor](#)
- [Testes de unidades de páginas Razor](#)

### NOTE

Para SPAs de teste, é recomendável uma ferramenta, como [Selenium](#), que pode automatizar um navegador.

## Introdução aos testes de integração

Componentes de um aplicativo em um nível mais amplo do que avaliar a testes de integração [testes de unidade](#). Testes de unidade são usados para testar os componentes de software isolado, tais como métodos de classe individual. Testes de integração confirme que dois ou mais componentes de aplicativo funcionam em conjunto para produzir um resultado esperado, possivelmente incluindo todos os componentes necessários para processar totalmente uma solicitação.

Esses testes mais amplos são usados para testar a infraestrutura do aplicativo e o framework inteiro, muitas vezes, incluindo os seguintes componentes:

- Banco de Dados
- Sistema de arquivos
- Dispositivos de rede
- Pipeline de solicitação-resposta

Componentes de uso gerado, conhecidos como testes de unidade *fakes* ou *objetos fictícios*, no lugar de componentes de infraestrutura.

Em contraste com testes de unidade, testes de integração:

- Use os componentes reais usados pelo aplicativo em produção.
- Exigem mais código e processamento de dados.
- Pode levar mais tempo.

Portanto, limite o uso de testes de integração para os cenários mais importantes de infraestrutura. Se um comportamento pode ser testado usando um teste de unidade ou um teste de integração, escolha o teste de unidade.

#### TIP

Não escreva testes de integração para cada permutação possíveis de acesso de dados e arquivos com sistemas de arquivos e bancos de dados. Independentemente de quantos coloca em um aplicativo de interagir com bancos de dados e sistemas de arquivos, um conjunto com foco de leitura, gravação, atualização e exclusão integração testes são geralmente capazes de banco de dados de teste adequadamente e componentes do sistema de arquivos. Testes de unidade de uso para testes de rotina da lógica do método que interagem com esses componentes. Em testes de unidade, o uso da infra-estrutura de falsificações/simulações resultado na execução de teste mais rápido.

#### NOTE

Em discussões de testes de integração, o projeto testado com frequência é chamado de *sistema em teste*, ou "SUT" de forma abreviada.

## Testes de integração do ASP.NET Core

Testes de integração no ASP.NET Core requerem o seguinte:

- Um projeto de teste é usado para conter e executar os testes. O projeto de teste tem uma referência ao projeto ASP.NET Core testada, chamado de *sistema em teste* (SUT). "SUT" é usado ao longo deste tópico para referir-se para o aplicativo testado.
- O projeto de teste cria um host da web de teste do SUT e usa um cliente do servidor de teste para lidar com solicitações e respostas para o SUT.
- Um executor de teste é usado para executar os testes e o relatório de resultados de teste.

Testes de integração seguem uma sequência de eventos que incluem o usual *organizar*, *Act*, e *Assert* etapas de teste:

1. Host de web do SUT está configurado.
2. Um cliente do servidor de teste é criado para enviar solicitações para o aplicativo.
3. O *organizar* etapa de teste é executada: O aplicativo de teste prepara uma solicitação.
4. O *Act* etapa de teste é executada: O cliente envia a solicitação e recebe a resposta.
5. O *Assert* etapa de teste é executada: O *reais* resposta é validada como um *passar* ou *falhar* com base em um *esperado* resposta.
6. O processo continua até que todos os testes são executados.
7. Os resultados do teste são relatados.

Normalmente, o host de teste da web está configurado diferentemente do host do aplicativo web normal para o teste é executado. Por exemplo, configurações de aplicativo diferente ou outro banco de dados podem ser usadas para os testes.

Componentes de infraestrutura, como o host do teste da web e o servidor de teste na memória ([TestServer](#)), são fornecidas ou gerenciados pelo [Microsoft.AspNetCore.Mvc.Testing](#) pacote. O uso desse pacote simplifica a criação de teste e execução.

O `Microsoft.AspNetCore.Mvc.Testing` pacote lida com as seguintes tarefas:

- Copia o arquivo de dependências (\*.deps) do SUT para o projeto de teste *bin* pasta.
- Define a raiz do conteúdo raiz do projeto do SUT para que os arquivos estáticos e páginas/exibições são

encontradas quando os testes são executados.

- Fornece o [WebApplicationFactory](#) classe para simplificar a inicialização com o SUT `TestServer`.

O [testes de unidade](#) documentação descreve como configurar um projeto e teste de executor de teste, junto com instruções detalhadas sobre como executar testes e recomendações sobre como a testes de nome e classes de teste.

#### NOTE

Ao criar um projeto de teste para um aplicativo, separe os testes de unidade dos testes de integração em projetos diferentes. Isso ajuda a garantir que o teste componentes da infraestrutura não acidentalmente incluídos nos testes de unidade. Também permite a separação dos testes de unidade e integração de controle sobre qual conjunto de testes são executados.

Não há praticamente nenhuma diferença entre a configuração para testes de aplicativos de páginas do Razor e aplicativos MVC. A única diferença está em como os testes são nomeados. Em um aplicativo de páginas do Razor, os testes de pontos de extremidade de página geralmente são nomeadas depois a classe de modelo de página (por exemplo, `IndexPageTests` para testar a integração do componente para a página de índice). Em um aplicativo MVC, testes são geralmente organizadas por classes de controlador e os controladores de testam por eles em homenagem (por exemplo, `HomeControllerTests` para testar a integração do componente para o controlador Home).

## Pré-requisitos de aplicativo de teste

O projeto de teste deve:

- Referenciar os seguintes pacotes:
  - [Microsoft.AspNetCore.App](#)
  - [Microsoft.AspNetCore.Mvc.Testing](#)
- Especifique o SDK para Web no arquivo de projeto (`<Project Sdk="Microsoft.NET.Sdk.Web">`). O SDK para Web é necessário ao fazer referência a [metapacote Microsoft](#).

Esses pré-requisitos podem ser vistos na [aplicativo de exemplo](#). Inspecione o `tests/RazorPagesProject.Tests/RazorPagesProject.Tests.csproj` arquivo. O aplicativo de exemplo usa o [xUnit](#) estrutura de teste e o [AngleSharp](#) biblioteca do analisador, portanto, o aplicativo de exemplo também referencia:

- [xUnit](#)
- [xUnit](#)
- [AngleSharp](#)

## Testes básicos com o padrão WebApplicationFactory

[WebApplicationFactory](#) <`TEntryPoint`> é usado para criar um [TestServer](#) para os testes de integração.

`TEntryPoint` é a classe de ponto de entrada do SUT, geralmente o `Startup` classe.

Teste as classes implementam uma *acessório de classe* interface ([IClassFixture](#)) para indicar a classe contém testes e fornece instâncias de objeto compartilhado entre os testes na classe.

### Teste básico de pontos de extremidade do aplicativo

O seguinte teste de classe, `BasicTests`, usa o [WebApplicationFactory](#) para inicializar o SUT e fornecer uma [HttpClient](#) para um método de teste, `Get_EndpointsReturnSuccessAndCorrectContentType`. O método verifica se o código de status de resposta for bem-sucedida (códigos de status no intervalo 200-299) e o `Content-Type` cabeçalho é `text/html; charset=utf-8` para várias páginas de aplicativo.

`CreateClient` cria uma instância de `HttpClient` que segue redirecionamentos e manipula cookies automaticamente.

```
public class BasicTests
    : IClassFixture<WebApplicationFactory<RazorPagesProject.Startup>>
{
    private readonly WebApplicationFactory<RazorPagesProject.Startup> _factory;

    public BasicTests(WebApplicationFactory<RazorPagesProject.Startup> factory)
    {
        _factory = factory;
    }

    [Theory]
    [InlineData("/")]
    [InlineData("/Index")]
    [InlineData("/About")]
    [InlineData("/Privacy")]
    [InlineData("/Contact")]
    public async Task Get_EndpointsReturnSuccessAndCorrectContentType(string url)
    {
        // Arrange
        var client = _factory.CreateClient();

        // Act
        var response = await client.GetAsync(url);

        // Assert
        response.EnsureSuccessStatusCode(); // Status Code 200-299
        Assert.Equal("text/html; charset=utf-8",
            response.Content.Headers.ContentType.ToString());
    }
}
```

## Testar um ponto de extremidade seguro

Outro teste no `BasicTests` classe verifica que um ponto de extremidade seguro redireciona um usuário não autenticado à página de logon do aplicativo.

No SUT, o `/SecurePage` página usa um `AuthorizePage` convenção para aplicar um `AuthorizeFilter` para a página. Para obter mais informações, consulte [convenções de autorização de páginas do Razor](#).

```
services.AddMvc()
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_2)
    .AddRazorPagesOptions(options =>
{
    options.Conventions.AuthorizePage("/SecurePage");
});
```

No `Get_SecurePageRequiresAnAuthenticatedUser` testar, uma `WebApplicationFactoryClientOptions` está definido para não permitir redirecionamentos definindo `AllowAutoRedirect` para `false`:

```

[Fact]
public async Task Get_SecurePageRequiresAnAuthenticatedUser()
{
    // Arrange
    var client = _factory.CreateClient(
        new WebApplicationFactoryClientOptions
    {
        AllowAutoRedirect = false
    });

    // Act
    var response = await client.GetAsync("/SecurePage");

    // Assert
    Assert.Equal(HttpStatusCode.Redirect, response.StatusCode);
    Assert.StartsWith("http://localhost/Identity/Account/Login",
        response.Headers.Location.OriginalString);
}

```

Não permitindo que o cliente para seguir o redirecionamento, podem ser feitas as seguintes verificações:

- O código de status retornado pelo SUT pode ser verificado em relação a esperada `(HttpStatusCode.Redirect)` resultado, não o código de status final após o redirecionamento para a página de logon, o que seria `Httpstatuscode`.
- O `Location` valor de cabeçalho nos cabeçalhos de resposta é verificado para confirmar que ela começa com `http://localhost/Identity/Account/Login`, não a logon página resposta final, onde o `Location` cabeçalho não está presente.

Para obter mais informações sobre `WebApplicationFactoryClientOptions`, consulte o [opções de cliente](#) seção.

## Personalizar WebApplicationFactory

Configuração do host da Web pode ser criada independentemente das classes de teste herdando de `WebApplicationFactory` para criar um ou mais fábricas personalizadas:

1. Herdar `WebApplicationFactory` e substituir `ConfigureWebHost`. O `IWebHostBuilder` permite a configuração da coleção com o serviço `ConfigureServices`:

```

public class CustomWebApplicationFactory<TStartup>
    : WebApplicationFactory<TStartup> where TStartup: class
{
    protected override void ConfigureWebHost(IWebHostBuilder builder)
    {
        builder.ConfigureServices(services =>
        {
            // Create a new service provider.
            var serviceProvider = new ServiceCollection()
                .AddEntityFrameworkInMemoryDatabase()
                .BuildServiceProvider();

            // Add a database context (ApplicationContext) using an in-memory
            // database for testing.
            services.AddDbContext<ApplicationContext>(options =>
            {
                options.UseInMemoryDatabase("InMemoryDbForTesting");
                options.UseInternalServiceProvider(serviceProvider);
            });

            // Build the service provider.
            var sp = services.BuildServiceProvider();

            // Create a scope to obtain a reference to the database
            // context (ApplicationContext).
            using (var scope = sp.CreateScope())
            {
                var scopedServices = scope.ServiceProvider;
                var db = scopedServices.GetRequiredService<ApplicationContext>();
                var logger = scopedServices
                    .GetRequiredService<ILogger<CustomWebApplicationFactory<TStartup>>>();

                // Ensure the database is created.
                db.Database.EnsureCreated();

                try
                {
                    // Seed the database with test data.
                    Utilities.InitializeDbForTests(db);
                }
                catch (Exception ex)
                {
                    logger.LogError(ex, $"An error occurred seeding the " +
                        "database with test messages. Error: {ex.Message}");
                }
            }
        });
    }
}

```

A propagação do banco de dados de [aplicativo de exemplo](#) é executada pelo `InitializeDbForTests` método. O método é descrito o [exemplo para testes de integração: Organização do aplicativo de teste](#) seção.

2. Usar o custom `CustomWebApplicationFactory` em classes de teste. O exemplo a seguir usa o alocador no `IndexPageTests` classe:

```

public class IndexPageTests :
    IClassFixture<CustomWebApplicationFactory<RazorPagesProject.Startup>>
{
    private readonly HttpClient _client;
    private readonly CustomWebApplicationFactory<RazorPagesProject.Startup>
        _factory;

    public IndexPageTests(
        CustomWebApplicationFactory<RazorPagesProject.Startup> factory)
    {
        _factory = factory;
        _client = factory.CreateClient(new WebApplicationFactoryClientOptions
        {
            AllowAutoRedirect = false
        });
    }
}

```

Cliente do aplicativo de exemplo está configurado para impedir o `HttpClient` de seguir redirecionamentos. Conforme explicado a [testar um ponto de extremidade seguro](#) seção, isso permite que os testes para verificar o resultado da primeira de resposta do aplicativo. A primeira resposta é um redirecionamento em muitos desses testes com um `Location` cabeçalho.

3. Um teste típico usa a `HttpClient` e métodos auxiliares para processar a solicitação e resposta:

```

[Fact]
public async Task Post_DeleteAllMessagesHandler_ReturnsRedirectToRoot()
{
    // Arrange
    var defaultPage = await _client.GetAsync("/");
    var content = await HtmlHelpers.GetDocumentAsync(defaultPage);

    //Act
    var response = await _client.SendAsync(
        (IHtmlFormElement)content.QuerySelector("form[id='messages']"),
        (IHtmlButtonElement)content.QuerySelector("button[id='deleteAllBtn']"));

    // Assert
    Assert.Equal(HttpStatusCode.OK, defaultPage.StatusCode);
    Assert.Equal(HttpStatusCode.Redirect, response.StatusCode);
    Assert.Equal("/", response.Headers.Location.OriginalString);
}

```

Todas as solicitações POST para o SUT devem satisfazer a verificação de antifalsificação é feita automaticamente pelo aplicativo do [sistema de proteção de dados antifalsificação](#). Para organizar para solicitação POST de um teste, o aplicativo de teste deve:

1. Faça uma solicitação para a página.
2. Analise o cookie antifalsificação e o token de validação de solicitação da resposta.
3. Fazer a solicitação POST com a validação antifalsificação do cookie e solicitação de token em vigor.

O `SendAsync` métodos de extensão auxiliar (`Helpers/HttpClientExtensions.cs`) e o `GetDocumentAsync` método auxiliar (`Helpers/HtmlHelpers.cs`) no [do aplicativo de exemplo](#) usar o [AngleSharp](#) analisador para lidar com a verificação de antiforgery com os seguintes métodos:

- `GetDocumentAsync` – Recebe o `HttpResponseMessage` e retorna um `IHtmlDocument`. `GetDocumentAsync` usa uma fábrica que prepara uma *resposta virtual* com base no original `HttpResponseMessage`. Para obter mais informações, consulte o [AngleSharp documentação](#).
- `SendAsync` métodos de extensão para o `HttpClient` compor uma `HttpRequestMessage` e chame `SendAsync(HttpRequestMessage)` para enviar solicitações para o SUT. Sobrecargas `SendAsync` aceite o

formulário HTML (`IHtmlFormElement`) e o seguinte:

- Enviar o botão do formulário (`IHtmlElement`)
- Coleção de valores de formulário (`IEnumerable<KeyValuePair<string, string>>`)
- Botão enviar (`IHtmlElement`) e valores de formulário (`IEnumerable<KeyValuePair<string, string>>`)

#### NOTE

[AngleSharp](#) é um terceiro biblioteca usada para fins de demonstração neste tópico e o aplicativo de exemplo de análise.

AngleSharp não é suportado ou necessários para os testes de integração de aplicativos ASP.NET Core. Outros analisadores podem usados, como o [Pack de agilidade de Html \(HAP\)](#). Outra abordagem é escrever código para lidar com o token de verificação de solicitação e o cookie antifalsificação o sistema antifalsificação diretamente.

## Personalizar o cliente com WithWebHostBuilder

Quando a configuração adicional é necessária dentro de um método de teste `WithWebHostBuilder` cria uma nova `WebApplicationFactory` com um `IWebHostBuilder` que é ainda mais personalizado pela configuração.

O `Post_DeleteMessageHandler_ReturnsRedirectToRoot` método de teste a [aplicativo de exemplo](#) demonstra o uso de `WithWebHostBuilder`. Esse teste executa uma exclusão de registro no banco de dados ao disparar um envio de formulário no SUT.

Porque outro teste na `IndexPageTests` classe executa uma operação que exclui todos os registros no banco de dados e podem ser executadas antes do `Post_DeleteMessageHandler_ReturnsRedirectToRoot` método, o banco de dados é propagado nesse método de teste para garantir que um registro está presente para o SUT excluir. Selecionando o `deleteBtn1` botão do `messages` formulário no SUT é simulado na solicitação para o SUT:

```

[Fact]
public async Task Post_DeleteMessageHandler_ReturnsRedirectToRoot()
{
    // Arrange
    var client = _factory.WithWebHostBuilder(builder =>
    {
        builder.ConfigureServices(services =>
        {
            var serviceProvider = services.BuildServiceProvider();

            using (var scope = serviceProvider.CreateScope())
            {
                var scopedServices = scope.ServiceProvider;
                var db = scopedServices
                    .GetRequiredService<ApplicationDbContext>();
                var logger = scopedServices
                    .GetRequiredService<ILogger<IndexPageTests>>();

                try
                {
                    Utilities.InitializeDbForTests(db);
                }
                catch (Exception ex)
                {
                    logger.LogError(ex, "An error occurred seeding " +
                        "the database with test messages. Error: " +
                        ex.Message);
                }
            }
        });
    })
    .CreateClient(new WebApplicationFactoryClientOptions
    {
        AllowAutoRedirect = false
    });
    var defaultPage = await client.GetAsync("/");
    var content = await HtmlHelpers.GetDocumentAsync(defaultPage);

    //Act
    var response = await client.SendAsync(
        (IHtmlFormElement)content.QuerySelector("form[id='messages']"),
        (IHtmlButtonElement)content.QuerySelector("button[id='deleteBtn1']"));

    // Assert
    Assert.Equal(HttpStatusCode.OK, defaultPage.StatusCode);
    Assert.Equal(HttpStatusCode.Redirect, response.StatusCode);
    Assert.Equal("/", response.Headers.Location.OriginalString);
}

```

## Opções do cliente

A tabela a seguir mostra o padrão [WebApplicationFactoryClientOptions](#) disponíveis ao criar [HttpClient](#) instâncias.

OPÇÃO	DESCRIÇÃO	PADRÃO
<a href="#">AllowAutoRedirect</a>	Obtém ou define se <a href="#">HttpClient</a> instâncias precisar seguir automaticamente as respostas de redirecionamento.	<code>true</code>

OPÇÃO	DESCRIÇÃO	PADRÃO
BaseAddress	Obtém ou define o endereço básico do <code>HttpClient</code> instâncias.	<code>http://localhost</code>
HandleCookies	Obtém ou define se <code>HttpClient</code> instâncias devem lidar com cookies.	<code>true</code>
MaxAutomaticRedirections	Obtém ou define o número máximo de respostas de redirecionamento que <code>HttpClient</code> instâncias devem seguir.	7

Criar o `WebApplicationFactoryClientOptions` de classe e passá-lo para o `CreateClient` método (padrão de valores são mostrados no exemplo de código):

```
// Default client option values are shown
var clientOptions = new WebApplicationFactoryClientOptions();
clientOptions.AllowAutoRedirect = true;
clientOptions.BaseAddress = new Uri("http://localhost");
clientOptions.HandleCookies = true;
clientOptions.MaxAutomaticRedirections = 7;

_client = _factory.CreateClient(clientOptions);
```

## Inserir serviços fictícios

Os serviços podem ser substituídos em um teste com uma chamada para `ConfigureTestServices` no construtor do host. **Para inserir os serviços de simulação, o SUT deve ter uma `Startup` classe com um `Startup.ConfigureServices` método.**

O exemplo SUT inclui um serviço com escopo definido que retorna uma citação. A cotação é inserida em um campo oculto na página de índice quando a página de índice é solicitada.

*Services/IQuoteService.cs:*

```
public interface IQuoteService
{
    Task<string> GenerateQuote();
}
```

*Services/QuoteService.cs:*

```
// Quote @1975 BBC: The Doctor (Tom Baker); Dr. Who: Planet of Evil
// https://www.bbc.co.uk/programmes/p00pyrx6
public class QuoteService : IQuoteService
{
    public Task<string> GenerateQuote()
    {
        return Task.FromResult<string>(
            "Come on, Sarah. We've an appointment in London, " +
            "and we're already 30,000 years late.");
    }
}
```

*Startup.cs:*

```
services.AddScoped<IQuoteService, QuoteService>();
```

#### Pages/Index.cshtml.cs:

```
public class IndexModel : PageModel
{
    private readonly ApplicationDbContext _db;
    private readonly IQuoteService _quoteService;

    public IndexModel(ApplicationDbContext db, IQuoteService quoteService)
    {
        _db = db;
        _quoteService = quoteService;
    }

    [BindProperty]
    public Message Message { get; set; }

    public IList<Message> Messages { get; private set; }

    [TempData]
    public string MessageAnalysisResult { get; set; }

    public string Quote { get; private set; }

    public async Task OnGetAsync()
    {
        Messages = await _db.GetMessagesAsync();

        Quote = await _quoteService.GenerateQuote();
    }
}
```

#### Pages/Index.cs:

```
<input id="quote" type="hidden" value="@Model.Quote">
```

A marcação a seguir é gerada quando o aplicativo SUT for executado:

```
<input id="quote" type="hidden" value="Come on, Sarah. We've an appointment in
London, and we're already 30,000 years late.">
```

Para testar a injeção de serviço e as aspas em um teste de integração, um serviço de simulação é injetado no SUT pelo teste. O serviço fictício substitui o aplicativo `QuoteService` com um serviço fornecido pelo aplicativo de teste, chamado `TestQuoteService`:

#### IntegrationTests.IndexPageTests.cs:

```
// Quote ©1975 BBC: The Doctor (Tom Baker); Pyramids of Mars
// https://www.bbc.co.uk/programmes/p00pys55
public class TestQuoteService : IQuoteService
{
    public Task<string> GenerateQuote()
    {
        return Task.FromResult<string>(
            "Something's interfering with time, Mr. Scarman, " +
            "and time is my business.");
    }
}
```

`ConfigureTestServices` é chamado, e o serviço com escopo está registrado:

```
[Fact]
public async Task Get_QuoteService_ProvidesQuoteInPage()
{
    // Arrange
    var client = _factory.WithWebHostBuilder(builder =>
    {
        builder.ConfigureTestServices(services =>
        {
            services.AddScoped<IQuoteService, TestQuoteService>();
        });
    })
    .CreateClient();

    //Act
    var defaultPage = await client.GetAsync("/");
    var content = await HtmlHelpers.GetDocumentAsync(defaultPage);
    var quoteElement = content.QuerySelector("#quote");

    // Assert
    Assert.Equal("Something's interfering with time, Mr. Scarman, " +
        "and time is my business.", quoteElement.Attributes["value"].Value);
}
```

A marcação produzida durante a execução do teste reflete o texto de cotação fornecido pelo `TestQuoteService`, portanto, os passos de asserção:

```
<input id="quote" type="hidden" value="Something's interfering with time,
Mr. Scarman, and time is my business.">
```

## Como a infraestrutura de teste infere o caminho de raiz do conteúdo do aplicativo

O `WebApplicationFactory` construtor infere o caminho de raiz do conteúdo do aplicativo procurando por um `WebApplicationFactoryContentRootAttribute` no assembly que contém os testes de integração com uma chave igual ao `TEntryPoint` assembly `System.Reflection.Assembly.FullName`. No caso de um atributo com a chave correta não ser encontrado, `WebApplicationFactory` reverterá para procurar por um arquivo de solução (`*.sln`) e anexa o `TEntryPoint` nome do assembly para o diretório da solução. O diretório raiz do aplicativo (o caminho raiz do conteúdo) é usado para descobrir os modos de exibição e arquivos de conteúdo.

Na maioria dos casos, ele não é necessário definir explicitamente a raiz do conteúdo de aplicativo, conforme a lógica de pesquisa geralmente encontra a raiz do conteúdo correta no tempo de execução. Em cenários especiais em que a raiz do conteúdo não for encontrada usando o algoritmo de pesquisa interna, o aplicativo de conteúdo raiz pode ser especificada explicitamente ou por meio de lógica personalizada. Para definir a raiz do conteúdo de aplicativo nesses cenários, chame o `useSolutionRelativeContentRoot` método de extensão do `testhost` pacote. Forneça um caminho relativo da solução e o padrão de nome ou glob do arquivo de solução opcional (padrão = `*.sln`).

Chame o `UseSolutionRelativeContentRoot` usando o método extensão *um* das seguintes abordagens:

- Ao configurar classes de teste com `WebApplicationFactory`, forneça uma configuração personalizada com o `IWebHostBuilder`:

```

public IndexPageTests(
    WebApplicationFactory<RazorPagesProject.Startup> factory)
{
    var _factory = factory.WithWebHostBuilder(builder =>
    {
        builder.UseSolutionRelativeContentRoot("<SOLUTION-RELATIVE-PATH>");

        ...
    });
}

```

- Ao configurar classes de teste com um personalizado `WebApplicationFactory`, herdam `WebApplicationFactory` e substituir `ConfigureWebHost`:

```

public class CustomWebApplicationFactory<TStartup>
    : WebApplicationFactory<RazorPagesProject.Startup>
{
    protected override void ConfigureWebHost(IWebHostBuilder builder)
    {
        builder.ConfigureServices(services =>
        {
            services.UseSolutionRelativeContentRoot("<SOLUTION-RELATIVE-PATH>");

            ...
        });
    }
}

```

## Desabilitar a cópia de sombra

A cópia de sombra faz com que os testes a serem executados em uma pasta diferente que a pasta de saída. Para testes funcione corretamente, a cópia de sombra deve ser desabilitada. O [aplicativo de exemplo](#) usa o xUnit e desabilita a cópia de sombra para xUnit, incluindo um `xunit.runner.json` arquivo com a configuração correta. Para obter mais informações, consulte [Configurando o xUnit.net com JSON](#).

Adicione a `xunit.runner.json` arquivo raiz do projeto de teste com o seguinte conteúdo:

```
{
    "shadowCopy": false
}
```

## Descarte de objetos

Após os testes do `IClassFixture` implementação são executadas, `TestServer` e `HttpClient` são descartados quando xUnit descarta o `WebApplicationFactory`. Se objetos instanciados pelo desenvolvedor exigem disposição, descartá-los no `IClassFixture` implementação. Para obter mais informações, consulte [implementando um método Dispose](#).

## Exemplo para testes de integração

O [aplicativo de exemplo](#) é composto de dois aplicativos:

APLICATIVO	PASTA DO PROJETO	DESCRIÇÃO
------------	------------------	-----------

APLICATIVO	PASTA DO PROJETO	DESCRIÇÃO
Aplicativo de mensagens (SUT)	<i>src/RazorPagesProject</i>	Permite que um usuário adicione, exclua uma, excluir todas as e analisar as mensagens.
Aplicativo de teste	<i>tests/RazorPagesProject.Tests</i>	Usado para teste de integração o SUT.

Os testes podem ser executados usando os recursos de teste interno de um IDE, como [Visual Studio](#). Se usando [Visual Studio Code](#) ou a linha de comando, execute o seguinte comando em um prompt de comando na *tests/RazorPagesProject.Tests* pasta:

```
dotnet test
```

### Organização de aplicativo (SUT) da mensagem

O SUT é um sistema de mensagem de páginas do Razor com as seguintes características:

- A página de índice do aplicativo (*Pages* e *Pages/Index.cshtml.cs*) fornece uma interface do usuário e a página de métodos de modelo para controlar a adição, exclusão e análise de mensagens (palavras médias por mensagem).
- Uma mensagem é descrita pela `Message` classe (*Data/Message.cs*) com duas propriedades: `Id` (chave) e `Text` (mensagem). O `Text` propriedade é necessária e é limitada a 200 caracteres.
- As mensagens são armazenadas usando [banco de dados do Entity Framework na memória](#)<sup>†</sup>.
- O aplicativo contém uma camada de acesso de dados (DAL) na sua classe de contexto do banco de dados, `AppDbContext` (*Data/AppDbContext.cs*).
- Se o banco de dados está vazio na inicialização do aplicativo, o repositório de mensagens é inicializado com três mensagens.
- O aplicativo inclui um `/SecurePage` que só pode ser acessado por um usuário autenticado.

<sup>†</sup>O tópico [EF testar com InMemory](#), explica como usar um banco de dados na memória para testes com MSTest. Este tópico usa o [xUnit](#) estrutura de teste. Conceitos de teste e teste implementações em estruturas de teste diferentes são semelhantes, mas não idênticos.

Embora o aplicativo não usa o padrão de repositório e não é um exemplo efetivação do [padrão de unidade de trabalho \(UoW\)](#), páginas do Razor dá suporte a esses padrões de desenvolvimento. Para obter mais informações, consulte [Projetando a camada de persistência de infraestrutura](#) e [lógica do controlador de teste](#) (o exemplo implementa o padrão de repositório).

### Organização de aplicativo de teste

O aplicativo de teste é um aplicativo de console dentro de *tests/RazorPagesProject.Tests* pasta.

PASTA DO APPLICATIVO DE TESTE	DESCRIÇÃO
<i>BasicTests</i>	<i>BasicTests.cs</i> contém métodos de teste para o roteamento, acesso a uma página segura por um usuário não autenticado e como obter um perfil de usuário do GitHub e verificação de logon do usuário do perfil.
<i>IntegrationTests</i>	<i>IndexPageTests.cs</i> contém os testes de integração para a página de índice usar custom <code>WebApplicationFactory</code> classe.

PASTA DO APlicativo DE TESTE	DESCRIÇÃO
<i>Os auxiliares/Utilities</i>	<ul style="list-style-type: none"> <li>• <i>Utilities.cs</i> contém o <code>InitializeDbForTests</code> método usado para propagar o banco de dados com dados de teste.</li> <li>• <i>HtmlHelpers.cs</i> fornece um método para retornar um AngleSharp <code>IHtmlDocument</code> para uso pelos métodos de teste.</li> <li>• <i>HttpClientExtensions.cs</i> fornecem sobrecargas para <code>SendAsync</code> para enviar solicitações para o SUT.</li> </ul>

É a estrutura de teste [xUnit](#). Testes de integração são realizados usando o `testhost`, que inclui o `TestServer`. Porque o [Microsoft.AspNetCore.Mvc.Testing](#) pacote é usado para configurar o servidor de host e o teste de teste, o `TestHost` e `TestServer` pacotes não exigem referências de pacote direto no arquivo de projeto de teste do aplicativo ou configuração de desenvolvedor do aplicativo de teste.

### A propagação do banco de dados para teste

Testes de integração geralmente exigem um pequeno conjunto de dados no banco de dados antes da execução de teste. Por exemplo, uma exclusão chamadas de teste para uma exclusão de registros de banco de dados, portanto, o banco de dados deve ter pelo menos um registro para a solicitação de exclusão seja bem-sucedida.

O aplicativo de exemplo propaga o banco de dados com três mensagens na *Utilities.cs* que testes podem usar quando elas são executadas:

```
public static void InitializeDbForTests(ApplicationDbContext db)
{
    db.Messages.AddRange(GetSeedingMessages());
    db.SaveChanges();
}

public static List<Message> GetSeedingMessages()
{
    return new List<Message>()
    {
        new Message(){ Text = "TEST RECORD: You're standing on my scarf." },
        new Message(){ Text = "TEST RECORD: Would you like a jelly baby?" },
        new Message(){ Text = "TEST RECORD: To the rational mind, " +
            "nothing is inexplicable; only unexplained." }
    };
}
```

## Recursos adicionais

- [Testes de unidade](#)
- [Testes de unidades de páginas Razor](#)
- [Middleware](#)
- [Controladores de teste](#)

# ASP.NET Core teste de estresse e carregar

11/01/2019 • 4 minutes to read • [Edit Online](#)

Teste de carga e testes de estresse são importantes para garantir que um aplicativo web é eficaz e escalonável. Suas metas são diferentes, mesmo que eles compartilham muitas vezes testes semelhantes.

**Testes de carga:** Testa se o aplicativo pode lidar com uma carga especificada de usuários para um determinado cenário e ainda assim satisfazer a meta de resposta. O aplicativo é executado em condições normais.

**Testes de estresse:** Estabilidade do aplicativo de testes ao executar sob condições extremas e muitas vezes um longo período de tempo:

- Carga de usuário com altos – picos ou aumentando gradualmente.
- Recursos de computação limitados.

Sob carga excessiva, pode o aplicativo se recuperar de falha e normalmente retornar ao comportamento esperado? Sob carga excessiva, o aplicativo está *não* executado sob condições normais.

## Ferramentas do Visual Studio

Visual Studio permite aos usuários criar, desenvolver e depurar testes de carga e desempenho na web. Uma opção está disponível para criar testes gravando ações em um navegador da web.

[Início Rápido: Criar um projeto de teste de carga](#) mostra como criar, configurar e executar um teste de carga projetos usando o Visual Studio 2017.

Consulte [Recursos adicionais](#) para obter mais informações.

Testes de carga podem ser configurados para executar no local ou executados na nuvem usando o DevOps do Azure.

## DevOps do Azure

Execuções de teste de carga podem ser iniciadas usando o [planos de teste do Azure DevOps](#) service.

The screenshot shows the Azure DevOps interface for 'CustomerSamples'. The left sidebar has 'Test Plans' selected under 'Load test'. A modal window is open in the center, titled 'Create and run high-scale load tests, analyze results – all using the browser! Learn more. During preview, this feature...' with a 'New' button. Below it are four options: 'Visual Studio test', 'HTTP Archive based test\*', 'URL based test', and 'Apache JMeter test'. To the right of the modal is a list of 'All load test runs' with entries like 'LoadTest1.loadtest', 'PrimacyLoadTests', 'Rate Service Performance Test Update...', and 'TestAPI.loadtest'. At the bottom right of the modal is a 'Run Type' section with 'Run I...' and 'Load Test' buttons.

O serviço suporta os seguintes tipos de formato de teste:

- Teste do Visual Studio – teste da web criado no Visual Studio.
- Teste com base em arquivo HTTP – tráfego HTTP capturado dentro do arquivo morto é reproduzida durante o teste.
- **Teste com base na URL** – permite especificar URLs para carregar testes, tipos de solicitação, cabeçalhos e cadeias de caracteres de consulta. Executar a configuração de parâmetros, como duração, padrão de carga, o número de usuários, etc., pode ser configurado.
- **Apache JMeter** de teste.

## Portal do Azure

[Portal do Azure](#) permite configurar e executar testes de carga de aplicativos Web, diretamente a partir da guia de desempenho do serviço de aplicativo no portal do Azure.

The screenshot shows the Azure Portal interface. The left sidebar has 'Performance test' selected under 'Development Tools'. The main area shows a 'Recent runs' section with a table header 'NAME' and a message 'No performance test found. C'. There are also other development tools listed: 'Change App Service plan', 'Clone App', 'Console', 'Advanced Tools', 'App Service Editor (Preview)', 'Resource explorer', 'Testing in production', and '...'. At the top, there's a search bar, a 'New' button, and a 'Set Organization' button.

O teste pode ser um teste manual com uma URL especificada, ou um arquivo de teste do Visual Studio Web, que pode testar várias URLs.

The screenshot shows the 'Configure test using' dialog for a new performance test. On the left, under 'CONFIGURE TEST USING', it says 'Test type: ManualTest 1 Url'. On the right, under 'TEST TYPE', 'Manual Test' is selected. Below that, the 'URL' field contains 'http://studymodulechooser.azurewebsites.net'. Other fields include 'NAME' (PerfTest01), 'GENERATE LOAD FROM' (Central US (Web app Location)), and 'USER LOAD' (250).

No final do teste, os relatórios são gerados para mostrar as características de desempenho do aplicativo.

Estatísticas de exemplo incluem:

- Tempo médio de resposta
- Taxa de transferência máxima: solicitações por segundo
- Percentual de falha

## Ferramentas de terceiros

A lista a seguir contém as ferramentas de desempenho da web de terceiros com vários conjuntos de recursos:

- [Apache JMeter](#) : Pacote de destaque completo de ferramentas de teste de carga. Limite de thread: precisa de um thread por usuário.
- [AB - servidor HTTP Apache ferramenta de benchmark](#)
- [Gatling](#) : Ferramenta da área de trabalho com gravadores de GUI e de teste. Mais eficaz do que o JMeter.
- [Locust.IO](#) : Não é limitado por threads.

## Recursos adicionais

[Série de blogs de teste de carga](#) por Charles Sterling. Com data, mas a maioria dos tópicos ainda é relevante.

# Solucionar problemas de projetos do ASP.NET Core

28/11/2018 • 6 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

Os links a seguir fornecem diretrizes de solução de problemas:

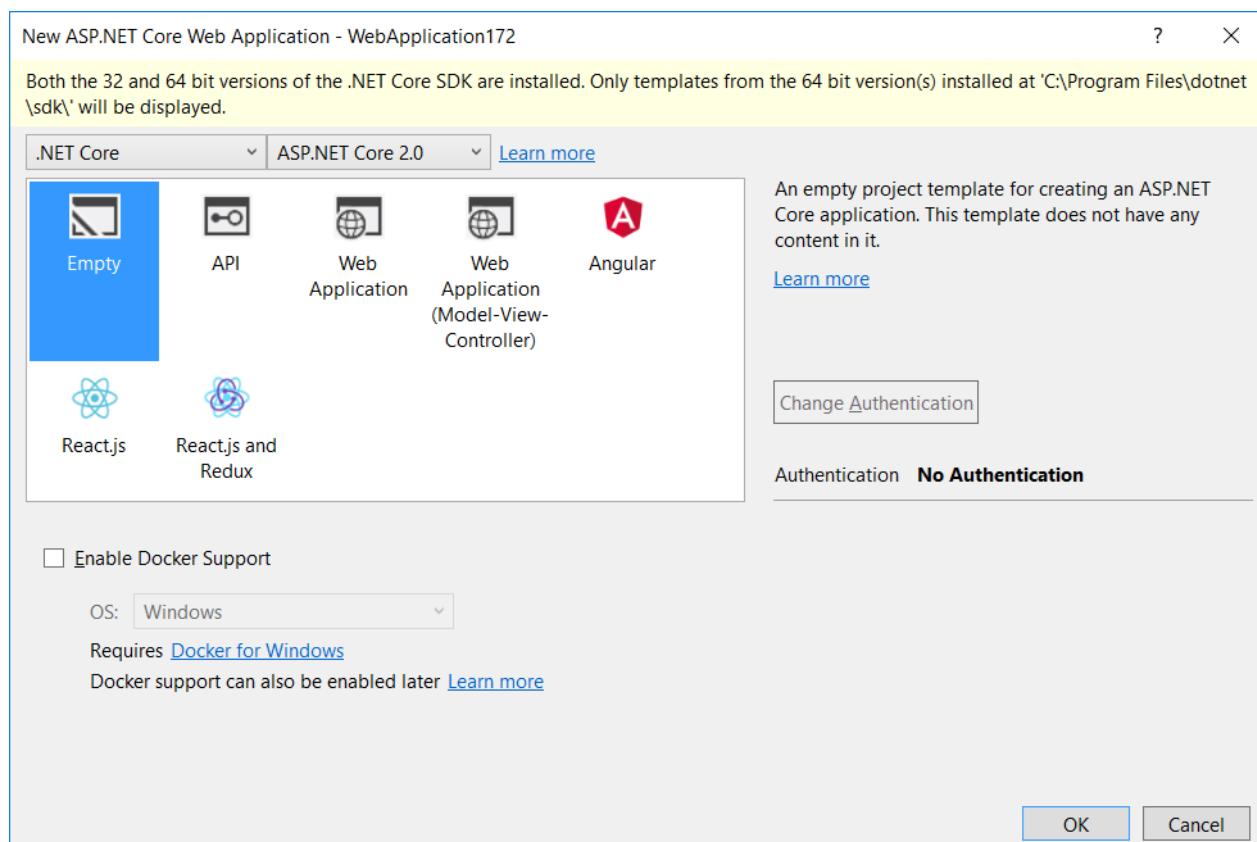
- [Solucionar problemas no ASP.NET Core no Serviço de Aplicativo do Azure](#)
- [Solucionar problemas do ASP.NET Core no IIS](#)
- [Referência de erros comuns para o Serviço de Aplicativo do Azure e o IIS com o ASP.NET Core](#)
- [NDC Conference \(2018 Londres\): Diagnosticar problemas em aplicativos do ASP.NET Core](#)
- [Blog do ASP.NET: Solucionando problemas de desempenho do ASP.NET Core](#)

## Avisos do SDK do .NET core

### Os 32 bits e versões de 64 bits do SDK do .NET Core estão instaladas

No **novo projeto** caixa de diálogo para o ASP.NET Core, você poderá ver o seguinte aviso:

As versões de bit 32 e 64 do SDK do .NET Core são instaladas. Somente modelos das versões de 64 bits instalados em 'c:\Program Files\dotnet\sdk\' serão exibida.



Esse aviso é exibido quando (x86) 32 bits e 64 bits (x64) versões dos [SDK do .NET Core](#) estão instalados. Ambas as versões podem ser instaladas os motivos comuns incluem:

- Você originalmente baixou o instalador do SDK do .NET Core usando um computador de 32 bits, mas, em seguida, copiado entre e instalado em um computador de 64 bits.
- O .NET Core SDK de 32 bits foi instalado por outro aplicativo.

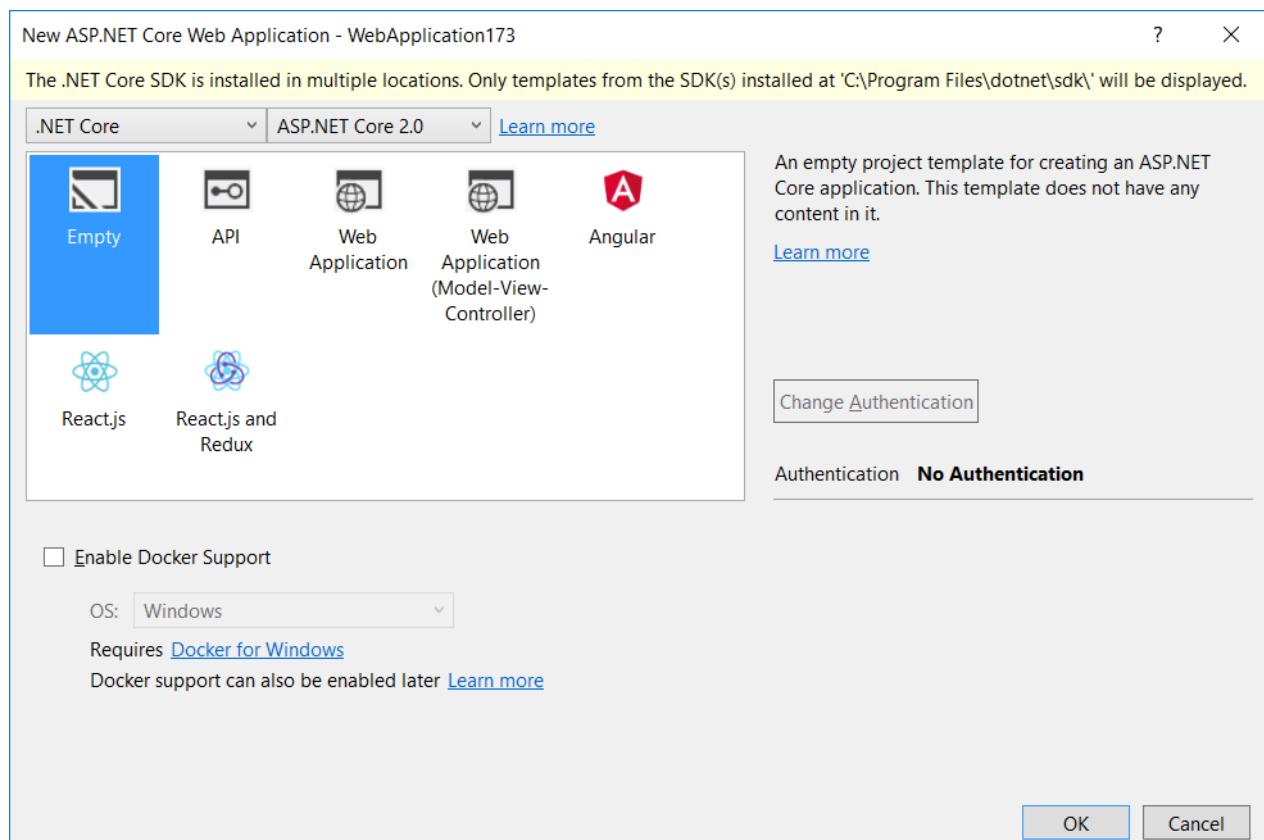
- A versão incorreta foi baixada e instalada.

Desinstale o .NET Core SDK 32 bits para evitar esse aviso. Desinstalar do **painel de controle > programas e recursos > desinstalar ou alterar um programa**. Se você entender por que o aviso ocorre e suas implicações, você pode ignorar o aviso.

### SDK do .NET Core é instalado em vários locais

No **novo projeto** caixa de diálogo para o ASP.NET Core, você poderá ver o seguinte aviso:

SDK do .NET Core é instalado em vários locais. Somente modelos dos SDKs instalados em 'c:\Program Files\dotnet\sdk\' será exibida.



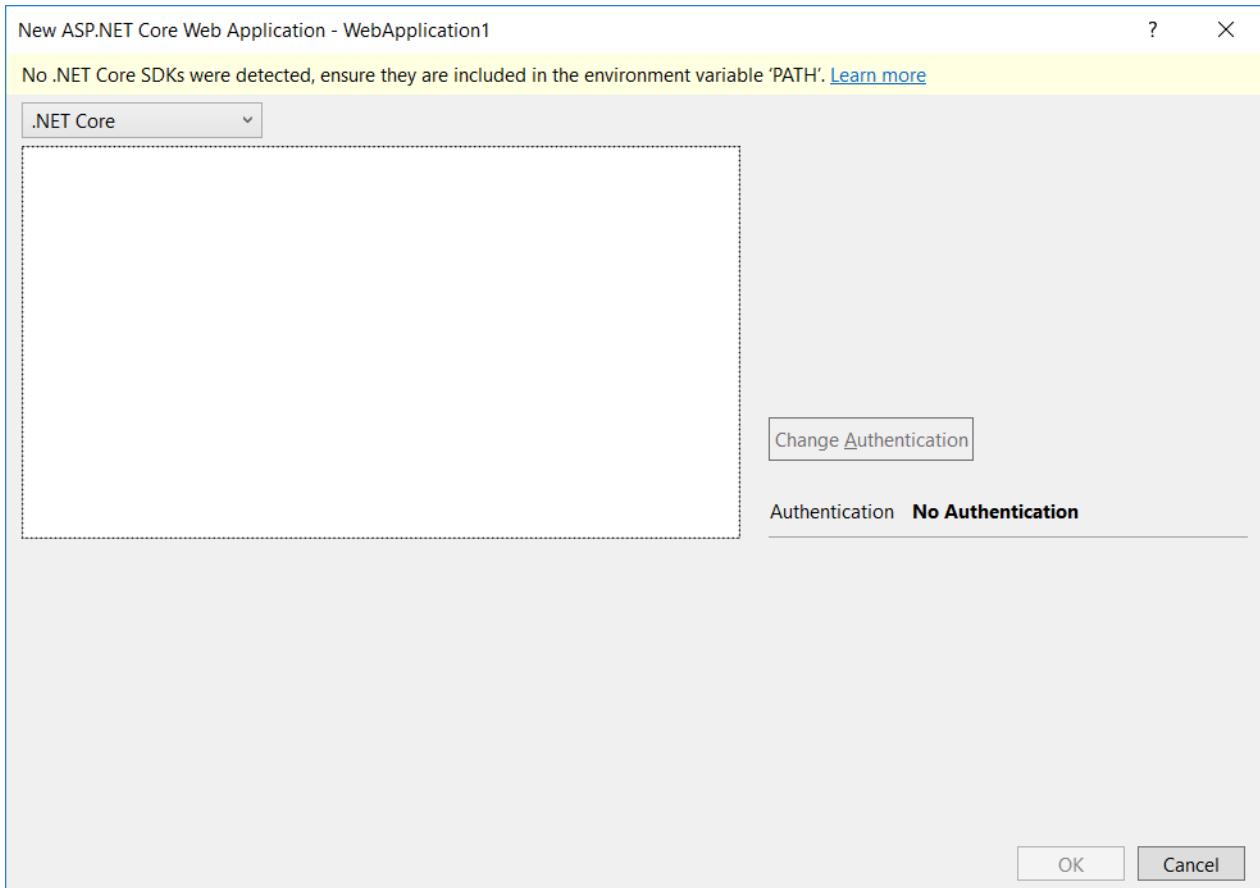
Você verá esta mensagem quando você tem pelo menos uma instalação do SDK do .NET Core em um diretório fora do *c:\arquivos de programas\dotnet\sdk*. Geralmente isso acontece quando o SDK do .NET Core foi implantado em um computador usando copiar/colar em vez do instalador MSI.

Desinstale o .NET Core SDK 32 bits para evitar esse aviso. Desinstalar do **painel de controle > programas e recursos > desinstalar ou alterar um programa**. Se você entender por que o aviso ocorre e suas implicações, você pode ignorar o aviso.

### Não há SDKs do .NET Core foram detectados

No **novo projeto** caixa de diálogo para o ASP.NET Core, você poderá ver o seguinte aviso:

Nenhum SDK do .NET Core foi detectado, verifique se que eles estão incluídos na variável de ambiente 'PATH'.



Esse aviso é exibido quando a variável de ambiente `PATH` não apontar para qualquer SDKs do .NET Core no computador. Para resolver esse problema:

- Instale ou verifique se o que SDK do .NET Core é instalado.
- Verifique o `PATH` variável de ambiente aponta para o local em que o SDK está instalado. O instalador normalmente define o `PATH`.

## Obter dados de um aplicativo

Se um aplicativo é capaz de responder a solicitações, você pode obter os seguintes dados do aplicativo usando o middleware:

- Solicitar – cadeia, cabeçalhos de consulta de método, esquema, host, pathbase, caminho,
- Conexão – endereço IP remoto, porta remota, endereço IP local, porta local, o certificado de cliente
- Identidade – nome, nome de exibição
- Definições de configuração
- Variáveis de ambiente

Coloque o seguinte middleware código no início do `Startup.Configure` pipeline de processamento de solicitação do método. O ambiente é verificado antes do middleware é executado para garantir que o código só é executado no ambiente de desenvolvimento.

Para obter o ambiente, use qualquer uma das seguintes abordagens:

- Injetar o `IHostingEnvironment` para o `Startup.Configure` método e verifique se o ambiente com a variável local. O código de exemplo a seguir demonstra essa abordagem.
- Atribuir o ambiente para uma propriedade no `Startup` classe. Verificar o ambiente usando a propriedade (por exemplo, `if (Environment.IsDevelopment())`).

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
```

```

IConfiguration config)
{
    if (env.IsDevelopment())
    {
        app.Run(async (context) =>
        {
            var sb = new StringBuilder();
            var nl = System.Environment.NewLine;
            var rule = string.Concat(nl, new string('-', 40), nl);
            var authSchemeProvider = app.ApplicationServices
                .GetRequiredService<IAuthenticationSchemeProvider>();

            sb.Append($"Request{rule}");
            sb.Append($"{DateTimeOffset.Now}{nl}");
            sb.Append($"{context.Request.Method} {context.Request.Path}{nl}");
            sb.Append($"Scheme: {context.Request.Scheme}{nl}");
            sb.Append($"Host: {context.Request.Headers["Host"]}{nl}");
            sb.Append($"PathBase: {context.Request.PathBase.Value}{nl}");
            sb.Append($"Path: {context.Request.Path.Value}{nl}");
            sb.Append($"Query: {context.Request.QueryString.Value}{nl}{nl}");

            sb.Append($"Connection{rule}");
            sb.Append($"RemoteIp: {context.Connection.RemoteIpAddress}{nl}");
            sb.Append($"RemotePort: {context.Connection.RemotePort}{nl}");
            sb.Append($"LocalIp: {context.Connection.LocalIpAddress}{nl}");
            sb.Append($"LocalPort: {context.Connection.LocalPort}{nl}");
            sb.Append($"ClientCert: {context.Connection.ClientCertificate}{nl}{nl}");

            sb.Append($"Identity{rule}");
            sb.Append($"User: {context.User.Identity.Name}{nl}");
            var scheme = await authSchemeProvider
                .GetSchemeAsync(IISDefaults.AuthenticationScheme);
            sb.Append($"DisplayName: {scheme?.DisplayName}{nl}{nl}");

            sb.Append($"Headers{rule}");
            foreach (var header in context.Request.Headers)
            {
                sb.Append($"{header.Key}: {header.Value}{nl}");
            }
            sb.Append(nl);

            sb.Append($"Websockets{rule}");
            if (context.Features.Get< IHttpUpgradeFeature>() != null)
            {
                sb.Append($"Status: Enabled{nl}{nl}");
            }
            else
            {
                sb.Append($"Status: Disabled{nl}{nl}");
            }

            sb.Append($"Configuration{rule}");
            foreach (var pair in config.AsEnumerable())
            {
                sb.Append($"{pair.Path}: {pair.Value}{nl}");
            }
            sb.Append(nl);

            sb.Append($"Environment Variables{rule}");
            var vars = System.Environment.GetEnvironmentVariables();
            foreach (var key in vars.Keys.Cast<string>().OrderBy(key => key,
                StringComparer.OrdinalIgnoreCase))
            {
                var value = vars[key];
                sb.Append($"{key}: {value}{nl}");
            }

            context.Response.ContentType = "text/plain";
            await context.Response.WriteAsync(sb.ToString());
        });
    }
}

```

```
    } );  
}  
}
```

# Registro em log no ASP.NET Core

04/02/2019 • 47 minutes to read • [Edit Online](#)

Por [Steve Smith](#) e [Tom Dykstra](#)

O ASP.NET Core oferece suporte a uma API de registro em log que funciona com uma variedade de provedores de logs internos e terceirizados. Este artigo mostra como usar a API de registro em log com provedores internos.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Adicionar provedores

Um provedor de log exibe ou armazena logs. Por exemplo, o provedor de Console exibe os logs no console, e o provedor do Azure Application Insights armazena-os no Azure Application Insights. Os logs podem ser enviados para vários destinos por meio da adição de vários provedores.

Para adicionar um provedor, chame o método de extensão `Add{provider name}` do provedor no *Program.cs*:

```
public static void Main(string[] args)
{
    var webHost = new WebHostBuilder()
        .UseKestrel()
        .UseContentRoot(Directory.GetCurrentDirectory())
        .ConfigureAppConfiguration((hostingContext, config) =>
    {
        var env = hostingContext.HostingEnvironment;
        config.AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
            .AddJsonFile($"appsettings.{env.EnvironmentName}.json",
                optional: true, reloadOnChange: true);
        config.AddEnvironmentVariables();
    })
    .ConfigureLogging((hostingContext, logging) =>
    {
        logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));
        logging.AddConsole();
        logging.AddDebug();
        logging.AddEventSourceLogger();
    })
    .UseStartup<Startup>()
    .Build();

    webHost.Run();
}
```

O modelo de projeto padrão chama o método de extensão `CreateDefaultBuilder`, que adiciona os seguintes provedores de log:

- Console
- Depurar
- EventSource (a partir do ASP.NET Core 2.2)

```

public static void Main(string[] args)
{
    BuildWebHost(args).Run();
}

public static IWebHost BuildWebHost(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .Build();

```

Se você usar `CreateDefaultBuilder`, poderá substituir os provedores padrão por aqueles que preferir. Chame `ClearProviders` e adicione os provedores desejados.

```

public static void Main(string[] args)
{
    var host = BuildWebHost(args);

    var todoRepository = host.Services.GetRequiredService<ITodoRepository>();
    todoRepository.Add(new Core.Model.TodoItem() { Name = "Feed the dog" });
    todoRepository.Add(new Core.Model.TodoItem() { Name = "Walk the dog" });

    var logger = host.Services.GetRequiredService<ILogger<Program>>();
    logger.LogInformation("Seeded the database.");

    host.Run();
}

public static IWebHost BuildWebHost(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .ConfigureLogging(logging =>
    {
        logging.ClearProviders();
        logging.AddConsole();
    })
        .Build();

```

Para usar um provedor, instale o pacote NuGet e chame o método de extensão do provedor em uma instância de `ILoggerFactory`:

```

public void Configure(IApplicationBuilder app,
    IHostingEnvironment env,
    ILoggerFactory loggerFactory)
{
    loggerFactory
        .AddConsole()
        .AddDebug();
}

```

A [DI \(injeção de dependência\)](#) do ASP.NET Core fornece a instância de `ILoggerFactory`. Os métodos de extensão `AddConsole` e `AddDebug` são definidos nos pacotes [Microsoft.Extensions.Logging.Console](#) e [Microsoft.Extensions.Logging.Debug](#). Cada método de extensão chama o método `ILoggerFactory.AddProvider`, passando uma instância do provedor.

#### NOTE

O [aplicativo de exemplo](#) adiciona provedores de log no método `startup.Configure`. Para obter saída de log do código que é executado anteriormente, adicione provedores de log no construtor da classe `Startup`.

Saiba mais sobre [provedores de log internos](#) e [provedores de log de terceiros](#) mais adiante no artigo.

## Criar logs

Obtenha um objeto `ILogger<TCategoryName>` da DI.

O exemplo de controlador a seguir cria os logs `Information` e `Warning`. A *categoria* é `TodoApiController` (o nome de classe totalmente qualificado do `TodoController` no aplicativo de exemplo):

```
public class TodoController : Controller
{
    private readonly ITodoRepository _todoRepository;
    private readonly ILogger _logger;

    public TodoController(ITodoRepository todoRepository,
        ILogger<TodoController> logger)
    {
        _todoRepository = todoRepository;
        _logger = logger;
    }
}
```

```
public IActionResult GetById(string id)
{
    _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
    var item = _todoRepository.Find(id);
    if (item == null)
    {
        _logger.LogWarning(LoggingEvents.GetItemNotFound, "GetById({ID}) NOT FOUND", id);
        return NotFound();
    }
    return new ObjectResult(item);
}
```

O exemplo do Razor Pages a seguir cria logs com `Information` como o *nível* e `TodoApiSample.Pages.AboutModel` como a *categoria*:

```
public class AboutModel : PageModel
{
    private readonly ILogger _logger;

    public AboutModel(ILogger<AboutModel> logger)
    {
        _logger = logger;
    }
}
```

```
public void OnGet()
{
    Message = $"About page visited at {DateTime.UtcNow.ToString("O")}";
    _logger.LogInformation("Message displayed: {Message}", Message);
}
```

```
public class TodoController : Controller
{
    private readonly ITodoRepository _todoRepository;
    private readonly ILogger _logger;

    public TodoController(ITodoRepository todoRepository,
        ILogger<TodoController> logger)
    {
        _todoRepository = todoRepository;
        _logger = logger;
    }
}
```

```
public IActionResult GetById(string id)
{
    _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
    var item = _todoRepository.Find(id);
    if (item == null)
    {
        _logger.LogWarning(LoggingEvents.GetItemNotFound, "GetById({ID}) NOT FOUND", id);
        return NotFound();
    }
    return new ObjectResult(item);
}
```

O exemplo acima cria logs com `Information` e `Warning` como o *nível* e a classe `TodoController` como a *categoria*.

O *nível* de log indica a gravidade do evento registrado. A *categoria* do log é uma cadeia de caracteres associada a cada log. A instância `ILogger<T>` cria logs que têm o nome totalmente qualificado do tipo `T` como a categoria. [Níveis](#) e [categorias](#) serão explicados com mais detalhes posteriormente neste artigo.

### Criar logs na inicialização

Para gravar logs na classe `Startup`, inclua um parâmetro `ILogger` na assinatura de construtor:

```
public class Startup
{
    private readonly ILogger _logger;

    public Startup(IConfiguration configuration, ILogger<Startup> logger)
    {
        Configuration = configuration;
        _logger = logger;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc();

        // Add our repository type
        services.AddSingleton<ITodoRepository, TodoRepository>();
        _logger.LogInformation("Added TodoRepository to services");
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        if (env.IsDevelopment())
        {
            _logger.LogInformation("In Development environment");
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Error");
            app.UseHsts();
        }

        app.UseHttpsRedirection();
        app.UseStaticFiles();
        app.UseCookiePolicy();

        app.UseMvc();
    }
}
```

## Criar logs no programa

Para gravar logs na classe `Program`, obtenha uma instância `ILogger` da DI:

```

public static void Main(string[] args)
{
    var host = BuildWebHost(args);

    var todoRepository = host.Services.GetRequiredService<ITodoRepository>();
    todoRepository.Add(new Core.Model.TodoItem() { Name = "Feed the dog" });
    todoRepository.Add(new Core.Model.TodoItem() { Name = "Walk the dog" });

    var logger = host.Services.GetRequiredService<ILogger<Program>>();
    logger.LogInformation("Seeded the database.");

    host.Run();
}

public static IWebHost BuildWebHost(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .ConfigureLogging(logging =>
    {
        logging.ClearProviders();
        logging.AddConsole();
    })
    .Build();

```

### Sem métodos de agente assíncronos

O registro em log deve ser tão rápido que não justifique o custo de desempenho de código assíncrono. Se o armazenamento de dados em log estiver lento, não grave diretamente nele. Grave as mensagens de log em um repositório rápido primeiro e, depois, mova-as para um repositório lento. Por exemplo, registre em uma fila de mensagens que seja lida e persistida em um armazenamento lento por outro processo.

## Configuração

A configuração do provedor de logs é fornecida por um ou mais provedores de sincronização:

- Formatos de arquivo (INI, JSON e XML).
- Argumentos de linha de comando.
- Variáveis de ambiente.
- Objetos do .NET na memória.
- O armazenamento do [Secret Manager](#) não criptografado.
- Um repositório de usuário criptografado, como o [Azure Key Vault](#).
- Provedores personalizados (instalados ou criados).

Por exemplo, a configuração de log geralmente é fornecida pela seção `Logging` dos arquivos de configurações do aplicativo. O exemplo a seguir mostra o conteúdo de um típico arquivo `appsettings.Development.json`:

```
{  
  "Logging": {  
    "LogLevel": {  
      "Default": "Debug",  
      "System": "Information",  
      "Microsoft": "Information"  
    },  
    "Console":  
    {  
      "IncludeScopes": true  
    }  
  }  
}
```

A propriedade `Logging` pode ter `LogLevel` e propriedades do provedor de logs (o Console é mostrado).

A propriedade `LogLevel` em `Logging` especifica o **nível** mínimo para log nas categorias selecionadas. No exemplo, as categorias `System` e `Microsoft` têm log no nível `Information`, e todas as outras no nível `Debug`.

Outras propriedades em `Logging` especificam provedores de logs. O exemplo se refere ao provedor de Console. Se um provedor oferecer suporte a `escopos de log`, `IncludeScopes` indicará se eles estão habilitados. Uma propriedade de provedor (como `Console`, no exemplo) também pode especificar uma propriedade `LogLevel`. `LogLevel` em um provedor especifica os níveis de log para esse provedor.

Se os níveis forem especificados em `Logging.{providername}.LogLevel`, eles substituirão o que estiver definido em `Logging.LogLevel`.

```
{  
  "Logging": {  
    "LogLevel": {  
      "Default": "Debug",  
      "System": "Information",  
      "Microsoft": "Information"  
    }  
  }  
}
```

Chaves `LogLevel` representam nomes de log. A chave `Default` aplica-se a logs não listados de forma explícita. O valor representa o **nível de log** aplicado ao log fornecido.

Saiba mais sobre como implementar provedores de configuração em [Configuração no ASP.NET Core](#).

## Exemplo de saída de registro em log

Com o código de exemplo mostrado na seção anterior, os logs serão exibidos no console quando o aplicativo for executado pela linha de comando. Aqui está um exemplo da saída do console:

```
info: Microsoft.AspNetCore.Hosting.Internal.WebHost[1]
      Request starting HTTP/1.1 GET http://localhost:5000/api/todo/0
info: Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker[1]
      Executing action method TodoApi.Controllers.TodoController.GetById (TodoApi) with arguments (0) - 
ModelState is Valid
info: TodoApi.Controllers.TodoController[1002]
      Getting item 0
warn: TodoApi.Controllers.TodoController[4000]
      GetById(0) NOT FOUND
info: Microsoft.AspNetCore.Mvc.StatusCodeResult[1]
      Executing HttpStatusCodeResult, setting HTTP status code 404
info: Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker[2]
      Executed action TodoApi.Controllers.TodoController.GetById (TodoApi) in 42.9286ms
info: Microsoft.AspNetCore.Hosting.Internal.WebHost[2]
      Request finished in 148.889ms 404
```

Os logs anteriores foram gerados por meio de uma solicitação HTTP Get para o aplicativo de exemplo em <http://localhost:5000/api/todo/0>.

Veja um exemplo de como os mesmos logs aparecem na janela Depuração quando você executa o aplicativo de exemplo no Visual Studio:

```
Microsoft.AspNetCore.Hosting.Internal.WebHost:Information: Request starting HTTP/1.1 GET
http://localhost:53104/api/todo/0
Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker:Information: Executing action method
TodoApi.Controllers.TodoController.GetById (TodoApi) with arguments (0) - ModelState is Valid
TodoApi.Controllers.TodoController:Information: Getting item 0
TodoApi.Controllers.TodoController:Warning: GetById(0) NOT FOUND
Microsoft.AspNetCore.Mvc.StatusCodeResult:Information: Executing HttpStatusCodeResult, setting HTTP
status code 404
Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker:Information: Executed action
TodoApi.Controllers.TodoController.GetById (TodoApi) in 152.5657ms
Microsoft.AspNetCore.Hosting.Internal.WebHost:Information: Request finished in 316.3195ms 404
```

Os logs criados pelas chamadas `ILogger` mostradas na seção anterior começam com "TodoApi.Controllers.TodoController". Os logs que começam com categorias "Microsoft" são de código da estrutura ASP.NET Core. O ASP.NET Core e o código do aplicativo estão usando a mesma API de registro em log e os mesmos provedores.

O restante deste artigo explica alguns detalhes e opções para registro em log.

## Pacotes NuGet

As interfaces `ILogger` e `ILoggerFactory` estão em [Microsoft.Extensions.Logging.Abstractions](#) e as implementações padrão para elas estão em [Microsoft.Extensions.Logging](#).

## Categoria de log

Quando um objeto `ILogger` é criado, uma *categoria* é especificada para ele. Essa categoria é incluída em cada mensagem de log criada por essa instância de `ILogger`. A categoria pode ser qualquer cadeia de caracteres, mas a convenção é usar o nome da classe, como "TodoApi.Controllers.TodoController".

Use `ILogger<T>` para obter uma instância `ILogger` que usa o nome de tipo totalmente qualificado do `T` como a categoria:

```
public class TodoController : Controller
{
    private readonly ITodoRepository _todoRepository;
    private readonly ILogger _logger;

    public TodoController(ITodoRepository todoRepository,
        ILogger<TodoController> logger)
    {
        _todoRepository = todoRepository;
        _logger = logger;
    }
}
```

```
public class TodoController : Controller
{
    private readonly ITodoRepository _todoRepository;
    private readonly ILogger _logger;

    public TodoController(ITodoRepository todoRepository,
        ILogger<TodoController> logger)
    {
        _todoRepository = todoRepository;
        _logger = logger;
    }
}
```

Para especificar explicitamente a categoria, chame `ILoggerFactory.CreateLogger`:

```
public class TodoController : Controller
{
    private readonly ITodoRepository _todoRepository;
    private readonly ILogger _logger;

    public TodoController(ITodoRepository todoRepository,
        ILoggerFactory logger)
    {
        _todoRepository = todoRepository;
        _logger = logger.CreateLogger("TodoApiSample.Controllers.TodoController");
    }
}
```

```
public class TodoController : Controller
{
    private readonly ITodoRepository _todoRepository;
    private readonly ILogger _logger;

    public TodoController(ITodoRepository todoRepository,
        ILoggerFactory logger)
    {
        _todoRepository = todoRepository;
        _logger = logger.CreateLogger("TodoApiSample.Controllers.TodoController");
    }
}
```

`ILogger<T>` é equivalente a chamar `CreateLogger` com o nome de tipo totalmente qualificado de `T`.

## Nível de log

Todo log especifica um valor [LogLevel](#). O nível de log indica a gravidade ou importância. Por exemplo, você pode gravar um log `Information` quando um método é finalizado normalmente e um log `Warning` quando um método retorna um código de status `404 Não Encontrado`.

O código a seguir cria os logs `Information` e `Warning`:

```

public IActionResult GetById(string id)
{
    _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
    var item = _todoRepository.Find(id);
    if (item == null)
    {
        _logger.LogWarning(LoggingEvents.GetItemNotFound, "GetById({ID}) NOT FOUND", id);
        return NotFound();
    }
    return new ObjectResult(item);
}

```

```

public IActionResult GetById(string id)
{
    _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
    var item = _todoRepository.Find(id);
    if (item == null)
    {
        _logger.LogWarning(LoggingEvents.GetItemNotFound, "GetById({ID}) NOT FOUND", id);
        return NotFound();
    }
    return new ObjectResult(item);
}

```

No código anterior, o primeiro parâmetro é a [ID de evento de log](#). O segundo parâmetro é um modelo de mensagem com espaços reservados para valores de argumento fornecidos pelos parâmetros de método restantes. Os parâmetros de método serão explicados com posteriormente neste artigo, na [seção de modelos de mensagem](#).

Os métodos de log que incluem o nível no nome do método (por exemplo, `.LogInformation` e `.LogWarning`) são [métodos de extensão para ILogger](#). Esses métodos chamam um método `Log` que recebe um parâmetro `LogLevel`. Você pode chamar o método `Log` diretamente em vez de um desses métodos de extensão, mas a sintaxe é relativamente complicada. Para saber mais, veja [ILogger](#) e o [código-fonte de extensões de agente](#).

O ASP.NET Core define os seguintes níveis de log, ordenados aqui da menor para a maior gravidade.

- Trace = 0

Para obter informações que normalmente são valiosas somente para depuração. Essas mensagens podem conter dados confidenciais de aplicativos e, portanto, não devem ser habilitadas em um ambiente de produção. *Desabilitado por padrão*.

- Debug = 1

Para obter informações que possam ser úteis durante o desenvolvimento e a depuração. Exemplo:  
`Entering method Configure with flag set to true.` habilite logs de nível `Debug` em produção somente ao solucionar problemas, devido ao alto volume de logs.

- Information = 2

Para rastrear o fluxo geral do aplicativo. Esses logs normalmente têm algum valor a longo prazo.  
Exemplo: `Request received for path /api/todo`

- Warning = 3

Para eventos anormais ou inesperados no fluxo de aplicativo. Eles podem incluir erros ou outras condições que não fazem com que o aplicativo pare, mas que talvez precisem ser investigados. Exceções manipuladas são um local comum para usar o nível de log `Warning`. Exemplo:

```
FileNotFoundException for file quotes.txt.
```

- Error = 4

Para erros e exceções que não podem ser manipulados. Essas mensagens indicam uma falha na atividade ou na operação atual (como a solicitação HTTP atual) e não uma falha em todo o aplicativo. Mensagem de log de exemplo: `Cannot insert record due to duplicate key violation.`

- Critical = 5

Para falhas que exigem atenção imediata. Exemplos: cenários de perda de dados, espaço em disco insuficiente.

Use o nível de log para controlar a quantidade de saída de log que é gravada em uma mídia de armazenamento específica ou em uma janela de exibição. Por exemplo:

- Em produção, envie `Trace` por meio do nível `Information` para um armazenamento de dados com volume. Envie `Warning` por meio de `Critical` para um armazenamento de dados com valor.
- Durante o desenvolvimento, envie `Warning` por meio `Critical` para o console e adicione `Trace` por meio de `Information` ao solucionar problemas.

A seção [Filtragem de log](#) mais adiante neste artigo explicará como controlar os níveis de log que um provedor manipula.

O ASP.NET Core grava logs para eventos de estrutura. Os exemplos de log anteriores neste artigo excluíram logs abaixo do nível `Information`, portanto, logs de nível `Debug` ou `Trace` não foram criados. Veja um exemplo de logs de console produzidos por meio da execução do aplicativo de exemplo configurado para mostrar logs `Debug`:

```
info: Microsoft.AspNetCore.Hosting.Internal.WebHost[1]
      Request starting HTTP/1.1 GET http://localhost:62555/api/todo/0
dbug: Microsoft.AspNetCore.Routing.Tree.TreeRouter[1]
      Request successfully matched the route with name 'GetTodo' and template 'api/Todo/{id}'.
dbug: Microsoft.AspNetCore.Mvc.Internal.ActionSelector[2]
      Action 'TodoApi.Controllers.TodoController.Update (TodoApi)' with id '089d59b6-92ec-472d-b552-cc613df625d' did not match the constraint
'Microsoft.AspNetCore.Mvc.Internal.HttpMethodActionConstraint'
dbug: Microsoft.AspNetCore.Mvc.Internal.ActionSelector[2]
      Action 'TodoApi.Controllers.TodoController.Delete (TodoApi)' with id 'f3476abe-4bd9-4ad3-9261-3ead09607366' did not match the constraint
'Microsoft.AspNetCore.Mvc.Internal.HttpMethodActionConstraint'
dbug: Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker[1]
      Executing action TodoApi.Controllers.TodoController.GetById (TodoApi)
info: Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker[1]
      Executing action method TodoApi.Controllers.TodoController.GetById (TodoApi) with arguments (0) - ModelState is Valid
info: TodoApi.Controllers.TodoController[1002]
      Getting item 0
warn: TodoApi.Controllers.TodoController[4000]
      GetById(0) NOT FOUND
dbug: Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker[2]
      Executed action method TodoApi.Controllers.TodoController.GetById (TodoApi), returned result Microsoft.AspNetCore.Mvc.NotFoundResult.
info: Microsoft.AspNetCore.Mvc.StatusCodeResult[1]
      Executing HttpStatusCodeResult, setting HTTP status code 404
info: Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker[2]
      Executed action TodoApi.Controllers.TodoController.GetById (TodoApi) in 0.8788ms
dbug: Microsoft.AspNetCore.Server.Kestrel[9]
      Connection id "0HL6L7NEFF2QD" completed keep alive response.
info: Microsoft.AspNetCore.Hosting.Internal.WebHost[2]
      Request finished in 2.7286ms 404
```

## ID de evento de log

Cada log pode especificar uma *ID do evento*. O aplicativo de exemplo faz isso usando uma classe `LoggingEvents` definida localmente:

```
public IActionResult GetById(string id)
{
    _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
    var item = _todoRepository.Find(id);
    if (item == null)
    {
        _logger.LogWarning(LoggingEvents.GetItemNotFound, ".GetById({ID}) NOT FOUND", id);
        return NotFound();
    }
    return new ObjectResult(item);
}
```

```
public class LoggingEvents
{
    public const int GenerateItems = 1000;
    public const int ListItems = 1001;
    public const int GetItem = 1002;
    public const int InsertItem = 1003;
    public const int UpdateItem = 1004;
    public const int DeleteItem = 1005;

    public const int GetItemNotFound = 4000;
    public const int UpdateItemNotFound = 4001;
}
```

```
public IActionResult GetById(string id)
{
    _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
    var item = _todoRepository.Find(id);
    if (item == null)
    {
        _logger.LogWarning(LoggingEvents.GetItemNotFound, ".GetById({ID}) NOT FOUND", id);
        return NotFound();
    }
    return new ObjectResult(item);
}
```

```
public class LoggingEvents
{
    public const int GenerateItems = 1000;
    public const int ListItems = 1001;
    public const int GetItem = 1002;
    public const int InsertItem = 1003;
    public const int UpdateItem = 1004;
    public const int DeleteItem = 1005;

    public const int GetItemNotFound = 4000;
    public const int UpdateItemNotFound = 4001;
}
```

Uma ID de evento associa um conjunto de eventos. Por exemplo, todos os logs relacionados à exibição de uma lista de itens em uma página podem ser 1001.

O provedor de logs pode armazenar a ID do evento em um campo de ID na mensagem de log ou não armazenar. O provedor de Depuração não mostra IDs de eventos. O provedor de console mostra IDs de

evento entre colchetes após a categoria:

```
info: TodoApi.Controllers.TodoController[1002]
      Getting item invalidid
warn: TodoApi.Controllers.TodoController[4000]
      GetById(invalidid) NOT FOUND
```

## Modelo de mensagem de log

Cada log especifica um modelo de mensagem. O modelo de mensagem pode conter espaços reservados para os quais são fornecidos argumentos. Use nomes para os espaços reservados, não números.

```
public IActionResult GetById(string id)
{
    _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
    var item = _todoRepository.Find(id);
    if (item == null)
    {
        _logger.LogWarning(LoggingEvents.GetItemNotFound, "GetById({ID}) NOT FOUND", id);
        return NotFound();
    }
    return new ObjectResult(item);
}
```

```
public IActionResult GetById(string id)
{
    _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
    var item = _todoRepository.Find(id);
    if (item == null)
    {
        _logger.LogWarning(LoggingEvents.GetItemNotFound, "GetById({ID}) NOT FOUND", id);
        return NotFound();
    }
    return new ObjectResult(item);
}
```

A ordem dos espaços reservados e não de seus nomes, determina quais parâmetros serão usados para fornecer seus valores. No código a seguir, observe que os nomes de parâmetro estão fora de sequência no modelo de mensagem:

```
string p1 = "parm1";
string p2 = "parm2";
_logger.LogInformation("Parameter values: {p2}, {p1}", p1, p2);
```

Esse código cria uma mensagem de log com os valores de parâmetro na sequência:

```
Parameter values: parm1, parm2
```

A estrutura de registros funciona dessa maneira para que os provedores de logs possam implementar [registro em log semântico](#), também conhecido como [registro em log estruturado](#). Os próprios argumentos são passados para o sistema de registro em log, não apenas o modelo de mensagem formatado. Essas informações permitem que os provedores de log armazenem os valores de parâmetro como campos. Por exemplo, suponha que as chamadas de método do agente sejam assim:

```
_logger.LogInformation("Getting item {ID} at {RequestTime}", id, DateTime.Now);
```

Se você estiver enviando os logs para o Armazenamento de Tabelas do Azure, cada entidade da Tabela do Azure poderá ter propriedades `ID` e `RequestTime`, o que simplificará as consultas nos dados de log. Uma consulta pode encontrar todos os logs em determinado intervalo de `RequestTime` sem analisar o tempo limite da mensagem de texto.

## Exceções de registro em log

Os métodos de agente têm sobrecargas que permitem que você passe uma exceção, como no exemplo a seguir:

```
catch (Exception ex)
{
    _logger.LogWarning(LoggingEvents.GetItemNotFound, ex, "GetById({ID}) NOT FOUND", id);
    return NotFound();
}
return new ObjectResult(item);
```

```
catch (Exception ex)
{
    _logger.LogWarning(LoggingEvents.GetItemNotFound, ex, "GetById({ID}) NOT FOUND", id);
    return NotFound();
}
return new ObjectResult(item);
```

Provedores diferentes manipulam as informações de exceção de maneiras diferentes. Aqui está um exemplo da saída do provedor Depuração do código mostrado acima.

```
TodoApi.Controllers.TodoController:Warning: GetById(036dd898-fb01-47e8-9a65-f92eb73cf924) NOT FOUND
System.Exception: Item not found exception.
at TodoApi.Controllers.TodoController.GetById(String id) in
C:\logging\sample\src\TodoApi\Controllers\TodoController.cs:line 226
```

## Filtragem de log

Você pode especificar um nível de log mínimo para um provedor e uma categoria específicos ou para todos os provedores ou todas as categorias. Os logs abaixo do nível mínimo não serão passados para esse provedor, para que não sejam exibidos ou armazenados.

Para suprimir todos os logs, especifique `LogLevel.None` como o nível de log mínimo. O valor inteiro de `LogLevel.None` é 6, que é maior do que `LogLevel.Critical` (5).

### Criar regras de filtro na configuração

O código do modelo de projeto chama `CreateDefaultBuilder` para configurar o registro em log para os provedores Console e Depuração. O método `CreateDefaultBuilder` também configura o registro em log para procurar a configuração em uma seção `Logging`, usando código semelhante ao seguinte:

```

public static void Main(string[] args)
{
    var webHost = new WebHostBuilder()
        .UseKestrel()
        .UseContentRoot(Directory.GetCurrentDirectory())
        .ConfigureAppConfiguration((hostingContext, config) =>
    {
        var env = hostingContext.HostingEnvironment;
        config.AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
            .AddJsonFile($"appsettings.{env.EnvironmentName}.json",
                optional: true, reloadOnChange: true);
        config.AddEnvironmentVariables();
    })
    .ConfigureLogging((hostingContext, logging) =>
    {
        logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));
        logging.AddConsole();
        logging.AddDebug();
        logging.AddEventSourceLogger();
    })
    .UseStartup<Startup>()
    .Build();

    webHost.Run();
}

```

Os dados de configuração especificam níveis de log mínimo por provedor e por categoria, como no exemplo a seguir:

```

{
    "Logging": {
        "Debug": {
            "LogLevel": {
                "Default": "Information"
            }
        },
        "Console": {
            "IncludeScopes": false,
            "LogLevel": {
                "Microsoft.AspNetCore.Mvc.Razor.Internal": "Warning",
                "Microsoft.AspNetCore.Mvc.Razor.Razor": "Debug",
                "Microsoft.AspNetCore.Mvc.Razor": "Error",
                "Default": "Information"
            }
        },
        "LogLevel": {
            "Default": "Debug"
        }
    }
}

```

Este JSON cria seis regras de filtro, uma para o provedor Depuração, quatro para o provedor Console e uma para todos os provedores. Apenas uma regra é escolhida para cada provedor quando um objeto `ILogger` é criado.

## Regras de filtro no código

O exemplo a seguir mostra como registrar regras de filtro no código:

```

WebHost.CreateDefaultBuilder(args)
    .UseStartup<Startup>()
    .ConfigureLogging(logging =>
        logging.AddFilter("System", LogLevel.Debug)
            .AddFilter<DebugLoggerProvider>("Microsoft", LogLevel.Trace))
    .Build();

```

O segundo `AddFilter` especifica o provedor Depuração usando seu nome de tipo. O primeiro `AddFilter` se aplica a todos os provedores porque ele não especifica um tipo de provedor.

### Como as regras de filtragem são aplicadas

Os dados de configuração e o código `AddFilter`, mostrados nos exemplos anteriores, criam as regras mostradas na tabela a seguir. As primeiras seis vêm do exemplo de configuração e as últimas duas vêm do exemplo de código.

NÚMERO	PROVIDER	CATEGORIAS QUE COMEÇAM COM...	NÍVEL DE LOG MÍNIMO
1	Depurar	Todas as categorias	Informações
2	Console	Microsoft.AspNetCore.Mvc.Razor.Internal	Aviso
3	Console	Microsoft.AspNetCore.Mvc.Razor.Razor	Depurar
4	Console	Microsoft.AspNetCore.Mvc.Razor	Erro
5	Console	Todas as categorias	Informações
6	Todos os provedores	Todas as categorias	Depurar
7	Todos os provedores	Sistema	Depurar
8	Depurar	Microsoft	Rastrear

Quando um objeto `ILogger` é criado, o objeto `ILoggerFactory` seleciona uma única regra por provedor para aplicar a esse agente. Todas as mensagens gravadas pela instância `ILogger` são filtradas com base nas regras selecionadas. A regra mais específica possível para cada par de categoria e provedor é selecionada dentre as regras disponíveis.

O algoritmo a seguir é usado para cada provedor quando um `ILogger` é criado para uma determinada categoria:

- Selecione todas as regras que correspondem ao provedor ou seu alias. Se nenhuma correspondência for encontrada, selecione todas as regras com um provedor vazio.
- Do resultado da etapa anterior, selecione as regras com o prefixo de categoria de maior correspondência. Se nenhuma correspondência for encontrada, selecione todas as regras que não especificam uma categoria.
- Se várias regras forem selecionadas, use a **última**.
- Se nenhuma regra for selecionada, use `MinimumLevel`.

Com a lista anterior de regras, suponha que você crie um objeto `ILogger` para a categoria "Microsoft.AspNetCore.Mvc.Razor.RazorViewEngine":

- Para o provedor Depuração as regras 1, 6 e 8 se aplicam. A regra 8 é mais específica, portanto é a que será selecionada.
- Para o provedor Console as regras 3, 4, 5 e 6 se aplicam. A regra 3 é a mais específica.

A instância `ILogger` resultante envia logs de nível `Trace` e superior para o provedor Depuração. Logs de nível `Debug` e superior são enviados para o provedor Console.

### Aliases de provedor

Cada provedor define um *alias* que pode ser usado na configuração no lugar do nome do tipo totalmente qualificado. Para os provedores internos, use os seguintes aliases:

- `Console`
- `Depurar`
- `EventLog`
- `AzureAppServices`
- `TraceSource`
- `EventSource`

### Nível mínimo padrão

Há uma configuração de nível mínimo que entra em vigor somente se nenhuma regra de código ou de configuração se aplicar a um provedor e uma categoria determinados. O exemplo a seguir mostra como definir o nível mínimo:

```
WebHost.CreateDefaultBuilder(args)
    .UseStartup<Startup>()
    .ConfigureLogging(logging => logging.SetMinimumLevel(LogLevel.Warning))
    .Build();
```

Se você não definir explicitamente o nível mínimo, o valor padrão será `Information`, o que significa que logs `Trace` e `Debug` serão ignorados.

### Funções de filtro

Uma função de filtro é invocada para todos os provedores e categorias que não têm regras atribuídas a eles por configuração ou código. O código na função tem acesso ao tipo de provedor, à categoria e ao nível de log. Por exemplo:

```
WebHost.CreateDefaultBuilder(args)
    .UseStartup<Startup>()
    .ConfigureLogging(logBuilder =>
{
    logBuilder.AddFilter((provider, category, LogLevel) =>
    {
        if (provider == "Microsoft.Extensions.Logging.Console.ConsoleLoggerProvider" &&
            category == "TodoApiSample.Controllers.TodoController")
        {
            return false;
        }
        return true;
    });
})
    .Build();
```

Alguns provedores de log permitem especificar quando os logs devem ser gravados em uma mídia de armazenamento ou ignorados, com base no nível de log e na categoria.

Os métodos de extensão `AddConsole` e `AddDebug` fornecem sobrecargas que aceitam critérios de filtragem.

O código de exemplo a seguir faz com que o provedor de console ignore os logs abaixo do nível `Warning`, enquanto o provedor Depuração ignora os logs criados pela estrutura.

```
public void Configure(IApplicationBuilder app,
    IHostingEnvironment env,
    ILoggerFactory loggerFactory)
{
    loggerFactory
        .AddConsole(LogLevel.Warning)
        .AddDebug((category, logLevel) => (category.Contains("TodoApi") && logLevel >= LogLevel.Trace));
```

O método `AddEventLog` tem uma sobrecarga que recebe uma instância `EventLogSettings`, que pode conter uma função de filtragem em sua propriedade `Filter`. O provedor TraceSource não fornece nenhuma dessas sobrecargas, porque seu nível de registro em log e outros parâmetros são baseados no `SourceSwitch` e no `TraceListener` que ele usa.

Para definir regras de filtragem para todos os provedores registrados com uma instância `ILoggerFactory`, use o método de extensão `WithFilter`. O exemplo abaixo limita os logs de estrutura (categoria começa com "Microsoft" ou "Sistema") a avisos, enquanto registra em nível de depuração os logs criados por código de aplicativo.

```
public void Configure(IApplicationBuilder app,
    IHostingEnvironment env,
    ILoggerFactory loggerFactory)
{
    loggerFactory
        .WithFilter(new FilterLoggerSettings
        {
            { "Microsoft", LogLevel.Warning },
            { "System", LogLevel.Warning },
            { "ToDoApi", LogLevel.Debug }
        })
        .AddConsole()
        .AddDebug();
```

Para impedir a gravação de logs, especifique `LogLevel.None` como o nível de log mínimo. O valor inteiro de `LogLevel.None` é 6, que é maior do que `LogLevel.Critical` (5).

O método de extensão `WithFilter` é fornecido pelo pacote NuGet [Microsoft.Extensions.Logging.Filter](#). O método retorna um nova instância `ILoggerFactory`, que filtrará as mensagens de log passadas para todos os provedores de agente registrados com ela. Isso não afeta nenhuma outra instância de `ILoggerFactory`, incluindo a instância de `ILoggerFactory` original.

## Categorias e níveis de sistema

Veja algumas categorias usadas pelo ASP.NET Core e Entity Framework Core, com anotações sobre quais logs esperar delas:

CATEGORIA	OBSERVAÇÕES
Microsoft.AspNetCore	Diagnóstico geral de ASP.NET Core.
Microsoft.AspNetCore.DataProtection	Quais chaves foram consideradas, encontradas e usadas.
Microsoft.AspNetCore.HostFiltering	Hosts permitidos.

CATEGORIA	OBSERVAÇÕES
Microsoft.AspNetCore.Hosting	Quanto tempo levou para que as solicitações de HTTP fossem concluídas e em que horário foram iniciadas. Quais assemblies de inicialização de hospedagem foram carregados.
Microsoft.AspNetCore.Mvc	Diagnóstico do MVC e Razor. Model binding, execução de filtro, compilação de exibição, seleção de ação.
Microsoft.AspNetCore.Routing	Informações de correspondência de rotas.
Microsoft.AspNetCore.Server	Respostas de início, parada e atividade da conexão. Informações sobre o certificado HTTPS.
Microsoft.AspNetCore.StaticFiles	Arquivos atendidos.
Microsoft.EntityFrameworkCore	Diagnóstico geral do Entity Framework Core. Atividade e configuração do banco de dados, detecção de alterações, migrações.

## Escopos de log

Um *escopo* pode agrupar um conjunto de operações lógicas. Esse agrupamento pode ser usado para anexar os mesmos dados para cada log criado como parte de um conjunto. Por exemplo, todo log criado como parte do processamento de uma transação pode incluir a ID da transação.

Um escopo é um tipo `IDisposable` retornado pelo método `BeginScope` e que dura até que seja descartado. Use um escopo por meio do encapsulamento de chamadas de agente em um bloco `using`:

```
public IActionResult GetById(string id)
{
    TodoItem item;
    using (_logger.BeginScope("Message attached to logs created in the using block"))
    {
        _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
        item = _todoRepository.Find(id);
        if (item == null)
        {
            _logger.LogWarning(LoggingEvents.GetItemNotFound, ".GetById({ID}) NOT FOUND", id);
            return NotFound();
        }
    }
    return new ObjectResult(item);
}
```

O código a seguir habilita os escopos para o provedor de console:

*Program.cs*:

```
.ConfigureLogging((hostingContext, logging) =>
{
    logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));
    logging.AddConsole(options => options.IncludeScopes = true);
    logging.AddDebug();
})
```

#### NOTE

A configuração da opção de agente de console `IncludeScopes` é necessária para habilitar o registro em log baseado em escopo.

Saiba mais sobre como configurar na seção [Configuração](#).

*Program.cs:*

```
.ConfigureLogging((hostingContext, logging) =>
{
    logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));
    logging.AddConsole(options => options.IncludeScopes = true);
    logging.AddDebug();
})
```

#### NOTE

A configuração da opção de agente de console `IncludeScopes` é necessária para habilitar o registro em log baseado em escopo.

*Startup.cs:*

```
public void Configure(IApplicationBuilder app,
    IHostingEnvironment env,
    ILoggerFactory loggerFactory)
{
    loggerFactory
        .AddConsole(includeScopes: true)
        .AddDebug();
```

Cada mensagem de log inclui as informações com escopo definido:

```
info: TodoApi.Controllers.TodoController[1002]
    => RequestId:0HKV9C49II9CK RequestPath:/api/todo/0 => TodoApi.Controllers.TodoController.GetById
(TodoApi) => Message attached to logs created in the using block
    Getting item 0
warn: TodoApi.Controllers.TodoController[4000]
    => RequestId:0HKV9C49II9CK RequestPath:/api/todo/0 => TodoApi.Controllers.TodoController.GetById
(TodoApi) => Message attached to logs created in the using block
    GetById(0) NOT FOUND
```

## Provedores de log internos

O ASP.NET Core vem com os seguintes provedores:

- [Console](#)
- [Depurar](#)
- [EventSource](#)
- [EventLog](#)
- [TraceSource](#)

As opções de [Registro em log no Azure](#) serão abordadas mais adiante, neste artigo.

Para saber mais sobre log de stdout, confira [Solucionar problemas do ASP.NET Core no IIS](#) e [Solucionar](#)

problemas no ASP.NET Core no Serviço de Aplicativo do Azure.

## Provedor do console

O pacote de provedor `Microsoft.Extensions.Logging.Console` envia a saída de log para o console.

```
logging.AddConsole();
```

```
loggerFactory.AddConsole();
```

As [sobrecargas do AddConsole](#) permitem que você passe um nível de log mínimo, uma função de filtro e um valor booleano que indica se escopos são compatíveis. Outra opção é passar um objeto `IConfiguration`, que pode especificar suporte de escopos e níveis de log.

O provedor de console tem um impacto significativo no desempenho e geralmente não é adequado para uso em produção.

Quando você cria um novo projeto no Visual Studio, o método `AddConsole` tem essa aparência:

```
loggerFactory.AddConsole(Configuration.GetSection("Logging"));
```

Esse código se refere à seção `Logging` do arquivo `appSettings.json`:

```
{
  "Logging": {
    "Console": {
      "IncludeScopes": false
    },
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    }
  }
}
```

As configurações mostradas limitam os logs de estrutura a avisos, permitindo que o aplicativo faça registros no nível de depuração, conforme explicado na [Filtragem de log](#) seção. Para obter mais informações, consulte [Configuração](#).

Para ver a saída de registro em log de console, abra um prompt de comando na pasta do projeto e execute o seguinte comando:

```
dotnet run
```

## Depurar provedor

O pacote de provedor `Microsoft.Extensions.Logging.Debug` grava a saída de log usando a classe `System.Diagnostics.Debug` (chamadas de método `Debug.WriteLine`).

No Linux, esse provedor grava logs em `/var/log/message`.

```
logging.AddDebug();
```

```
loggerFactory.AddDebug();
```

As [sobrecargas de AddDebug](#) permitem que você passe um nível de log mínimo ou uma função de filtro.

## Provedor EventSource

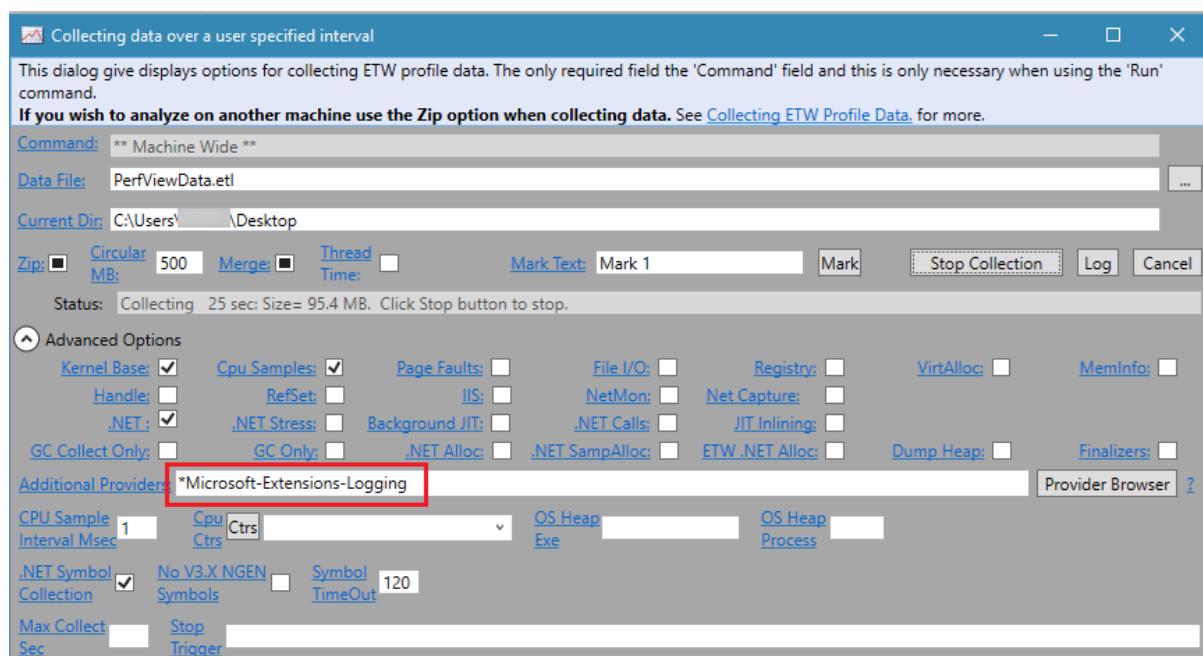
Para aplicativos que se destinam ao ASP.NET Core 1.1.0 ou posterior, o pacote de provedor [Microsoft.Extensions.Logging.EventSource](#) pode implementar o rastreamento de eventos. No Windows, ele usa [ETW](#). O provedor é multiplataforma, mas ainda não há ferramentas de coleta e exibição de eventos para Linux ou macOS.

```
logging.AddEventSourceLogger();
```

```
loggerFactory.AddEventSourceLogger();
```

Uma boa maneira de coletar e exibir logs é usar o [utilitário PerfView](#). Há outras ferramentas para exibir os logs do ETW, mas o PerfView proporciona a melhor experiência para trabalhar com os eventos de ETW emitidos pelo ASP.NET.

Para configurar o PerfView para coletar eventos registrados por esse provedor, adicione a cadeia de caracteres `*Microsoft-Extensions-Logging` à lista **Provedores Adicionais**. (Não se esqueça do asterisco no início da cadeia de caracteres).



## Provedor EventLog do Windows

O pacote de provedor [Microsoft.Extensions.Logging.EventLog](#) envia a saída de log para o Log de Eventos do Windows.

```
logging.AddEventLog();
```

```
loggerFactory.AddEventLog();
```

As [sobrecargas de AddEventLog](#) permitem que você passe `EventLogSettings` ou um nível de log mínimo.

## Provedor TraceSource

O pacote de provedor [Microsoft.Extensions.Logging.TraceSource](#) usa as bibliotecas e provedores de [TraceSource](#).

```
logging.AddTraceSource(sourceSwitchName);
```

```
loggerFactory.AddTraceSource(sourceSwitchName);
```

As [sobrecargas de AddTraceSource](#) permitem que você passe um comutador de fonte e um ouvinte de rastreamento.

Para usar esse provedor, o aplicativo deve ser executado no .NET Framework (em vez do .NET Core). O provedor pode rotear mensagens a uma variedade de [ouvintes](#), como o [TextWriterTraceListener](#) usado no aplicativo de exemplo.

O exemplo a seguir configura um provedor [TraceSource](#) que registra mensagens [Warning](#) e superiores na janela de console.

```
public void Configure(IApplicationBuilder app,
    IHostingEnvironment env,
    ILoggerFactory loggerFactory)
{
    loggerFactory
        .AddDebug();

    // add Trace Source logging
    var testSwitch = new SourceSwitch("sourceSwitch", "Logging Sample");
    testSwitch.Level = SourceLevels.Warning;
    loggerFactory.AddTraceSource(testSwitch,
        new TextWriterTraceListener(writer: Console.Out));
```

## Registro em log no Azure

Para saber mais sobre registro em log no Azure, consulte as seguintes seções:

- [Provedor do Serviço de Aplicativo do Azure](#)
- [Fluxo de log do Azure](#)
- [Log de rastreamento do Azure Application Insights](#)

### Provedor do Serviço de Aplicativo do Azure

O pacote de provedor [Microsoft.Extensions.Logging.AzureAppServices](#) grava logs em arquivos de texto no sistema de arquivos de um aplicativo do Serviço de Aplicativo do Azure e no [armazenamento de blobs](#) em uma conta de Armazenamento do Azure. O pacote de provedor está disponível para aplicativos destinados ao .NET Core 1.1 ou posterior.

Se você estiver direcionando para o .NET Core, observe os seguintes pontos:

- O pacote de provedor está incluído no [metapacote Microsoft.AspNetCore.All](#) do ASP.NET Core.
- O pacote de provedor não está incluído no [metapacote Microsoft.AspNetCore.App](#). Para usar o provedor, instale o pacote.
- Não chame explicitamente [AddAzureWebAppDiagnostics](#). Quando o aplicativo for implantado no Serviço de Aplicativo do Azure, o provedor ficará automaticamente disponível para ele.

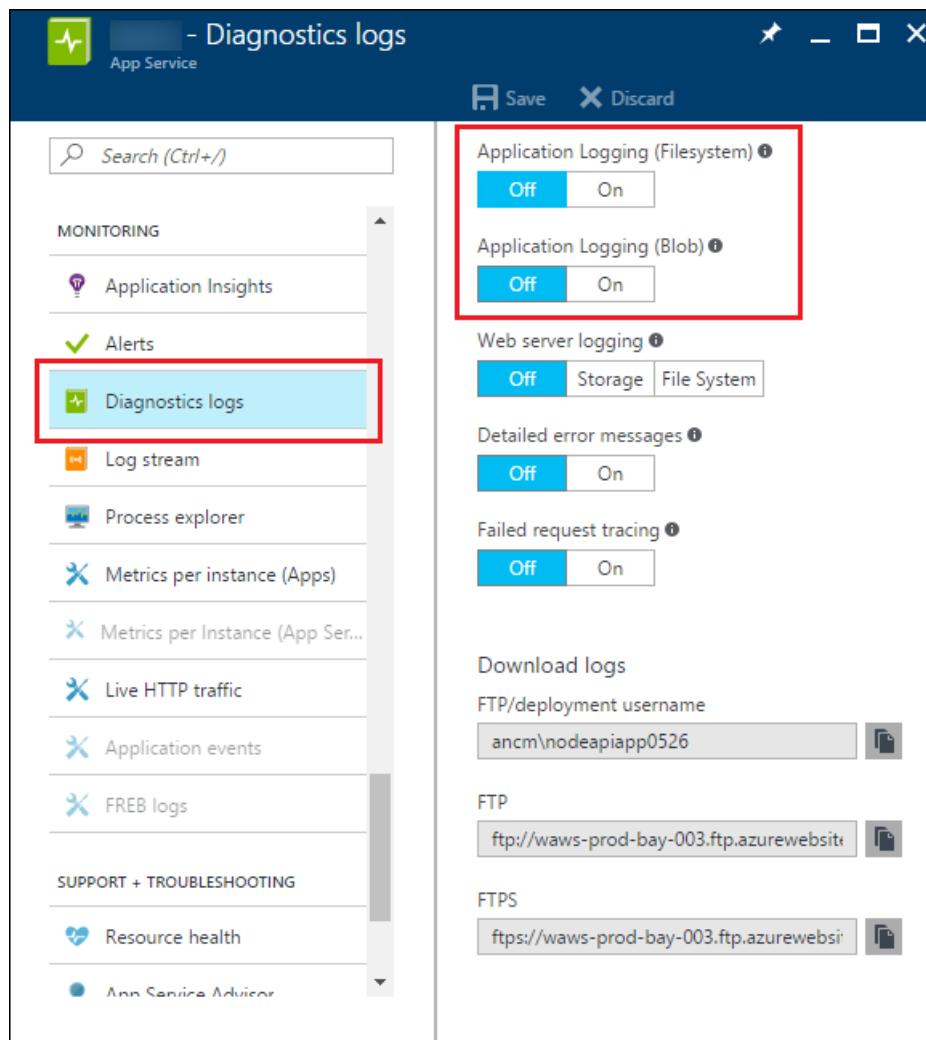
Se você estiver direcionando para o .NET Framework ou referenciando o metapacote

`Microsoft.AspNetCore.App`, adicione o pacote do provedor ao projeto. Invoque `AddAzureWebAppDiagnostics` em uma instância `ILoggerFactory`:

```
logging.AddAzureWebAppDiagnostics();  
  
loggerFactory.AddAzureWebAppDiagnostics();
```

Uma sobrecarga `AddAzureWebAppDiagnostics` permite passar `AzureAppServicesDiagnosticsSettings`. O objeto `settings` pode substituir as configurações padrão, como o modelo de saída de registro em log, o nome do blob e o limite do tamanho do arquivo. (O *modelo Output* é um modelo de mensagem aplicado a todos os logs, além daquele fornecido com uma chamada ao método `ILogger`).

Ao implantar um aplicativo do Serviço de Aplicativo, o aplicativo respeita as configurações na seção [Logs de Diagnóstico](#) da página **Serviço de Aplicativo** do portal do Azure. Quando essas configurações são atualizadas, as alterações entram em vigor imediatamente sem a necessidade de uma reinicialização ou reimplementação do aplicativo.



O local padrão para arquivos de log é na pasta `D:\home\LogFiles\Application` e o nome de arquivo padrão é `diagnostics-aaaammmdd.txt`. O limite padrão de tamanho do arquivo é 10 MB e o número padrão máximo de arquivos mantidos é 2. O nome de blob padrão é `{app-name}{timestamp}/aaaa/mm/dd/hh/{guid}-applicationLog.txt`. Para saber mais sobre o comportamento padrão, confira [AzureAppServicesDiagnosticsSettings](#).

O provedor funciona somente quando o projeto é executado no ambiente do Azure. Ele não tem nenhum efeito quando o projeto é executado localmente—ele não grava em arquivos locais ou no armazenamento

de desenvolvimento local para blobs.

## Fluxo de log do Azure

O fluxo de log do Azure permite que você exiba a atividade de log em tempo real:

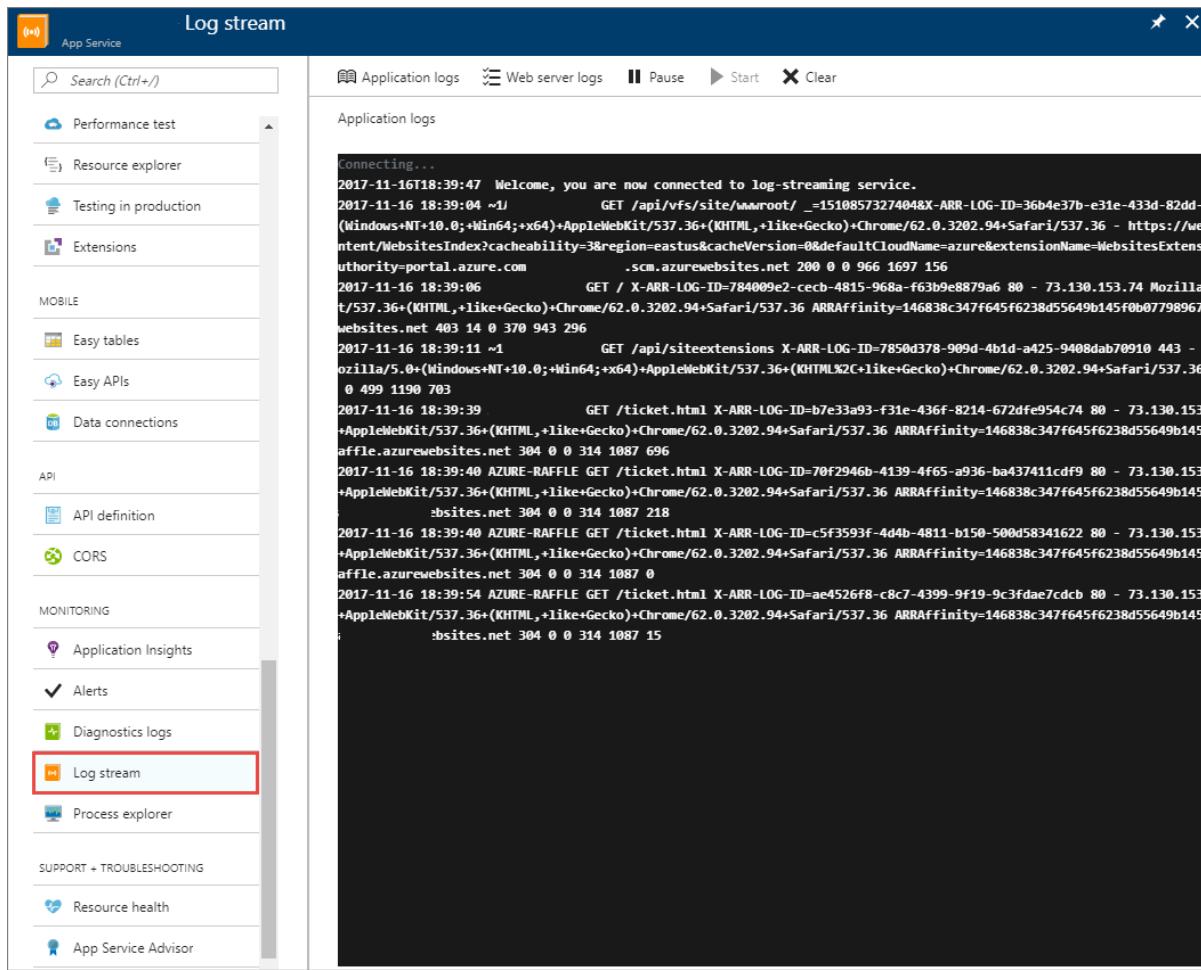
- O servidor de aplicativos
- Do servidor Web
- De uma solicitação de rastreio com falha

Para configurar o fluxo de log do Azure:

- Navegue até a página **Logs de Diagnóstico** da página do portal do seu aplicativo.
- Defina o **Log de aplicativo (Sistema de Arquivos)** como **Ativado**.

The screenshot shows the 'Diagnostics logs' configuration page for an App Service. On the left, a sidebar lists various monitoring and troubleshooting options. The 'Diagnostics logs' option is highlighted with a red box. The main pane contains settings for Application Logging (Filesystem), which is set to 'On'. Other sections include Application Logging (Blob), Web server logging (set to Storage), Detailed error messages, Failed request tracing, Download logs (FTP deployment username: 'azure-raffle\azure-ftp-deployment-user'), and FTP/FTPS connection details ('ftp://waws-prod-am2-135.ftp.azurewebsites.windows.net' and 'ftps://waws-prod-am2-135.ftp.azurewebsites.windows.net').

Navegue até a página **Fluxo de Log** para exibir as mensagens de aplicativo. Elas são registradas pelo aplicativo por meio da interface `ILogger`.



## Log de rastreamento do Azure Application Insights

O SDK do Application Insights pode coletar e relatar logs gerados por meio da infraestrutura de log do ASP.NET Core. Para obter mais informações, consulte os seguintes recursos:

- [Visão geral do Application Insights](#)
- [Application Insights para ASP.NET Core](#)
- [Application Insights logging adapters \(Adaptadores de registro em log do Application Insights\).](#)

## Provedores de log de terceiros

Estruturas de log de terceiros que funcionam com o ASP.NET Core:

- [elmah.io \(repositório GitHub\)](#)
- [Gelf \(repositório do GitHub\)](#)
- [JSNLog \(repositório GitHub\)](#)
- [KissLog.net \(Repositório do GitHub\)](#)
- [Loggr \(repositório GitHub\)](#)
- [NLog \(repositório GitHub\)](#)
- [Sentry \(repositório GitHub\)](#)
- [Serilog \(repositório GitHub\)](#)
- [Stackdriver \(repositório GitHub\)](#)

Algumas estruturas de terceiros podem fazer o [log semântico](#), também conhecido como [registro em log estruturado](#).

Usar uma estrutura de terceiros é semelhante ao uso de um dos provedores internos:

1. Adicione um pacote NuGet ao projeto.

2. Chame um `ILoggerFactory`.

Para saber mais, consulte a documentação de cada provedor. Não há suporte para provedores de log de terceiros na Microsoft.

## Recursos adicionais

- [Registro em log de alto desempenho com o LoggerMessage no ASP.NET Core](#)

# Páginas Razor do ASP.NET Core com EF Core – série de tutoriais

11/07/2018 • 2 minutes to read • [Edit Online](#)

Esta série de tutoriais ensina a criar aplicativos Web de Razor Pages do ASP.NET Core que usam o EF (Entity Framework) Core para o acesso a dados.

1. [Introdução](#)
2. [Operações Create, Read, Update e Delete](#)
3. [Classificação, filtragem, paginação e agrupamento](#)
4. [Migrações](#)
5. [Criar um modelo de dados complexo](#)
6. [Lendo dados relacionados](#)
7. [Atualizando dados relacionados](#)
8. [Tratar conflitos de simultaneidade](#)

# Páginas Razor com o Entity Framework Core no ASP.NET Core – Tutorial 1 de 8

16/01/2019 • 28 minutes to read • [Edit Online](#)

A versão deste tutorial para o ASP.NET Core 2.0 pode ser encontrada neste [arquivo PDF](#).

A versão deste tutorial para o ASP.NET Core 2.1 contém diversas melhorias em relação à versão 2.0.

Por [Tom Dykstra](#) e [Rick Anderson](#)

O aplicativo Web de exemplo Contoso University demonstra como criar um aplicativo Razor Pages do ASP.NET Core usando o EF (Entity Framework) Core.

O aplicativo de exemplo é um site de uma Contoso University fictícia. Ele inclui funcionalidades como admissão de alunos, criação de cursos e atribuições de instrutor. Esta página é a primeira de uma série de tutoriais que explica como criar o aplicativo de exemplo Contoso University.

[Baixe ou exiba o aplicativo concluído. Instruções de download.](#)

## Pré-requisitos

- [Visual Studio](#)
- [CLI do .NET Core](#)

[Visual Studio 2017 versão 15.7.3 ou posterior](#) com as cargas de trabalho a seguir:

- **ASP.NET e desenvolvimento para a Web**
- **Desenvolvimento entre plataformas do .NET Core**

[SDK do .NET Core 2.1 ou posteriores](#)

Familiaridade com as [Páginas do Razor](#). Os novos programadores devem concluir a [Introdução às Páginas do Razor](#) antes de começar esta série.

## Solução de problemas

Caso tenha um problema que não consiga resolver, em geral, você poderá encontrar a solução comparando o código com o [projeto concluído](#). Uma boa maneira de obter ajuda é postando uma pergunta no [StackOverflow.com](#) sobre o [ASP.NET Core](#) ou [EF Core](#).

## O aplicativo Web Contoso University

O aplicativo criado nesses tutoriais é um site básico de universidade.

Os usuários podem exibir e atualizar informações de alunos, cursos e instrutores. Veja a seguir algumas das telas criadas no tutorial.

The screenshot shows a web browser window titled "Index - Contoso University". The address bar displays "localhost:1234/Students". The main content area is titled "Index" and contains a "Create New" link. Below it is a search bar with the placeholder "Find by name:" and a "Search" button. A link "Back to full List" is also present. A table lists three student records:

Last Name	First Name	Enrollment Date	
Smith	Joe	2017-10-31	Edit   Details   Delete
Alexander	Carson	2010-09-01	Edit   Details   Delete
Alonso	Meredith	2012-09-01	Edit   Details   Delete

Navigation buttons "Previous" and "Next" are at the bottom. A copyright notice "© 2017 - Contoso University" is at the very bottom.

The screenshot shows a web browser window titled "Edit - Contoso University". The address bar displays "localhost:1234/Students/Edit/1". The main content area is titled "Edit" and has a subtitle "Student". It contains three form fields: "Last Name" (value: Alexander), "First Name" (value: Carson), and "Enrollment Date" (value: 09/01/2010). A "Save" button is at the bottom left. A "Back to List" link is at the bottom right. A copyright notice "© 2017 - Contoso University" is at the very bottom.

O estilo de interface do usuário deste site é próximo ao que é gerado pelos modelos internos. O foco do

Este tutorial é o EF Core com as Páginas do Razor, não a interface do usuário.

## Criar o aplicativo Web Razor Pages da ContosoUniversity

- [Visual Studio](#)
  - [CLI do .NET Core](#)
- No menu **Arquivo** do Visual Studio, selecione **Novo > Projeto**.
  - Crie um novo Aplicativo Web ASP.NET Core. Nomeie o projeto **ContosoUniversity**. É importante nomear o projeto *ContosoUniversity* para que os namespaces sejam correspondentes quando o código for copiado/colado.
  - Selecione **ASP.NET Core 2.1** na lista suspensa e selecione **Aplicativo Web**.

Para ver imagens das etapas anteriores, confira [Criar um aplicativo Web do Razor](#). Execute o aplicativo.

## Configurar o estilo do site

Algumas alterações configuraram o menu do site, o layout e a home page. Atualize `Pages/Shared/_Layout.cshtml` com as seguintes alterações:

- Altere cada ocorrência de "ContosoUniversity" para "Contoso University". Há três ocorrências.
- Adicione entradas de menu para **Alunos, Cursos, Instrutores e Departamentos** e exclua a entrada de menu **Contato**.

As alterações são realçadas. (Toda a marcação *não* é exibida.)

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ ViewData["Title"] : Contoso University</title>

    <environment include="Development">
        <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
        <link rel="stylesheet" href("~/css/site.css" />
    </environment>
    <environment exclude="Development">
        <link rel="stylesheet"
            href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/css/bootstrap.min.css"
            asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"
            asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-
            test-value="absolute" />
        <link rel="stylesheet" href "~/css/site.min.css" asp-append-version="true" />
    </environment>
</head>
<body>
    <nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-
                target=".navbar-collapse">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a asp-page="/Index" class="navbar-brand">Contoso University</a>
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li><a asp-page="/Index">Home</a></li>
                    <li><a asp-page="/About">About</a></li>
                    <li><a asp-page="/Students/Index">Students</a></li>
                    <li><a asp-page="/Courses/Index">Courses</a></li>
                    <li><a asp-page="/Instructors/Index">Instructors</a></li>
                    <li><a asp-page="/Departments/Index">Departments</a></li>
                </ul>
            </div>
        </div>
    </nav>

    <partial name="_CookieConsentPartial" />

    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; 2018 : Contoso University</p>
        </footer>
    </div>

```

@\*Remaining markup not shown for brevity.\*@

Em *Pages/Index.cshtml*, substitua o conteúdo do arquivo pelo seguinte código para substituir o texto sobre o ASP.NET e MVC pelo texto sobre este aplicativo:

```

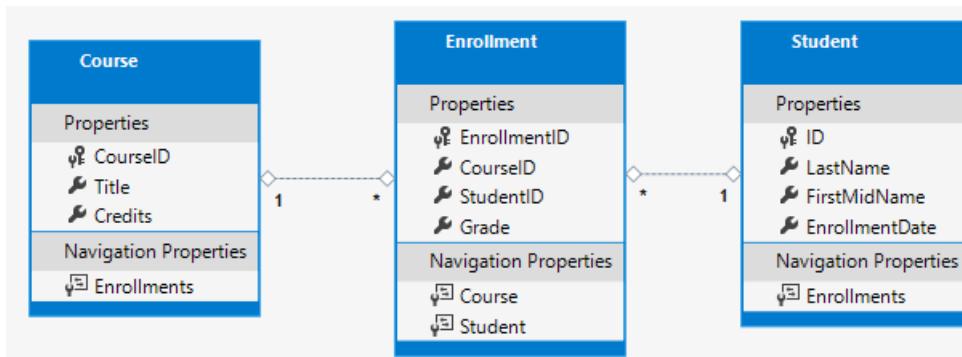
@page
@model IndexModel
 @{
     ViewData["Title"] = "Home page";
 }

 <div class="jumbotron">
    <h1>Contoso University</h1>
</div>
<div class="row">
    <div class="col-md-4">
        <h2>Welcome to Contoso University</h2>
        <p>
            Contoso University is a sample application that
            demonstrates how to use Entity Framework Core in an
            ASP.NET Core Razor Pages web app.
        </p>
    </div>
    <div class="col-md-4">
        <h2>Build it from scratch</h2>
        <p>You can build the application by following the steps in a series of tutorials.</p>
        <p>
            <a class="btn btn-default"
                href="https://docs.microsoft.com/aspnet/core/data/ef-rp/intro">
                See the tutorial &raquo;
            </a>
        </p>
    </div>
    <div class="col-md-4">
        <h2>Download it</h2>
        <p>You can download the completed project from GitHub.</p>
        <p>
            <a class="btn btn-default"
                href="https://github.com/aspnet/Docs/tree/master/aspnetcore/data/ef-rp/intro/samples/cu-final">
                See project source code &raquo;
            </a>
        </p>
    </div>
</div>

```

## Criar o modelo de dados

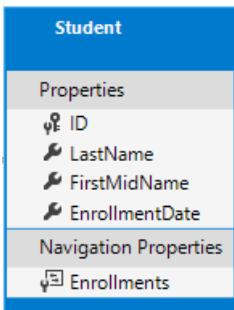
Crie classes de entidade para o aplicativo Contoso University. Comece com as três seguintes entidades:



Há uma relação um-para-muitos entre as entidades `Student` e `Enrollment`. Há uma relação um-para-muitos entre as entidades `Course` e `Enrollment`. Um aluno pode se registrar em qualquer quantidade de cursos. Um curso pode ter qualquer quantidade de alunos registrados.

Nas seções a seguir, é criada uma classe para cada uma dessas entidades.

### A entidade Student



Crie uma pasta *Models*. Na pasta *Models*, crie um arquivo de classe chamado *Student.cs* com o seguinte código:

```
using System;
using System.Collections.Generic;

namespace ContosoUniversity.Models
{
    public class Student
    {
        public int ID { get; set; }
        public string LastName { get; set; }
        public string FirstMidName { get; set; }
        public DateTime EnrollmentDate { get; set; }

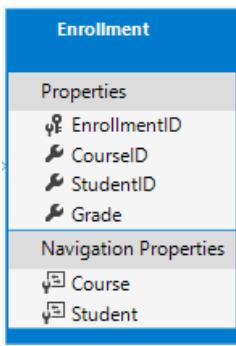
        public ICollection<Enrollment> Enrollments { get; set; }
    }
}
```

A propriedade `ID` se torna a coluna de chave primária da tabela de BD (banco de dados) que corresponde a essa classe. Por padrão, o EF Core interpreta uma propriedade nomeada `ID` ou `classnameID` como a chave primária. Em `classnameID`, `classname` é o nome da classe. A chave primária alternativa reconhecida automaticamente é `StudentID` no exemplo anterior.

A propriedade `Enrollments` é uma [propriedade de navegação](#). As propriedades de navegação vinculam-se a outras entidades que estão relacionadas a essa entidade. Nesse caso, a propriedade `Enrollments` de uma `Student` entity armazena todas as entidades `Enrollment` relacionadas a essa `student`. Por exemplo, se uma linha Aluno no BD tiver duas linhas Registro relacionadas, a propriedade de navegação `Enrollments` conterá duas entidades `Enrollment`. Uma linha `Enrollment` relacionada é uma linha que contém o valor de chave primária do aluno na coluna `StudentID`. Por exemplo, suponha que o aluno com ID=1 tenha duas linhas na tabela `Enrollment`. A tabela `Enrollment` tem duas linhas com `StudentID` = 1. `StudentID` é uma chave estrangeira na tabela `Enrollment` que especifica o aluno na tabela `Student`.

Se uma propriedade de navegação puder armazenar várias entidades, a propriedade de navegação deverá ser um tipo de lista, como `ICollection<T>`. `ICollection<T>` pode ser especificado ou um tipo como `List<T>` ou `HashSet<T>`. Quando `ICollection<T>` é usado, o EF Core cria uma coleção `HashSet<T>` por padrão. As propriedades de navegação que armazenam várias entidades são provenientes de relações muitos para muitos e um-para-muitos.

## A entidade Enrollment



Na pasta *Models*, crie *Enrollment.cs* com o seguinte código:

```
namespace ContosoUniversity.Models
{
    public enum Grade
    {
        A, B, C, D, F
    }

    public class Enrollment
    {
        public int EnrollmentID { get; set; }
        public int CourseID { get; set; }
        public int StudentID { get; set; }
        public Grade? Grade { get; set; }

        public Course Course { get; set; }
        public Student Student { get; set; }
    }
}
```

A propriedade `EnrollmentID` é a chave primária. Essa entidade usa o padrão `classnameID` em vez de `ID` como a entidade `Student`. Normalmente, os desenvolvedores escolhem um padrão e o usam em todo o modelo de dados. Em um tutorial posterior, o uso de uma ID sem nome de classe é mostrado para facilitar a implementação da herança no modelo de dados.

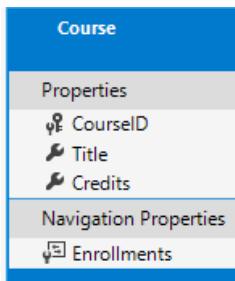
A propriedade `Grade` é um `enum`. O ponto de interrogação após a declaração de tipo `Grade` indica que a propriedade `Grade` permite valor nulo. Uma nota nula é diferente de uma nota zero – nulo significa que uma nota não é conhecida ou que ainda não foi atribuída.

A propriedade `StudentID` é uma chave estrangeira e a propriedade de navegação correspondente é `Student`. Uma entidade `Enrollment` está associada a uma entidade `student` e, portanto, a propriedade contém uma única entidade `student`. A entidade `Student` é distinta da propriedade de navegação `Student.Enrollments`, que contém várias entidades `Enrollment`.

A propriedade `CourseID` é uma chave estrangeira e a propriedade de navegação correspondente é `Course`. Uma entidade `Enrollment` está associada a uma entidade `Course`.

O EF Core interpreta uma propriedade como uma chave estrangeira se ela é nomeada `<navigation property name><primary key property name>`. Por exemplo, `StudentID` para a propriedade de navegação `Student`, pois a chave primária da entidade `Student` é `ID`. Propriedades de chave estrangeira também podem ser nomeadas `<primary key property name>`. Por exemplo, `CourseID`, pois a chave primária da entidade `Course` é `CourseID`.

## A entidade Course



Na pasta *Models*, crie *Course.cs* com o seguinte código:

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Course
    {
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int CourseID { get; set; }
        public string Title { get; set; }
        public int Credits { get; set; }

        public ICollection<Enrollment> Enrollments { get; set; }
    }
}
```

A propriedade `Enrollments` é uma propriedade de navegação. Uma entidade `Course` pode estar relacionada a qualquer quantidade de entidades `Enrollment`.

O atributo `DatabaseGenerated` permite que o aplicativo especifique a chave primária em vez de fazer com que ela seja gerada pelo BD.

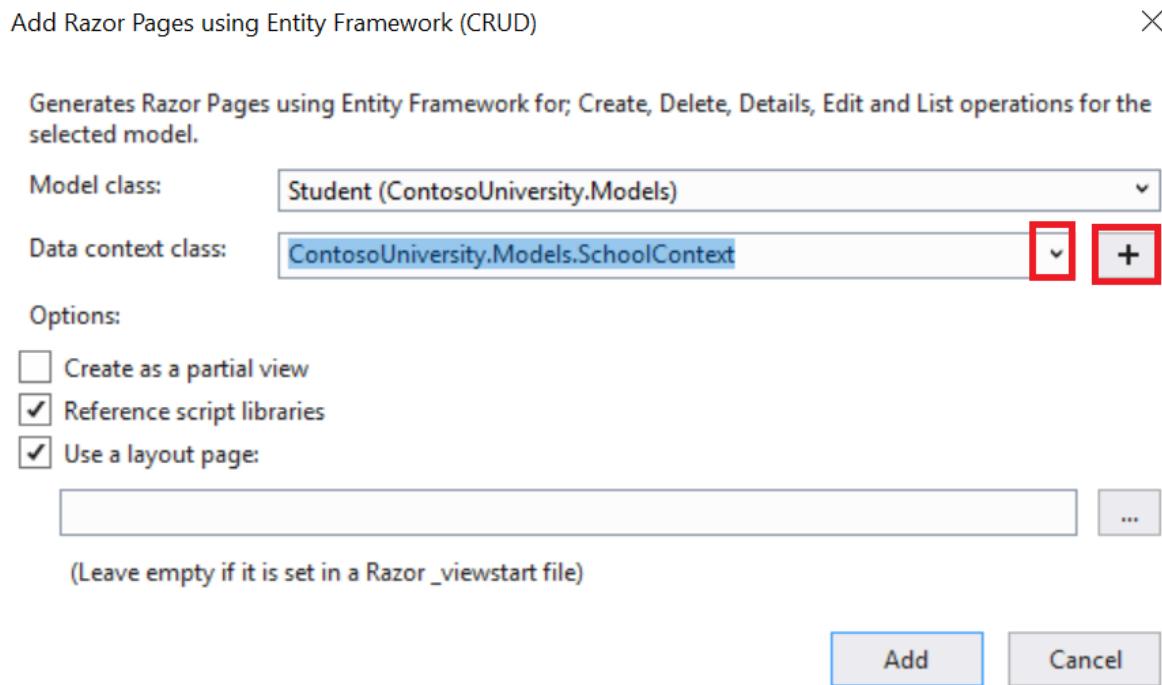
## Gere um modelo de aluno por scaffold

Nesta seção, é feito o scaffold do modelo de aluno. Ou seja, a ferramenta de scaffolding gera páginas para operações de CRUD (Criar, Ler, Atualizar e Excluir) para o modelo de aluno.

- Compile o projeto.
- Crie a pasta *Pages/Students*.
  - [Visual Studio](#)
  - [CLI do .NET Core](#)
- No **Gerenciador de Soluções**, clique com o botão direito do mouse na pasta *Páginas/Alunos* pasta > **Adicionar > Novo item com scaffold**.
- Na caixa de diálogo **Adicionar Scaffold**, selecione **Razor Pages usando o Entity Framework (CRUD) > Adicionar**.

Conclua a caixa de diálogo **Adicionar Razor Pages usando o Entity Framework (CRUD)**:

- Na lista suspensa **classe Modelo**, selecione **Aluno (ContosoUniversity.Models)**.
- Na linha **Classe de contexto de dados**, selecione o sinal de (mais) + e altere o nome gerado para **ContosoUniversity.Models.SchoolContext**.
- Na lista suspensa **Classe de contexto de dados**, selecione **ContosoUniversity.Models.SchoolContext**
- Selecione **Adicionar**.



Confira [Fazer scaffold do modelo de filme](#) se tiver problemas na etapa anterior.

O processo de scaffold criou e alterou os seguintes arquivos:

#### Arquivos criados

- *Pages/Students* Criar, Excluir, Detalhes, Editar, Índice.
- *Data/SchoolContext.cs*

#### Atualizações de arquivo

- *Startup.cs*: alterações a esse arquivo serão detalhadas na próxima seção.
- *appsettings.json*: a cadeia de conexão usada para se conectar a um banco de dados local é adicionada.

## Examinar o contexto registrado com a injeção de dependência

O ASP.NET Core é construído com a [injeção de dependência](#). Serviços (como o contexto de BD do EF Core) são registrados com injeção de dependência durante a inicialização do aplicativo. Os componentes que exigem esses serviços (como as Páginas do Razor) recebem esses serviços por meio de parâmetros do construtor. O código de construtor que obtém uma instância de contexto do BD é mostrado mais adiante no tutorial.

A ferramenta de scaffolding criou automaticamente um contexto de BD e o registrou no contêiner da injeção de dependência.

Examine o método `ConfigureServices` em *Startup.cs*. A linha destacada foi adicionada pelo scaffolder:

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        // This lambda determines whether user consent for
        // non-essential cookies is needed for a given request.
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    services.AddDbContext<SchoolContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("SchoolContext")));
}
```

O nome da cadeia de conexão é passado para o contexto com a chamada de um método em um objeto [DbContextOptions](#). Para o desenvolvimento local, o [sistema de configuração do ASP.NET Core](#) lê a cadeia de conexão do arquivo *appsettings.json*.

## Atualizar o principal

Em *Program.cs*, modifique o método `Main` para fazer o seguinte:

- Obtenha uma instância de contexto de BD do contêiner de injeção de dependência.
- Chame o [EnsureCreated](#).
- Descarte o contexto quando o método `EnsureCreated` for concluído.

O código a seguir mostra o arquivo *Program.cs* atualizado.

```

using ContosoUniversity.Models; // SchoolContext
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection; // CreateScope
using Microsoft.Extensions.Logging;
using System;

namespace ContosoUniversity
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var host = CreateWebHostBuilder(args).Build();

            using (var scope = host.Services.CreateScope())
            {
                var services = scope.ServiceProvider;

                try
                {
                    var context = services.GetRequiredService<SchoolContext>();
                    context.Database.EnsureCreated();
                }
                catch (Exception ex)
                {
                    var logger = services.GetRequiredService<ILogger<Program>>();
                    logger.LogError(ex, "An error occurred creating the DB.");
                }
            }

            host.Run();
        }

        public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>();
    }
}

```

`EnsureCreated` garante que o banco de dados do contexto exista. Se ele existir, nenhuma ação será realizada. Se ele não existir, o banco de dados e todos os seus esquemas serão criados. `EnsureCreated` não usa migrações para criar o banco de dados. Um banco de dados criado com `EnsureCreated` não pode ser atualizado posteriormente usando migrações.

`EnsureCreated` é chamado na inicialização do aplicativo, que permite que o seguinte fluxo de trabalho:

- Exclua o BD.
- Altere o esquema de BD (por exemplo, adicione um campo `EmailAddress`).
- Execute o aplicativo.
- `EnsureCreated` cria um BD com a coluna `EmailAddress`.

`EnsureCreated` é conveniente no início do desenvolvimento quando o esquema está evoluindo rapidamente. Mais tarde, no tutorial do banco de dados, é excluído e as migrações são usadas.

## Testar o aplicativo

Execute o aplicativo e aceite a política de cookies. Este aplicativo não armazena informações pessoais. Você pode ler sobre a política de cookies em [Suporte para a RGPD \(Regulamento Geral sobre a Proteção de Dados\) da UE](#).

- Selecione o link **Alunos e Criar Novo**.

- Teste os links Editar, Detalhes e Excluir.

## Examine o contexto de BD SchoolContext

A classe principal que coordena a funcionalidade do EF Core de um modelo de dados é a classe de contexto de BD. O contexto de dados deriva de [Microsoft.EntityFrameworkCore.DbContext](#). O contexto de dados especifica quais entidades são incluídas no modelo de dados. Neste projeto, a classe é chamada `SchoolContext`.

Atualize `SchoolContext.cs` com o seguinte código:

```
using Microsoft.EntityFrameworkCore;

namespace ContosoUniversity.Models
{
    public class SchoolContext : DbContext
    {
        public SchoolContext(DbContextOptions<SchoolContext> options)
            : base(options)
        {
        }

        public DbSet<Student> Student { get; set; }
        public DbSet<Enrollment> Enrollment { get; set; }
        public DbSet<Course> Course { get; set; }
    }
}
```

O código destacado cria uma propriedade `DbSet< TEntity >` para cada conjunto de entidades. Na terminologia do EF Core:

- Um conjunto de entidades normalmente corresponde a uma tabela de BD.
- Uma entidade corresponde a uma linha da tabela.

`DbSet<Enrollment>` e `DbSet<Course>` podem ser omitidos. O EF Core inclui-os de forma implícita porque a entidade `Student` referencia a entidade `Enrollment` e a entidade `Enrollment` referencia a entidade `Course`. Para este tutorial, mantenha `DbSet<Enrollment>` e `DbSet<Course>` no `SchoolContext`.

### SQL Server Express LocalDB

A cadeia de conexão especifica um [LocalDB do SQL Server](#). LocalDB é uma versão leve do Mecanismo de Banco de Dados do SQL Server Express destinado ao desenvolvimento de aplicativos, e não ao uso em produção. O LocalDB é iniciado sob demanda e executado no modo de usuário e, portanto, não há nenhuma configuração complexa. Por padrão, o LocalDB cria arquivos `.mdf` de BD no diretório `C:/Users/<user>`.

## Adicionar um código para inicializar o BD com os dados de teste

O EF Core cria um BD vazio. Nesta seção, um método `Initialize` é escrito para populá-lo com os dados de teste.

Na pasta `Dados`, crie um novo arquivo de classe chamado `DbInitializer.cs` e adicione o seguinte código:

```
using ContosoUniversity.Models;
using System;
using System.Linq;

namespace ContosoUniversity.Models
{
    public static class DbInitializer
```

```

    {
        public static void Initialize(SchoolContext context)
        {
            // context.Database.EnsureCreated();

            // Look for any students.
            if (context.Student.Any())
            {
                return; // DB has been seeded
            }

            var students = new Student[]
            {
                new Student{FirstMidName="Carson", LastName="Alexander", EnrollmentDate=DateTime.Parse("2005-09-01")},
                new Student{FirstMidName="Meredith", LastName="Alonso", EnrollmentDate=DateTime.Parse("2002-09-01")},
                new Student{FirstMidName="Arturo", LastName="Anand", EnrollmentDate=DateTime.Parse("2003-09-01")},
                new Student{FirstMidName="Gytis", LastName="Barzdukas", EnrollmentDate=DateTime.Parse("2002-09-01")},
                new Student{FirstMidName="Yan", LastName="Li", EnrollmentDate=DateTime.Parse("2002-09-01")},
                new Student{FirstMidName="Peggy", LastName="Justice", EnrollmentDate=DateTime.Parse("2001-09-01")},
                new Student{FirstMidName="Laura", LastName="Norman", EnrollmentDate=DateTime.Parse("2003-09-01")},
                new Student{FirstMidName="Nino", LastName="Olivetto", EnrollmentDate=DateTime.Parse("2005-09-01")}
            };
            foreach (Student s in students)
            {
                context.Student.Add(s);
            }
            context.SaveChanges();

            var courses = new Course[]
            {
                new Course{CourseID=1050, Title="Chemistry", Credits=3},
                new Course{CourseID=4022, Title="Microeconomics", Credits=3},
                new Course{CourseID=4041, Title="Macroeconomics", Credits=3},
                new Course{CourseID=1045, Title="Calculus", Credits=4},
                new Course{CourseID=3141, Title="Trigonometry", Credits=4},
                new Course{CourseID=2021, Title="Composition", Credits=3},
                new Course{CourseID=2042, Title="Literature", Credits=4}
            };
            foreach (Course c in courses)
            {
                context.Course.Add(c);
            }
            context.SaveChanges();

            var enrollments = new Enrollment[]
            {
                new Enrollment{StudentID=1, CourseID=1050, Grade=Grade.A},
                new Enrollment{StudentID=1, CourseID=4022, Grade=Grade.C},
                new Enrollment{StudentID=1, CourseID=4041, Grade=Grade.B},
                new Enrollment{StudentID=2, CourseID=1045, Grade=Grade.B},
                new Enrollment{StudentID=2, CourseID=3141, Grade=Grade.F},
                new Enrollment{StudentID=2, CourseID=2021, Grade=Grade.F},
                new Enrollment{StudentID=3, CourseID=1050},
                new Enrollment{StudentID=4, CourseID=1050},
                new Enrollment{StudentID=4, CourseID=4022, Grade=Grade.F},
                new Enrollment{StudentID=5, CourseID=4041, Grade=Grade.C},
                new Enrollment{StudentID=6, CourseID=1045},
                new Enrollment{StudentID=7, CourseID=3141, Grade=Grade.A},
            };
            foreach (Enrollment e in enrollments)
            {
                context.Enrollment.Add(e);
            }
        }
    }

```

```

        context.ChangeTracker.DetectChanges(),
    }
    context.SaveChanges();
}
}

```

Observação: O código anterior usa `Models` para o namespace (`namespace ContosoUniversity.Models`) em vez de `Data`. `Models` é consistente com o código gerado pelo scaffolder. Para saber mais, confira [este problema de scaffolding do GitHub](#).

O código verifica se há alunos no BD. Se não houver nenhum aluno no BD, o BD será inicializado com os dados de teste. Ele carrega os dados de teste em matrizes em vez de em coleções `List<T>` para otimizar o desempenho.

O método `EnsureCreated` cria o BD automaticamente para o contexto de BD. Se o BD existir, `EnsureCreated` retornará sem modificar o BD.

Em `Program.cs`, modifique o método `Main` para chamar `Initialize`:

```

public class Program
{
    public static void Main(string[] args)
    {
        var host = CreateWebHostBuilder(args).Build();

        using (var scope = host.Services.CreateScope())
        {
            var services = scope.ServiceProvider;

            try
            {
                var context = services.GetRequiredService<SchoolContext>();
                // using ContosoUniversity.Data;
                DbInitializer.Initialize(context);
            }
            catch (Exception ex)
            {
                var logger = services.GetRequiredService<ILogger<Program>>();
                logger.LogError(ex, "An error occurred creating the DB.");
            }
        }

        host.Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>();
}

```

Exclua todos os registros de alunos e reinicie o aplicativo. Se o BD não for inicializado, defina um ponto de interrupção em `Initialize` para diagnosticar o problema.

## Exibir o BD

Abra o **SSOX** (Pesquisador de Objetos do SQL Server) no menu **Exibir** do Visual Studio. No SSOX, clique em **(localdb)\MSSQLLocalDB > Bancos de Dados > ContosoUniversity1**.

Expanda o nó **Tabelas**.

Clique com o botão direito do mouse na tabela **Aluno** e clique em **Exibir Dados** para ver as colunas

criadas e as linhas inseridas na tabela.

## Código assíncrono

A programação assíncrona é o modo padrão do ASP.NET Core e EF Core.

Um servidor Web tem um número limitado de threads disponíveis e, em situações de alta carga, todos os threads disponíveis podem estar em uso. Quando isso acontece, o servidor não pode processar novas solicitações até que os threads são liberados. Com um código síncrono, muitos threads podem ser vinculados enquanto realmente não estão fazendo nenhum trabalho porque estão aguardando a conclusão da E/S. Com um código assíncrono, quando um processo está aguardando a conclusão da E/S, seu thread é liberado para o servidor para ser usado para processar outras solicitações. Como resultado, o código assíncrono permite que os recursos do servidor sejam usados com mais eficiência, e o servidor fica capacitado a manipular mais tráfego sem atrasos.

O código assíncrono introduz uma pequena quantidade de sobrecarga em tempo de execução. Para situações de baixo tráfego, o impacto no desempenho é insignificante, enquanto para situações de alto tráfego, a melhoria de desempenho potencial é significativa.

No código a seguir, a palavra-chave `async`, o valor retornado `Task<T>`, a palavra-chave `await` e o método `ToListAsync` fazem o código ser executado de forma assíncrona.

```
public async Task OnGetAsync()
{
    Student = await _context.Student.ToListAsync();
}
```

- A palavra-chave `async` instrui o compilador a:
  - Gerar retornos de chamada para partes do corpo do método.
  - Criar automaticamente o objeto `Task` que é retornado. Para obter mais informações, consulte [Tipo de retorno de Tarefa](#).
- O tipo de retorno implícito `Task` representa um trabalho em andamento.
- A palavra-chave `await` faz com que o compilador divida o método em duas partes. A primeira parte termina com a operação que é iniciada de forma assíncrona. A segunda parte é colocada em um método de retorno de chamada que é chamado quando a operação é concluída.
- `ToListAsync` é a versão assíncrona do método de extensão `ToList`.

Algumas coisas a serem consideradas ao escrever um código assíncrono que usa o EF Core:

- Somente instruções que fazem com que consultas ou comandos sejam enviados ao BD são executadas de forma assíncrona. Isso inclui `ToListAsync`, `SingleOrDefaultAsync`, `FirstOrDefaultAsync` e `SaveChangesAsync`. Isso não inclui instruções que apenas alteram um `IQueryable`, como  
`var students = context.Students.Where(s => s.LastName == "Davolio")`.
- Um contexto do EF Core não é thread-safe: não tente realizar várias operações em paralelo.
- Para aproveitar os benefícios de desempenho do código assíncrono, verifique se os pacotes de biblioteca (como para paginação) usam o código assíncrono se eles chamam métodos do EF Core que enviam consultas ao BD.

Para obter mais informações sobre a programação assíncrona, consulte [Visão geral de Async](#) e [Programação assíncrona com async e await](#).

No próximo tutorial, as operações CRUD (criar, ler, atualizar e excluir) básicas são examinadas.

AVANÇAR

# Páginas Razor com o EF Core no ASP.NET Core – CRUD – 2 de 8

30/10/2018 • 21 minutes to read • [Edit Online](#)

A versão deste tutorial para o ASP.NET Core 2.0 pode ser encontrada neste [arquivo PDF](#).

A versão deste tutorial para o ASP.NET Core 2.1 contém diversas melhorias em relação à versão 2.0.

Por [Tom Dykstra](#), [Jon P Smith](#) e [Rick Anderson](#)

O aplicativo Web Contoso University demonstra como criar aplicativos Web das Páginas do Razor usando o EF Core e o Visual Studio. Para obter informações sobre a série de tutoriais, consulte [o primeiro tutorial](#).

Neste tutorial, o código CRUD (criar, ler, atualizar e excluir) gerado por scaffolding é examinado e personalizado.

Para minimizar a complexidade e manter o foco destes tutoriais no EF Core, o código do EF Core é usado nos modelos de página. Alguns desenvolvedores usam um padrão de repositório ou de camada de serviço para criar uma camada de abstração entre a interface do usuário (Páginas do Razor) e a camada de acesso a dados.

Neste tutorial, as Razor Pages Criar, Editar, Excluir e Detalhes na pasta *Student* são examinadas.

O código gerado por scaffolding usa o seguinte padrão para as páginas Criar, Editar e Excluir:

- Obtenha e exiba os dados solicitados com o método HTTP GET `OnGetAsync`.
- Salve as alterações nos dados com o método HTTP POST `OnPostAsync`.

As páginas Índice e Detalhes obtêm e exibem os dados solicitados com o método HTTP GET `OnGetAsync`

## SingleOrDefaultAsync vs. FirstOrDefaultAsync

O código gerado usa [FirstOrDefaultAsync](#), que geralmente é uma preferência em relação a [SingleOrDefaultAsync](#).

`FirstOrDefaultAsync` é mais eficiente que `SingleOrDefaultAsync` na busca de uma entidade:

- A menos que o código precise verificar se não há mais de uma entidade retornada da consulta.
- `SingleOrDefaultAsync` busca mais dados e faz o trabalho desnecessário.
- `SingleOrDefaultAsync` gera uma exceção se há mais de uma entidade que se ajusta à parte do filtro.
- `FirstOrDefaultAsync` não gera uma exceção se há mais de uma entidade que se ajusta à parte do filtro.

## FindAsync

Em grande parte do código gerado por scaffolding, [FindAsync](#) pode ser usado no lugar de `FirstOrDefaultAsync`.

`FindAsync`:

- Encontra uma entidade com o PK (chave primária). Se uma entidade com o PK está sendo controlada pelo contexto, ela é retornada sem uma solicitação para o BD.
- É simples e conciso.
- É otimizado para pesquisar uma única entidade.
- Pode ter benefícios de desempenho em algumas situações, mas isso é raro em aplicativos Web.
- Usa [FirstAsync](#) em vez de [SingleAsync](#) de forma implícita.

Mas se você deseja `Include` outras entidades, `FindAsync` não é mais apropriado. Isso significa que talvez seja

necessário abandonar `FindAsync` e passar para uma consulta à medida que o aplicativo avança.

## Personalizar a página Detalhes

Navegue para a página `Pages/Students`. Os links **Editar**, **Detalhes** e **Excluir** são gerados pelo [Auxiliar de Marcação de Âncora](#) no arquivo `Pages/Students/Index.cshtml`.

```
<td>
    <a asp-page=".Edit" asp-route-id="@item.ID">Edit</a> |
    <a asp-page=".Details" asp-route-id="@item.ID">Details</a> |
    <a asp-page=".Delete" asp-route-id="@item.ID">Delete</a>
</td>
```

Execute o aplicativo e selecione o link **Detalhes**. A URL tem o formato

`http://localhost:5000/Students/Details?id=2`. A ID do Aluno é passada com uma cadeia de caracteres de consulta (`?id=2`).

Atualize as Páginas Editar, Detalhes e Excluir do Razor para que elas usem o modelo de rota `{id:int}`. Altere a diretiva de página de cada uma dessas páginas de `@page` para `@page "{id:int}"`.

Uma solicitação para a página com o modelo de rota `{id:int}` que **não** inclui um valor inteiro de rota retorna um erro HTTP 404 (não encontrado). Por exemplo, `http://localhost:5000/Students/Details` retorna um erro 404. Para tornar a ID opcional, acrescente `?` à restrição de rota:

```
@page "{id:int?}"
```

Execute o aplicativo, clique em um link Detalhes e verifique se a URL está passando a ID como dados de rota (`http://localhost:5000/Students/Details/2`).

Não altere `@page` globalmente para `@page "{id:int}"`, pois isso desfaz os links para as páginas Início e Criar.

### Adicionar dados relacionados

O código gerado por scaffolding para a página Índice de Alunos não inclui a propriedade `Enrollments`. Nesta seção, o conteúdo da coleção `Enrollments` é exibido na página Detalhes.

O método `OnGetAsync` de `Pages/Students/Details.cshtml.cs` usa o método `FirstOrDefaultAsync` para recuperar uma única entidade `Student`. Adicione o seguinte código realçado:

```
public async Task<IActionResult> OnGetAsync(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    Student = await _context.Student
        .Include(s => s.Enrollments)
            .ThenInclude(e => e.Course)
        .AsNoTracking()
        .FirstOrDefaultAsync(m => m.ID == id);

    if (Student == null)
    {
        return NotFound();
    }
    return Page();
}
```

Os métodos `Include` e `ThenInclude` fazem com que o contexto carregue a propriedade de navegação `Student.Enrollments` e, dentro de cada registro, a propriedade de navegação `Enrollment.Course`. Esses métodos são examinados em detalhes no tutorial de dados relacionado à leitura.

O método `AsNoTracking` melhora o desempenho em cenários em que as entidades retornadas não são atualizadas no contexto atual. `AsNoTracking` é abordado mais adiante neste tutorial.

### Exibir registros relacionados na página Detalhes

Abra `Pages/Students/Details.cshtml`. Adicione o seguinte código realçado para exibir uma lista de registros:

```

@page "{id:int}"
@model ContosoUniversity.Pages.Students.DetailsModel

 @{
     ViewData["Title"] = "Details";
 }

<h2>Details</h2>

<div>
    <h4>Student</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Student.LastName)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Student.LastName)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Student.FirstMidName)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Student.FirstMidName)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Student.EnrollmentDate)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Student.EnrollmentDate)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Student.Enrollments)
        </dt>
        <dd>
            <table class="table">
                <tr>
                    <th>Course Title</th>
                    <th>Grade</th>
                </tr>
                @foreach (var item in Model.Student.Enrollments)
                {
                    <tr>
                        <td>
                            @Html.DisplayFor(modelItem => item.Course.Title)
                        </td>
                        <td>
                            @Html.DisplayFor(modelItem => item.Grade)
                        </td>
                    </tr>
                }
            </table>
        </dd>
    </dl>
</div>
<div>
    <a asp-page=".Edit" asp-route-id="@Model.Student.ID">Edit</a> |
    <a asp-page=".Index">Back to List</a>
</div>

```

Se o recuo do código estiver incorreto depois que o código for colado, pressione CTRL-K-D para corrigi-lo.

O código anterior percorre as entidades na propriedade de navegação `Enrollments`. Para cada registro, ele exibe o nome do curso e a nota. O título do curso é recuperado da entidade `Course`, que é armazenada na propriedade de navegação `Course` da entidade `Enrollments`.

Execute o aplicativo, selecione a guia **Alunos** e clique no link **Detalhes** de um aluno. A lista de cursos e notas do aluno selecionado é exibida.

## Atualizar a página Criar

Atualize o método `OnPostAsync` em *Pages/Students/Create.cshtml.cs* com o seguinte código:

```
public async Task<IActionResult> OnPostAsync()
{
    if (!ModelState.IsValid)
    {
        return Page();
    }

    var emptyStudent = new Student();

    if (await TryUpdateModelAsync<Student>(
        emptyStudent,
        "student", // Prefix for form value.
        s => s.FirstMidName, s => s.LastName, s => s.EnrollmentDate))
    {
        _context.Student.Add(emptyStudent);
        await _context.SaveChangesAsync();
        return RedirectToPage("./Index");
    }

    return null;
}
```

### TryUpdateModelAsync

Examine o código `TryUpdateModelAsync`:

```
var emptyStudent = new Student();

if (await TryUpdateModelAsync<Student>(
    emptyStudent,
    "student", // Prefix for form value.
    s => s.FirstMidName, s => s.LastName, s => s.EnrollmentDate))
{
```

No código anterior, `TryUpdateModelAsync<Student>` tenta atualizar o objeto `emptyStudent` usando os valores de formulário postados da propriedade `PageContext` no `PageModel`. `TryUpdateModelAsync` atualiza apenas as propriedades listadas (`s => s.FirstMidName, s => s.LastName, s => s.EnrollmentDate`).

Na amostra anterior:

- O segundo argumento (`"student", // Prefix`) é o prefixo usado para pesquisar valores. Não diferencia maiúsculas de minúsculas.
- Os valores de formulário postados são convertidos nos tipos no modelo `Student` usando o [model binding](#).

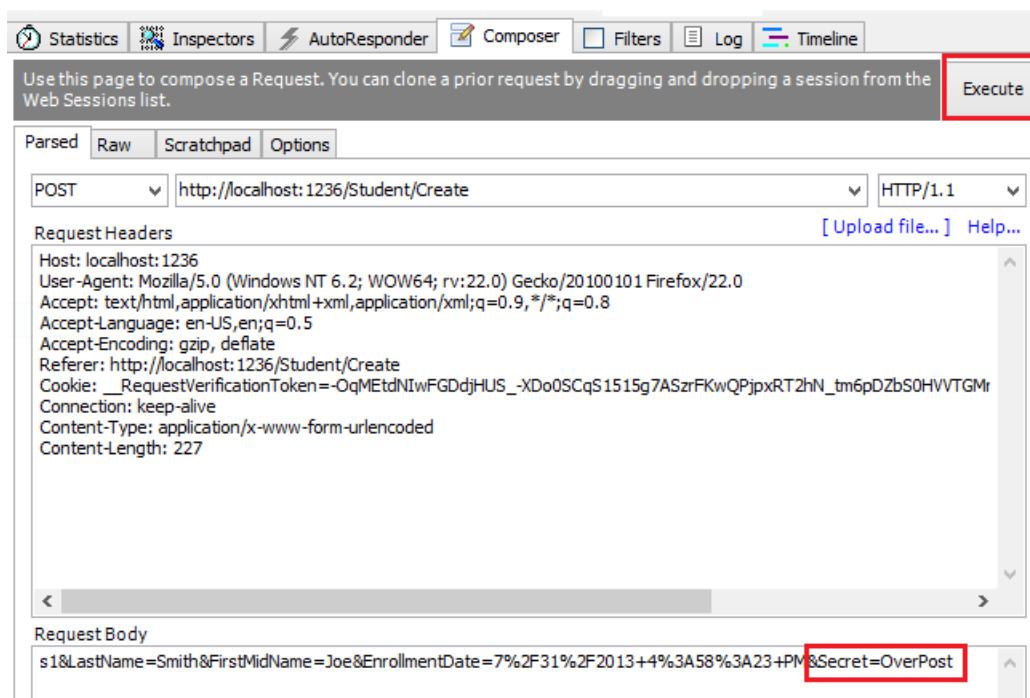
### Excesso de postagem

O uso de `TryUpdateModel` para atualizar campos com valores postados é uma melhor prática de segurança porque ele impede o excesso de postagem. Por exemplo, suponha que a entidade `Student` inclua uma propriedade `Secret` que esta página da Web não deve atualizar nem adicionar:

```
public class Student
{
    public int ID { get; set; }
    public string LastName { get; set; }
    public string FirstMidName { get; set; }
    public DateTime EnrollmentDate { get; set; }
    public string Secret { get; set; }
}
```

Mesmo que o aplicativo não tenha um campo `Secret` na Página criar/atualizar do Razor, um invasor pode definir o valor `Secret` por excesso de postagem. Um invasor pode usar uma ferramenta como o Fiddler ou escrever um JavaScript para postar um valor de formulário `Secret`. O código original não limita os campos que o associador de modelos usa quando ele cria uma instância Student.

Seja qual for o valor que o invasor especificou para o campo de formulário `Secret`, ele será atualizado no BD. A imagem a seguir mostra a ferramenta Fiddler adicionando o campo `Secret` (com o valor "OverPost") aos valores de formulário postados.



O valor "OverPost" foi adicionado com êxito à propriedade `Secret` da linha inserida. O designer do aplicativo nunca desejou que a propriedade `Secret` fosse definida com a página Criar.

## Modelo de exibição

Um modelo de exibição normalmente contém um subconjunto das propriedades incluídas no modelo usado pelo aplicativo. O modelo de aplicativo costuma ser chamado de modelo de domínio. O modelo de domínio normalmente contém todas as propriedades necessárias para a entidade correspondente no BD. O modelo de exibição contém apenas as propriedades necessárias para a camada de interface do usuário (por exemplo, a página Criar). Além do modelo de exibição, alguns aplicativos usam um modelo de associação ou modelo de entrada para passar dados entre a classe de modelo de página das Páginas do Razor e o navegador. Considere o seguinte modelo de exibição `Student`:

```
using System;

namespace ContosoUniversity.Models
{
    public class StudentVM
    {
        public int ID { get; set; }
        public string LastName { get; set; }
        public string FirstMidName { get; set; }
        public DateTime EnrollmentDate { get; set; }
    }
}
```

Os modelos de exibição fornecem uma maneira alternativa para impedir o excesso de postagem. O modelo de exibição contém apenas as propriedades a serem exibidas ou atualizadas.

O seguinte código usa o modelo de exibição `StudentVM` para criar um novo aluno:

```
[BindProperty]
public StudentVM StudentVM { get; set; }

public async Task<IActionResult> OnPostAsync()
{
    if (!ModelState.IsValid)
    {
        return Page();
    }

    var entry = _context.Add(new Student());
    entry.CurrentValues.SetValues(StudentVM);
    await _context.SaveChangesAsync();
    return RedirectToPage("./Index");
}
```

O método `SetValues` define os valores desse objeto lendo os valores de outro objeto `PropertyValues`. `SetValues` usa a correspondência de nomes de propriedade. O tipo de modelo de exibição não precisa estar relacionado ao tipo de modelo, apenas precisa ter as propriedades correspondentes.

O uso de `StudentVM` exige a atualização de `CreateVM.cshtml` para usar `StudentVM` em vez de `Student`.

Nas Páginas do Razor, a classe derivada `PageModel` é o modelo de exibição.

## Atualizar a página Editar

Atualize o modelo de página para a página de edição. As alterações principais são realçadas:

```

public class EditModel : PageModel
{
    private readonly SchoolContext _context;

    public EditModel(SchoolContext context)
    {
        _context = context;
    }

    [BindProperty]
    public Student Student { get; set; }

    public async Task<IActionResult> OnGetAsync(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        Student = await _context.Student.FindAsync(id);

        if (Student == null)
        {
            return NotFound();
        }
        return Page();
    }

    public async Task<IActionResult> OnPostAsync(int? id)
    {
        if (!ModelState.IsValid)
        {
            return Page();
        }

        var studentToUpdate = await _context.Student.FindAsync(id);

        if (await TryUpdateModelAsync<Student>(
            studentToUpdate,
            "student",
            s => s.FirstMidName, s => s.LastName, s => s.EnrollmentDate))
        {
            await _context.SaveChangesAsync();
            return RedirectToPage("./Index");
        }

        return Page();
    }
}

```

As alterações de código são semelhantes à página Criar, com algumas exceções:

- `OnPostAsync` tem um parâmetro `id` opcional.
- O aluno atual é buscado do BD, em vez de criar um aluno vazio.
- `FirstOrDefaultAsync` foi substituído por `FindAsync`. `FindAsync` é uma boa opção ao selecionar uma entidade da chave primária. Consulte [FindAsync](#) para obter mais informações.

### Testar as páginas Editar e Criar

Crie e edite algumas entidades de aluno.

## Estados da entidade

O contexto de BD controla se as entidades em memória estão em sincronia com suas linhas correspondentes no

BD. As informações de sincronização de contexto do BD determinam o que acontece quando `SaveChangesAsync` é chamado. Por exemplo, quando uma nova entidade é passada para o método `AddAsync`, o estado da entidade é definido como `Added`. Quando `SaveChangesAsync` é chamado, o contexto de BD emite um comando SQL INSERT.

Uma entidade pode estar em um dos [seguintes estados](#):

- `Added`: a entidade ainda não existe no BD. O método `SaveChanges` emite uma instrução INSERT.
- `Unchanged`: nenhuma alteração precisa ser salva com essa entidade. Uma entidade tem esse status quando é lida do BD.
- `Modified`: alguns ou todos os valores de propriedade da entidade foram modificados. O método `SaveChanges` emite uma instrução UPDATE.
- `Deleted`: a entidade foi marcada para exclusão. O método `SaveChanges` emite uma instrução DELETE.
- `Detached`: a entidade não está sendo controlada pelo contexto de BD.

Em um aplicativo da área de trabalho, em geral, as alterações de estado são definidas automaticamente. Uma entidade é lida, as alterações são feitas e o estado da entidade é alterado automaticamente para `Modified`. A chamada a `SaveChanges` gera uma instrução SQL UPDATE que atualiza apenas as propriedades alteradas.

Em um aplicativo Web, o `DbContext` que lê uma entidade e exibe os dados é descartado depois que uma página é renderizada. Quando o método `OnPostAsync` de uma página é chamado, é feita uma nova solicitação da Web e com uma nova instância do `DbContext`. A nova leitura da entidade nesse novo contexto simula o processamento da área de trabalho.

## Atualizar a página Excluir

Nesta seção, o código é adicionado para implementar uma mensagem de erro personalizada quando há falha na chamada a `SaveChanges`. Adicione uma cadeia de caracteres para que ela contenha as possíveis mensagens de erro:

```
public class DeleteModel : PageModel
{
    private readonly SchoolContext _context;

    public DeleteModel(SchoolContext context)
    {
        _context = context;
    }

    [BindProperty]
    public Student Student { get; set; }
    public string ErrorMessage { get; set; }
```

Substitua o método `OnGetAsync` pelo seguinte código:

```

public async Task<IActionResult> OnGetAsync(int? id, bool? saveChangesError = false)
{
    if (id == null)
    {
        return NotFound();
    }

    Student = await _context.Student
        .AsNoTracking()
        .FirstOrDefaultAsync(m => m.ID == id);

    if (Student == null)
    {
        return NotFound();
    }

    if (saveChangesError.GetValueOrDefault())
    {
        ErrorMessage = "Delete failed. Try again";
    }

    return Page();
}

```

O código anterior contém o parâmetro opcional `saveChangesError`. `saveChangesError` indica se o método foi chamado após uma falha ao excluir o objeto de aluno. A operação de exclusão pode falhar devido a problemas de rede temporários. Erros de rede transitórios serão mais prováveis de ocorrerem na nuvem. `saveChangesError` é falso quando a página Excluir `OnGetAsync` é chamada na interface do usuário. Quando `OnGetAsync` é chamado por `OnPostAsync` (devido à falha da operação de exclusão), o parâmetro `saveChangesError` é verdadeiro.

### O método `OnPostAsync` nas páginas Excluir

Substitua `OnPostAsync` pelo seguinte código:

```

public async Task<IActionResult> OnPostAsync(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var student = await _context.Student
        .AsNoTracking()
        .FirstOrDefaultAsync(m => m.ID == id);

    if (student == null)
    {
        return NotFound();
    }

    try
    {
        _context.Student.Remove(student);
        await _context.SaveChangesAsync();
        return RedirectToPage("./Index");
    }
    catch (DbUpdateException /* ex */)
    {
        //Log the error (uncomment ex variable name and write a log.)
        return RedirectToAction("./Delete",
            new { id, saveChangesError = true });
    }
}

```

O código anterior recupera a entidade selecionada e, em seguida, chama o método `Remove` para definir o status da entidade como `Deleted`. Quando `SaveChanges` é chamado, um comando SQL DELETE é gerado. Se `Remove` falhar:

- A exceção de BD é capturada.
- O método `OnGetAsync` das páginas Excluir é chamado com `saveChangesError=true`.

## Atualizar a Página Excluir do Razor

Adicione a mensagem de erro realçada a seguir à Página Excluir do Razor.

```
@page "{id:int}"
@model ContosoUniversity.Pages.Students.DeleteModel

 @{
    ViewData["Title"] = "Delete";
}

<h2>Delete</h2>

<p class="text-danger">@Model.ErrorMessage</p>

<h3>Are you sure you want to delete this?</h3>
<div>
```

Exclusão de teste.

## Erros comuns

Alunos/Índice ou outros links não funcionam:

Verifique se a Página do Razor contém a diretiva `@page` correta. Por exemplo, a Razor Page Alunos/Índice **não** deve conter um modelo de rota:

```
@page "{id:int}"
```

Cada Página do Razor deve incluir a diretiva `@page`.

[ANTERIOR](#)

[PRÓXIMO](#)

# Páginas Razor com o EF Core no ASP.NET Core – Classificação, filtro, paginação – 3 de 8

30/01/2019 • 25 minutes to read • [Edit Online](#)

A versão deste tutorial para o ASP.NET Core 2.0 pode ser encontrada neste [arquivo PDF](#).

A versão deste tutorial para o ASP.NET Core 2.1 contém diversas melhorias em relação à versão 2.0.

Por [Tom Dykstra](#), [Rick Anderson](#) e [Jon P Smith](#)

O aplicativo Web Contoso University demonstra como criar aplicativos Web das Páginas do Razor usando o EF Core e o Visual Studio. Para obter informações sobre a série de tutoriais, consulte [o primeiro tutorial](#).

Neste tutorial, as funcionalidades de classificação, filtragem, agrupamento e paginação são adicionadas.

A ilustração a seguir mostra uma página concluída. Os títulos de coluna são links clicáveis para classificar a coluna. Clicar em um título de coluna alterna repetidamente entre a ordem de classificação ascendente e descendente.

Last Name	First Name	Enrollment Date	
Alexander	Carson	9/1/2005 12:00:00 AM	Edit   Details   Delete
Alonso	Meredith	9/1/2002 12:00:00 AM	Edit   Details   Delete
Anand	Arturo	9/1/2003 12:00:00 AM	Edit   Details   Delete

Caso tenha problemas que não consiga resolver, baixe o [aplicativo concluído](#).

## Adicionar uma classificação à página Índice

Adicione cadeias de caracteres ao `PageModel` de `Students/Index.cshtml.cs` para que ele contenha os parâmetros de classificação:

```

public class IndexModel : PageModel
{
    private readonly SchoolContext _context;

    public IndexModel(SchoolContext context)
    {
        _context = context;
    }

    public string NameSort { get; set; }
    public string DateSort { get; set; }
    public string CurrentFilter { get; set; }
    public string CurrentSort { get; set; }

```

Atualize `OnGetAsync` de *Students/Index.cshtml.cs* com o seguinte código:

```

public async Task OnGetAsync(string sortOrder)
{
    NameSort = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
    DateSort = sortOrder == "Date" ? "date_desc" : "Date";

    IQueryable<Student> studentIQ = from s in _context.Student
                                      select s;

    switch (sortOrder)
    {
        case "name_desc":
            studentIQ = studentIQ.OrderByDescending(s => s.LastName);
            break;
        case "Date":
            studentIQ = studentIQ.OrderBy(s => s.EnrollmentDate);
            break;
        case "date_desc":
            studentIQ = studentIQ.OrderByDescending(s => s.EnrollmentDate);
            break;
        default:
            studentIQ = studentIQ.OrderBy(s => s.LastName);
            break;
    }

    Student = await studentIQ.AsNoTracking().ToListAsync();
}

```

O código anterior recebe um parâmetro `sortOrder` da cadeia de caracteres de consulta na URL. A URL (incluindo a cadeia de caracteres de consulta) é gerada pelo [Auxiliar de Marcação de Âncora](#)

O parâmetro `sortOrder` é "Name" ou "Data". O parâmetro `sortOrder` é opcionalmente seguido de "\_desc" para especificar a ordem descendente. A ordem de classificação crescente é padrão.

Quando a página Índice é solicitada do link **Alunos**, não há nenhuma cadeia de caracteres de consulta. Os alunos são exibidos em ordem ascendente por sobrenome. A ordem ascendente por sobrenome é o padrão (caso fall-through) na instrução `switch`. Quando o usuário clica em um link de título de coluna, o valor `sortOrder` apropriado é fornecido no valor de cadeia de caracteres de consulta.

`NameSort` e `DateSort` são usados pela Página do Razor para configurar os hiperlinks de título de coluna com os valores de cadeia de caracteres de consulta apropriados:

```

public async Task OnGetAsync(string sortOrder)
{
    NameSort = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
    DateSort = sortOrder == "Date" ? "date_desc" : "Date";

    IQueryable<Student> studentIQ = from s in _context.Student
                                      select s;

    switch (sortOrder)
    {
        case "name_desc":
            studentIQ = studentIQ.OrderByDescending(s => s.LastName);
            break;
        case "Date":
            studentIQ = studentIQ.OrderBy(s => s.EnrollmentDate);
            break;
        case "date_desc":
            studentIQ = studentIQ.OrderByDescending(s => s.EnrollmentDate);
            break;
        default:
            studentIQ = studentIQ.OrderBy(s => s.LastName);
            break;
    }

    Student = await studentIQ.AsNoTracking().ToListAsync();
}

```

O seguinte código contém o [operador ?: condicional do C#](#):

```

NameSort = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
DateSort = sortOrder == "Date" ? "date_desc" : "Date";

```

A primeira linha especifica que, quando `sortOrder` é nulo ou vazio, `NameSort` é definido como "name\_desc". Se `sortOrder` **não** é nulo nem vazio, `NameSort` é definido como uma cadeia de caracteres vazia.

O `?: operator` também é conhecido como o operador ternário.

Essas duas instruções permitem que a página defina os hiperlinks de título de coluna da seguinte maneira:

ORDEM DE CLASSIFICAÇÃO ATUAL	HIPERLINK DO SOBRENOME	HIPERLINK DE DATA
Sobrenome ascendente	descending	ascending
Sobrenome descendente	ascending	ascending
Data ascendente	ascending	descending
Data descendente	ascending	ascending

O método usa o LINQ to Entities para especificar a coluna pela qual classificar. O código inicializa um `IQueryable<Student>` antes da instrução `switch` e modifica-o na instrução `switch`:

```

public async Task OnGetAsync(string sortOrder)
{
    NameSort = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
    DateSort = sortOrder == "Date" ? "date_desc" : "Date";

    IQueryable<Student> studentIQ = from s in _context.Student
                                      select s;

    switch (sortOrder)
    {
        case "name_desc":
            studentIQ = studentIQ.OrderByDescending(s => s.LastName);
            break;
        case "Date":
            studentIQ = studentIQ.OrderBy(s => s.EnrollmentDate);
            break;
        case "date_desc":
            studentIQ = studentIQ.OrderByDescending(s => s.EnrollmentDate);
            break;
        default:
            studentIQ = studentIQ.OrderBy(s => s.LastName);
            break;
    }

    Student = await studentIQ.AsNoTracking().ToListAsync();
}

```

Quando um `IQueryable` é criado ou modificado, nenhuma consulta é enviada ao banco de dados. A consulta não é executada até que o objeto `IQueryable` seja convertido em uma coleção. `IQueryable` são convertidos em uma coleção com uma chamada a um método como `ToListAsync`. Portanto, o código `IQueryable` resulta em uma única consulta que não é executada até que a seguinte instrução:

```
Student = await studentIQ.AsNoTracking().ToListAsync();
```

`OnGetAsync` pode ficar detalhado com um grande número de colunas classificáveis.

### Adicionar hiperlinks de título de coluna à página Student Index

Substitua o código em *Students/Index.cshtml*, pelo seguinte código realçado:

```

@page
@model ContosoUniversity.Pages.Students.IndexModel

 @{
     ViewData["Title"] = "Index";
 }

<h2>Index</h2>
<p>
    <a asp-page="Create">Create New</a>
</p>

<table class="table">
    <thead>
        <tr>
            <th>
                <a asp-page="./Index" asp-route-sortOrder="@Model.NameSort">
                    @Html.DisplayNameFor(model => model.Student[0].LastName)
                </a>
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Student[0].FirstMidName)
            </th>
            <th>
                <a asp-page="./Index" asp-route-sortOrder="@Model.DateSort">
                    @Html.DisplayNameFor(model => model.Student[0].EnrollmentDate)
                </a>
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model.Student)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.LastName)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.FirstMidName)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.EnrollmentDate)
                </td>
                <td>
                    <a asp-page="./Edit" asp-route-id="@item.ID">Edit</a> |
                    <a asp-page="./Details" asp-route-id="@item.ID">Details</a> |
                    <a asp-page=".Delete" asp-route-id="@item.ID">Delete</a>
                </td>
            </tr>
        }
    </tbody>
</table>

```

O código anterior:

- Adiciona hiperlinks aos títulos de coluna `LastName` e `EnrollmentDate`.
- Usa as informações em `NameSort` e `DateSort` para configurar hiperlinks com os valores de ordem de classificação atuais.

Para verificar se a classificação funciona:

- Execute o aplicativo e selecione a guia **Alunos**.
- Clique em **Sobrenome**.

- Clique em **Data de Registro**.

Para obter um melhor entendimento do código:

- Em *Students/Index.cshtml.cs*, defina um ponto de interrupção em `switch (sortOrder)`.
- Adicione uma inspeção para `NameSort` e `DateSort`.
- Em *Students/Index.cshtml*, defina um ponto de interrupção em  
`@Html.DisplayNameFor(model => model.Student[0].LastName)`.

Execute o depurador em etapas.

## Adicionar uma Caixa de Pesquisa à página Índice de Alunos

Para adicionar a filtragem à página Índice de Alunos:

- Uma caixa de texto e um botão Enviar são adicionados à Página do Razor. A caixa de texto fornece uma cadeia de caracteres de pesquisa no nome ou sobrenome.
- O modelo de página é atualizado para usar o valor da caixa de texto.

### Adicionar a funcionalidade de filtragem a método Index

Atualize `OnGetAsync` de *Students/Index.cshtml.cs* com o seguinte código:

```
public async Task OnGetAsync(string sortOrder, string searchString)
{
    NameSort = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
    DateSort = sortOrder == "Date" ? "date_desc" : "Date";
    CurrentFilter = searchString;

    IQueryable<Student> studentIQ = from s in _context.Student
                                         select s;
    if (!String.IsNullOrEmpty(searchString))
    {
        studentIQ = studentIQ.Where(s => s.LastName.Contains(searchString)
                               || s.FirstMidName.Contains(searchString));
    }

    switch (sortOrder)
    {
        case "name_desc":
            studentIQ = studentIQ.OrderByDescending(s => s.LastName);
            break;
        case "Date":
            studentIQ = studentIQ.OrderBy(s => s.EnrollmentDate);
            break;
        case "date_desc":
            studentIQ = studentIQ.OrderByDescending(s => s.EnrollmentDate);
            break;
        default:
            studentIQ = studentIQ.OrderBy(s => s.LastName);
            break;
    }

    Student = await studentIQ.AsNoTracking().ToListAsync();
}
```

O código anterior:

- Adiciona o parâmetro `searchString` ao método `OnGetAsync`. O valor de cadeia de caracteres de pesquisa é recebido de uma caixa de texto que é adicionada na próxima seção.
- Adicionou uma cláusula `Where` à instrução LINQ. A cláusula `Where` seleciona somente os alunos cujo nome ou sobrenome contém a cadeia de caracteres de pesquisa. A instrução LINQ é executada somente se há um

valor a ser pesquisado.

Observação: o código anterior chama o método `Where` em um objeto `IQueryable`, e o filtro é processado no servidor. Em alguns cenários, o aplicativo pode chamar o método `Where` como um método de extensão em uma coleção em memória. Por exemplo, suponha que `_context.Students` seja alterado do `DbSet` do EF Core para um método de repositório que retorna uma coleção `IEnumerable`. O resultado normalmente é o mesmo, mas em alguns casos pode ser diferente.

Por exemplo, a implementação do .NET Framework do `Contains` executa uma comparação diferencia maiúsculas de minúsculas por padrão. No SQL Server, a diferenciação de maiúsculas e minúsculas de `Contains` é determinada pela configuração de ordenação da instância do SQL Server. O SQL Server usa como padrão a não diferenciação de maiúsculas e minúsculas. `ToUpper` pode ser chamado para fazer com que o teste diferencie maiúsculas de minúsculas de forma explícita:

```
Where(s => s.LastName.ToUpper().Contains(searchString.ToUpper()))
```

O código anterior garantirá que os resultados diferenciem maiúsculas de minúsculas se o código for alterado para usar `IEnumerable`. Quando `Contains` é chamado em uma coleção `IEnumerable`, a implementação do .NET Core é usada. Quando `Contains` é chamado em um objeto `IQueryable`, a implementação do banco de dados é usada. O retorno de um `IEnumerable` de um repositório pode ter uma penalidade significativa de desempenho:

1. Todas as linhas são retornadas do servidor de BD.
2. O filtro é aplicado a todas as linhas retornadas no aplicativo.

Há uma penalidade de desempenho por chamar `ToUpper`. O código `ToUpper` adiciona uma função à cláusula WHERE da instrução TSQL SELECT. A função adicionada impede que o otimizador use um índice. Considerando que o SQL é instalado como diferenciando maiúsculas de minúsculas, é melhor evitar a chamada `ToUpper` quando ela não for necessária.

## Adicionar uma Caixa de Pesquisa à página Student Index

Em `Pages/Students/Index.cshtml`, adicione o código realçado a seguir para criar um botão **Pesquisar** e o cromado variado.

```
@page
@model ContosoUniversity.Pages.Students.IndexModel

 @{
    ViewData["Title"] = "Index";
}

<h2>Index</h2>

<p>
    <a asp-page="Create">Create New</a>
</p>

<form asp-page="./Index" method="get">
    <div class="form-actions no-color">
        <p>
            Find by name:
            <input type="text" name="SearchString" value="@Model.CurrentFilter" />
            <input type="submit" value="Search" class="btn btn-default" /> |
            <a asp-page="./Index">Back to full List</a>
        </p>
    </div>
</form>

<table class="table">
```

O código anterior usa o auxiliar de marcação `<form>` para adicionar o botão e a caixa de texto de pesquisa. Por padrão, o auxiliar de marcação `<form>` envia dados de formulário com um POST. Com o POST, os parâmetros são passados no corpo da mensagem HTTP e não na URL. Quando o HTTP GET é usado, os dados de formulário são passados na URL como cadeias de consulta. Passar os dados com cadeias de consulta permite aos usuários marcar a URL. As [diretrizes do W3C](#) recomendam o uso de GET quando a ação não resulta em uma atualização.

Teste o aplicativo:

- Selecione a guia **Alunos** e insira uma cadeia de caracteres de pesquisa.
- Selecione **Pesquisar**.

Observe que a URL contém a cadeia de caracteres de pesquisa.

```
http://localhost:5000/Students?SearchString=an
```

Se a página estiver marcada, o indicador conterá a URL para a página e a cadeia de caracteres de consulta `SearchString`. O `method="get"` na marcação `form` é o que fez com que a cadeia de caracteres de consulta fosse gerada.

Atualmente, quando um link de classificação de título de coluna é selecionado, o valor de filtro da caixa **Pesquisa** é perdido. O valor de filtro perdido é corrigido na próxima seção.

## Adicionar a funcionalidade de paginação à página Índice de Alunos

Nesta seção, uma classe `PaginatedList` é criada para dar suporte à paginação. A classe `PaginatedList` usa as instruções `Skip` e `Take` para filtrar dados no servidor em vez de recuperar todas as linhas da tabela. A ilustração a seguir mostra os botões de paginação.

The screenshot shows a web browser window with the title "Index - Contoso University". The address bar shows "localhost:5813/Student". The main content area has a header "Contoso University". Below it, the word "Index" is displayed. There is a "Create New" link. A search bar with the placeholder "Find by name:" and a "Search" button is present. Below the search bar is a table with three columns: "Last Name", "First Name", and "Enrollment Date". Three rows of data are shown: Alexander Carson (9/1/2005), Alonso Meredith (9/1/2002), and Anand Arturo (9/1/2003). Each row has "Edit | Details | Delete" links. At the bottom of the table are "Previous" and "Next" navigation buttons.

Last Name	First Name	Enrollment Date	
Alexander	Carson	9/1/2005 12:00:00 AM	Edit   Details   Delete
Alonso	Meredith	9/1/2002 12:00:00 AM	Edit   Details   Delete
Anand	Arturo	9/1/2003 12:00:00 AM	Edit   Details   Delete

Na pasta do projeto, crie `PaginatedList.cs` com o seguinte código:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;

namespace ContosoUniversity
{
    public class PaginatedList<T> : List<T>
    {
        public int PageIndex { get; private set; }
        public int TotalPages { get; private set; }

        public PaginatedList(List<T> items, int count, int pageIndex, int pageSize)
        {
            PageIndex = pageIndex;
            TotalPages = (int)Math.Ceiling(count / (double)pageSize);

            this.AddRange(items);
        }

        public bool HasPreviousPage
        {
            get
            {
                return (PageIndex > 1);
            }
        }

        public bool HasNextPage
        {
            get
            {
                return (PageIndex < TotalPages);
            }
        }

        public static async Task<PaginatedList<T>> CreateAsync(
            IQueryable<T> source, int pageIndex, int pageSize)
        {
            var count = await source.CountAsync();
            var items = await source.Skip(
                (pageIndex - 1) * pageSize)
                .Take(pageSize).ToListAsync();
            return new PaginatedList<T>(items, count, pageIndex, pageSize);
        }
    }
}

```

O método `CreateAsync` no código anterior usa o tamanho da página e o número da página e aplica as instruções `Skip` e `Take` ao `IQueryable`. Quando `ToToListAsync` é chamado no `IQueryable`, ele retorna uma Lista que contém somente a página solicitada. As propriedades `HasPreviousPage` e `HasNextPage` são usadas para habilitar ou desabilitar os botões de paginação **Anterior** e **Próximo**.

O método `CreateAsync` é usado para criar o `PaginatedList<T>`. Um construtor não pode criar o objeto `PaginatedList<T>`; construtores não podem executar um código assíncrono.

## Adicionar a funcionalidade de paginação ao método Index

Em `Students/Index.cshtml.cs`, atualize o tipo de `Student` em `IList<Student>` para `PaginatedList<Student>`:

```
public PaginatedList<Student> Student { get; set; }
```

Atualize `OnGetAsync` de `Students/Index.cshtml.cs` com o seguinte código:

```
public async Task OnGetAsync(string sortOrder,
    string currentFilter, string searchString, int? pageIndex)
{
    CurrentSort = sortOrder;
    NameSort = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
    DateSort = sortOrder == "Date" ? "date_desc" : "Date";
    if (searchString != null)
    {
        pageIndex = 1;
    }
    else
    {
        searchString = currentFilter;
    }

    CurrentFilter = searchString;

    IQueryable<Student> studentIQ = from s in _context.Student
                                         select s;
    if (!String.IsNullOrEmpty(searchString))
    {
        studentIQ = studentIQ.Where(s => s.LastName.Contains(searchString)
                                || s.FirstMidName.Contains(searchString));
    }
    switch (sortOrder)
    {
        case "name_desc":
            studentIQ = studentIQ.OrderByDescending(s => s.LastName);
            break;
        case "Date":
            studentIQ = studentIQ.OrderBy(s => s.EnrollmentDate);
            break;
        case "date_desc":
            studentIQ = studentIQ.OrderByDescending(s => s.EnrollmentDate);
            break;
        default:
            studentIQ = studentIQ.OrderBy(s => s.LastName);
            break;
    }

    int pageSize = 3;
    Student = await PaginatedList<Student>.CreateAsync(
        studentIQ.AsNoTracking(), pageIndex ?? 1, pageSize);
}
```

O código anterior adiciona o índice de página, o `sortOrder` atual e o `currentFilter` à assinatura do método.

```
public async Task OnGetAsync(string sortOrder,
    string currentFilter, string searchString, int? pageIndex)
```

Todos os parâmetros são nulos quando:

- A página é chamada no link **Alunos**.
- O usuário ainda não clicou em um link de paginação ou classificação.

Quando um link de paginação recebe um clique, a variável de índice de páginas contém o número da página a ser exibido.

`CurrentSort` fornece à Página do Razor a ordem de classificação atual. A ordem de classificação atual precisa ser incluída nos links de paginação para que a ordem de classificação seja mantida durante a paginação.

`CurrentFilter` fornece à Página do Razor a cadeia de caracteres de filtro atual. O valor `CurrentFilter`:

- Deve ser incluído nos links de paginação para que as configurações de filtro sejam mantidas durante a paginação.
- Deve ser restaurado para a caixa de texto quando a página é exibida novamente.

Se a cadeia de caracteres de pesquisa é alterada durante a paginação, a página é redefinida como 1. A página precisa ser redefinida como 1, porque o novo filtro pode resultar na exibição de dados diferentes. Quando um valor de pesquisa é inserido e **Enviar** é selecionado:

- A cadeia de caracteres de pesquisa foi alterada.
- O parâmetro `searchString` não é nulo.

```
if (searchString != null)
{
    pageIndex = 1;
}
else
{
    searchString = currentFilter;
}
```

O método `PaginatedList.CreateAsync` converte a consulta de alunos em uma única página de alunos de um tipo de coleção compatível com paginação. Essa única página de alunos é passada para a Página do Razor.

```
Student = await PaginatedList<Student>.CreateAsync(
    studentIQ.AsNoTracking(), pageIndex ?? 1, pageSize);
```

Os dois pontos de interrogação em `PaginatedList.CreateAsync` representam o [operador de união de nulo](#). O operador de união de nulo define um valor padrão para um tipo que permite valor nulo. A expressão `(pageIndex ?? 1)` significará retornar o valor de `pageIndex` se ele tiver um valor. Se `pageIndex` não tiver um valor, 1 será retornado.

## Adicionar links de paginação à Página do Razor do aluno

Atualize a marcação em `Students/Index.cshtml`. As alterações são realçadas:

```
@page
@model ContosoUniversity.Pages.Students.IndexModel

 @{
     ViewData["Title"] = "Index";
 }

<h2>Index</h2>

<p>
    <a asp-page="Create">Create New</a>
</p>

<form asp-page=".Index" method="get">
    <div class="form-actions no-color">
        <p>
            Find by name: <input type="text" name="SearchString" value="@Model.CurrentFilter" />
            <input type="submit" value="Search" class="btn btn-default" /> |
            <a asp-page=".Index">Back to full List</a>
        </p>
    </div>
</form>
```

```





```

Os links de cabeçalho de coluna usam a cadeia de caracteres de consulta para passar a cadeia de caracteres de pesquisa atual para o método `OnGetAsync`, de modo que o usuário possa classificar nos resultados do filtro:

```

<a asp-page=".~/Index" asp-route-sortOrder="@Model.NameSort"
    asp-route-currentFilter="@Model.CurrentFilter">
    @Html.DisplayNameFor(model => model.Student[0].LastName)
</a>

```

Os botões de paginação são exibidos por auxiliares de marcação:

```

<a asp-page=".~/Index"
    asp-route-sortOrder="@Model.CurrentSort"
    asp-route-pageIndex="@((Model.StudentPageIndex - 1))"
    asp-route-currentFilter="@Model.CurrentFilter"
    class="btn btn-default @prevDisabled">
    Previous
</a>
<a asp-page=".~/Index"
    asp-route-sortOrder="@Model.CurrentSort"
    asp-route-pageIndex="@((Model.StudentPageIndex + 1))"
    asp-route-currentFilter="@Model.CurrentFilter"
    class="btn btn-default @nextDisabled">
    Next
</a>

```

Execute o aplicativo e navegue para a página de alunos.

- Para verificar se a paginação funciona, clique nos links de paginação em ordens de classificação diferentes.
- Para verificar se a paginação funciona corretamente com a classificação e filtragem, insira uma cadeia de caracteres de pesquisa e tente fazer a paginação.

Last Name	First Name	Enrollment Date	
Alexander	Carson	9/1/2005 12:00:00 AM	Edit   Details   Delete
Alonso	Meredith	9/1/2002 12:00:00 AM	Edit   Details   Delete
Anand	Arturo	9/1/2003 12:00:00 AM	Edit   Details   Delete

Para obter um melhor entendimento do código:

- Em `Students/Index.cshtml.cs`, defina um ponto de interrupção em `switch (sortOrder)`.
- Adicione uma inspeção para `NameSort`, `DateSort`, `CurrentSort` e `Model.StudentPageIndex`.
- Em `Students/Index.cshtml`, defina um ponto de interrupção em

```
@Html.DisplayNameFor(model => model.Student[0].LastName).
```

Execute o depurador em etapas.

## Atualizar a página Sobre para mostras estatísticas de alunos

Nesta etapa, *Pages/About.cshtml* é atualizada para exibir quantos alunos se registraram para cada data de registro. A atualização usa o agrupamento e inclui as seguintes etapas:

- Criar um modelo de exibição para os dados usados pela página **Sobre**.
- Atualizar a página Sobre para usar o modelo de exibição.

### Criar o modelo de exibição

Crie uma pasta *SchoolViewModels* na pasta *Models*.

Na pasta *SchoolViewModels*, adicione um *EnrollmentDateGroup.cs* com o seguinte código:

```
using System;
using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models.SchoolViewModels
{
    public class EnrollmentDateGroup
    {
        [DataType(DataType.Date)]
        public DateTime? EnrollmentDate { get; set; }

        public int StudentCount { get; set; }
    }
}
```

### Atualizar o modelo da página Sobre

Os modelos da Web no ASP.NET Core 2.2 não incluem a página Sobre. Se estiver usando o ASP.NET Core 2.2, crie a página Sobre o Razor.

Atualize o arquivo *Pages/About.cshtml.cs* com o seguinte código:

```

using ContosoUniversity.Models.SchoolViewModels;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using ContosoUniversity.Models;

namespace ContosoUniversity.Pages
{
    public class AboutModel : PageModel
    {
        private readonly SchoolContext _context;

        public AboutModel(SchoolContext context)
        {
            _context = context;
        }

        public IList<EnrollmentDateGroup> Student { get; set; }

        public async Task OnGetAsync()
        {
            IQueryable<EnrollmentDateGroup> data =
                from student in _context.Student
                group student by student.EnrollmentDate into dateGroup
                select new EnrollmentDateGroup()
                {
                    EnrollmentDate = dateGroup.Key,
                    StudentCount = dateGroup.Count()
                };

            Student = await data.AsNoTracking().ToListAsync();
        }
    }
}

```

A instrução LINQ agrupa as entidades de alunos por data de registro, calcula o número de entidades em cada grupo e armazena os resultados em uma coleção de objetos de modelo de exibição `EnrollmentDateGroup`.

## Modificar a Página Sobre do Razor

Substitua o código no arquivo `Pages/About.cshtml` pelo seguinte código:

```

@page
@model ContosoUniversity.Pages.AboutModel

 @{
     ViewData["Title"] = "Student Body Statistics";
 }

<h2>Student Body Statistics</h2>

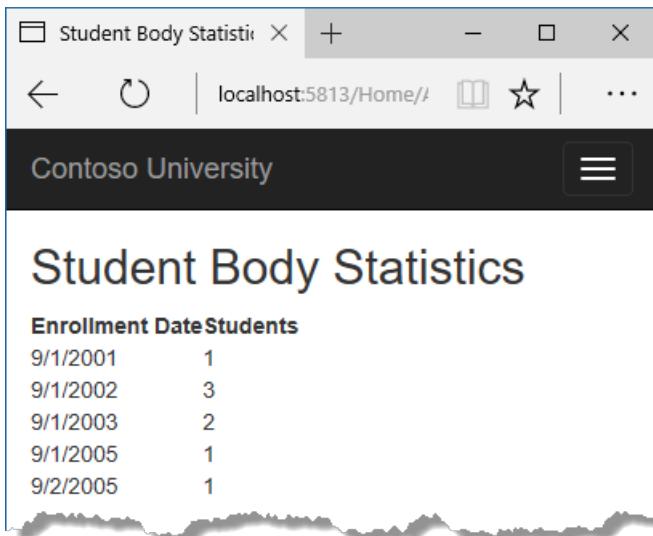


| Enrollment Date | Students |
|-----------------|----------|
|-----------------|----------|


```

Execute o aplicativo e navegue para a página Sobre. A contagem de alunos para cada data de registro é exibida em uma tabela.

Caso tenha problemas que não consiga resolver, baixe o [aplicativo concluído para este estágio](#).



## Recursos adicionais

- [Depuração de origem do ASP.NET Core 2.x](#)

No próximo tutorial, o aplicativo usa migrações para atualizar o modelo de dados.



# Páginas Razor com o EF Core no ASP.NET Core – Migrações – 4 de 8

16/07/2018 • 10 minutes to read • [Edit Online](#)

A versão deste tutorial para o ASP.NET Core 2.0 pode ser encontrada neste [arquivo PDF](#).

A versão deste tutorial para o ASP.NET Core 2.1 contém diversas melhorias em relação à versão 2.0.

Por [Tom Dykstra](#), [Jon P Smith](#) e [Rick Anderson](#)

O aplicativo Web Contoso University demonstra como criar aplicativos Web das Páginas do Razor usando o EF Core e o Visual Studio. Para obter informações sobre a série de tutoriais, consulte [o primeiro tutorial](#).

Neste tutorial, o recurso de migrações do EF Core para o gerenciamento de alterações do modelo de dados é usado.

Caso tenha problemas que não consiga resolver, baixe o [aplicativo concluído](#).

Quando um novo aplicativo é desenvolvido, o modelo de dados é alterado com frequência. Sempre que o modelo é alterado, ele fica fora de sincronia com o banco de dados. Este tutorial começa configurando o Entity Framework para criar o banco de dados, caso ele não exista. Sempre que o modelo de dados é alterado:

- O BD é removido.
- O EF cria um novo que corresponde ao modelo.
- O aplicativo propaga o BD com os dados de teste.

Essa abordagem para manter o BD em sincronia com o modelo de dados funciona bem até que você implante o aplicativo em produção. Quando o aplicativo é executado em produção, normalmente, ele armazena dados que precisam ser mantidos. O aplicativo não pode começar com um BD de teste sempre que uma alteração é feita (como a adição de uma nova coluna). O recurso Migrações do EF Core resolve esse problema, permitindo que o EF Core atualize o esquema de BD em vez de criar um novo BD.

Em vez de remover e recriar o BD quando o modelo de dados é alterado, as migrações atualizam o esquema e retêm os dados existentes.

## Remover o banco de dados

Use o **SSOX** (Pesquisador de Objetos do SQL Server) ou o comando `database drop`:

- [Visual Studio](#)
- [CLI do .NET Core](#)

No **PMC** (Console do Gerenciador de Pacotes), execute o seguinte comando:

```
Drop-Database
```

Execute `Get-Help about_EntityFrameworkCore` no PMC para obter informações de ajuda.

## Criar uma migração inicial e atualizar o BD

Crie o projeto e a primeira migração.

- [Visual Studio](#)
- [CLI do .NET Core](#)

```
Add-Migration InitialCreate
Update-Database
```

### Examinar os métodos Up e Down

O comando `migrations add` do EF Core gerou um código para criar o BD. Esse código de migrações está localizado no arquivo *Migrations<timestamp>\_InitialCreate.cs*. O método `Up` da classe `InitialCreate` cria as tabelas de BD que correspondem aos conjuntos de entidades do modelo de dados. O método `Down` exclui-os, conforme mostrado no seguinte exemplo:

```
public partial class InitialCreate : Migration
{
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "Course",
            columns: table => new
            {
                CourseID = table.Column<int>(nullable: false),
                Title = table.Column<string>(nullable: true),
                Credits = table.Column<int>(nullable: false)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_Course", x => x.CourseID);
            });
        migrationBuilder.CreateTable(
protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "Enrollment");

    migrationBuilder.DropTable(
        name: "Course");

    migrationBuilder.DropTable(
        name: "Student");
}
}
```

As migrações chamam o método `Up` para implementar as alterações do modelo de dados para uma migração. Quando você insere um comando para reverter a atualização, as migrações chamam o método `Down`.

O código anterior refere-se à migração inicial. Esse código foi criado quando o comando `migrations add InitialCreate` foi executado. O parâmetro de nome da migração ("InitialCreate" no exemplo) é usado para o nome do arquivo. O nome da migração pode ser qualquer nome de arquivo válido. É melhor escolher uma palavra ou frase que resume o que está sendo feito na migração. Por exemplo, uma migração que adicionou uma tabela de departamento pode ser chamada "AddDepartmentTable".

Se a migração inicial foi criada e o BD existe:

- O código de criação do BD é gerado.
- O código de criação do BD não precisa ser executado porque o BD já corresponde ao modelo de dados. Se o código de criação do BD for executado, ele não fará nenhuma alteração porque o BD já corresponde ao modelo de dados.

Quando o aplicativo é implantado em um novo ambiente, o código de criação do BD precisa ser executado para criar o BD.

Anteriormente, o BD foi removido, e não existe mais. Então, as migrações criam o novo BD.

## O instantâneo do modelo de dados

As migrações criam um *instantâneo* do esquema de banco de dados atual em `Migrations/SchoolContextModelSnapshot.cs`. Quando você adiciona uma migração, o EF determina o que foi alterado, comparando o modelo de dados com o arquivo de instantâneo.

Para excluir uma migração, use o seguinte comando:

- [Visual Studio](#)
- [CLI do .NET Core](#)

`Remove-Migration`

O comando de exclusão de migrações exclui a migração e garante que o instantâneo seja redefinido corretamente.

## Remover `EnsureCreated` e testar o aplicativo

Para o desenvolvimento inicial, `EnsureCreated` foi usado. Neste tutorial, as migrações são usadas. `EnsureCreated` tem as seguintes limitações:

- Ignora as migrações e cria o BD e o esquema.
- Não cria uma tabela de migrações.
- *Não* pode ser usado com migrações.
- Foi projetado para teste ou criação rápida de protótipos em que o BD é removido e recriado com frequência.

Remova a seguinte linha de `DbInitializer`:

```
context.Database.EnsureCreated();
```

Execute o aplicativo e verifique se o BD é propagado.

## Inspecionar o banco de dados

Use o **Pesquisador de Objetos do SQL Server** para inspecionar o BD. Observe a adição de uma tabela `_EFMigrationsHistory`. A tabela `_EFMigrationsHistory` controla quais migrações foram aplicadas ao BD. Exiba os dados na `_EFMigrationsHistory` tabela; ela mostra uma linha para a primeira migração. O último log no exemplo de saída da CLI anterior mostra a instrução INSERT que cria essa linha.

Execute o aplicativo e verifique se tudo funciona.

## Aplicando migrações na produção

Recomendamos que os aplicativos de produção **não** chamem `Database.Migrate` na inicialização do aplicativo. `Migrate` não deve ser chamado em um aplicativo no farm de servidores. Por exemplo, se o aplicativo foi implantado na nuvem com escalabilidade horizontal (várias instâncias do aplicativo estão sendo executadas).

A migração de banco de dados deve ser feita como parte da implantação e de maneira controlada. Abordagens de migração de banco de dados de produção incluem:

- Uso de migrações para criar scripts SQL e uso dos scripts SQL na implantação.
- Execução de `dotnet ef database update` em um ambiente controlado.

O EF Core usa a tabela `_MigrationsHistory` para ver se uma migração precisa ser executada. Se o BD estiver

atualizado, nenhuma migração será executada.

## Solução de problemas

Baixar o [aplicativo concluído](#).

O aplicativo gera a seguinte exceção:

```
SqlException: Cannot open database "ContosoUniversity" requested by the login.  
The login failed.  
Login failed for user 'user name'.
```

Solução: execute `dotnet ef database update`

### Recursos adicionais

- [CLI do .NET Core](#).
- [Console do Gerenciador de Pacotes \(Visual Studio\)](#)

[ANTERIOR](#)

[PRÓXIMO](#)

# Páginas Razor com o EF Core no ASP.NET Core – Modelo de dados – 5 de 8

10/01/2019 • 47 minutes to read • [Edit Online](#)

A versão deste tutorial para o ASP.NET Core 2.0 pode ser encontrada neste [arquivo PDF](#).

A versão deste tutorial para o ASP.NET Core 2.1 contém diversas melhorias em relação à versão 2.0.

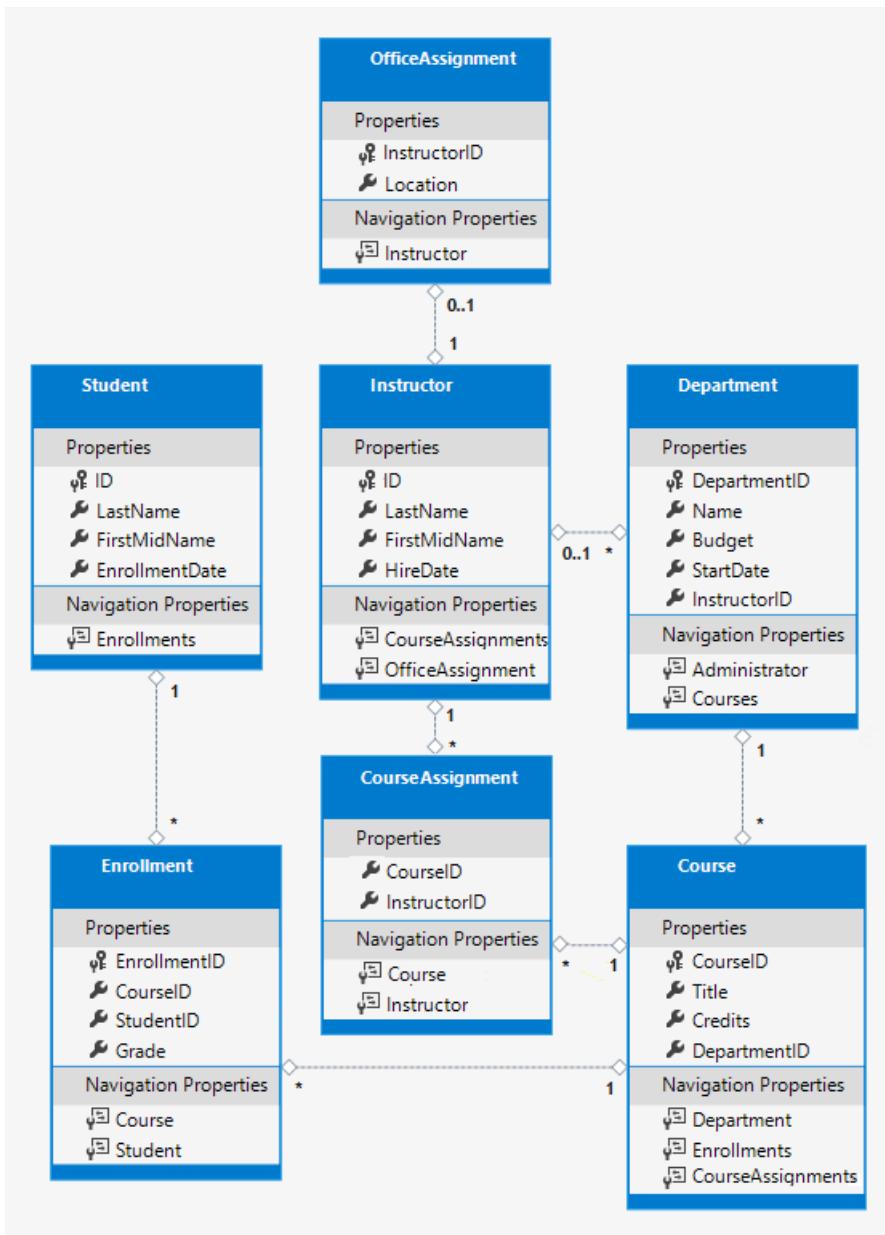
Por [Tom Dykstra](#) e [Rick Anderson](#)

O aplicativo Web Contoso University demonstra como criar aplicativos Web das Páginas do Razor usando o EF Core e o Visual Studio. Para obter informações sobre a série de tutoriais, consulte [o primeiro tutorial](#).

Os tutoriais anteriores trabalharam com um modelo de dados básico composto por três entidades. Neste tutorial:

- Mais entidades e relações são adicionadas.
- O modelo de dados é personalizado com a especificação das regras de formatação, validação e mapeamento de banco de dados.

As classes de entidade para o modelo de dados concluído são mostradas na seguinte ilustração:



Caso tenha problemas que não consiga resolver, baixe o [aplicativo concluído](#).

## Personalizar o modelo de dados com atributos

Nesta seção, o modelo de dados é personalizado com atributos.

### O atributo `DataType`

As páginas de alunos atualmente exibem a hora da data de registro. Normalmente, os campos de data mostram apenas a data e não a hora.

Atualize `Models/Student.cs` com o seguinte código realçado:

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models
{
    public class Student
    {
        public int ID { get; set; }
        public string LastName { get; set; }
        public string FirstMidName { get; set; }
        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        public DateTime EnrollmentDate { get; set; }

        public ICollection<Enrollment> Enrollments { get; set; }
    }
}

```

O atributo `DataType` especifica um tipo de dados mais específico do que o tipo intrínseco de banco de dados. Neste caso, apenas a data deve ser exibida, não a data e a hora. A [Enumeração `DataType`](#) fornece muitos tipos de dados, como Date, Time, PhoneNumber, Currency, EmailAddress, etc. O atributo `DataType` também pode permitir que o aplicativo forneça automaticamente recursos específicos a um tipo. Por exemplo:

- O link `mailto:` é criado automaticamente para `DataType.EmailAddress`.
- O seletor de data é fornecido para `DataType.Date` na maioria dos navegadores.

O atributo `DataType` emite atributos `data-` HTML 5 (pronunciados “data dash”) que são consumidos pelos navegadores HTML 5. Os atributos `DataType` não fornecem validação.

`DataType.Date` não especifica o formato da data exibida. Por padrão, o campo de dados é exibido de acordo com os formatos padrão com base nas [CultureInfo](#) do servidor.

O atributo `DisplayFormat` é usado para especificar explicitamente o formato de data:

```
[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
```

A configuração `ApplyFormatInEditMode` especifica que a formatação também deve ser aplicada à interface do usuário de edição. Alguns campos não devem usar `ApplyFormatInEditMode`. Por exemplo, o símbolo de moeda geralmente não deve ser exibido em uma caixa de texto de edição.

O atributo `DisplayFormat` pode ser usado por si só. Geralmente, é uma boa ideia usar o atributo `DataType` com o atributo `DisplayFormat`. O atributo `DataType` transmite a semântica dos dados em vez de como renderizá-los em uma tela. O atributo `DataType` oferece os seguintes benefícios que não estão disponíveis em `DisplayFormat`:

- O navegador pode habilitar recursos do HTML5. Por exemplo, mostra um controle de calendário, o símbolo de moeda apropriado à localidade, links de email, validação de entrada do lado do cliente, etc.
- Por padrão, o navegador renderiza os dados usando o formato correto de acordo com a localidade.

Para obter mais informações, consulte a [documentação do Auxiliar de Marcação <input>](#).

Execute o aplicativo. Navegue para a página Índice de Alunos. As horas não são mais exibidas. Cada exibição que usa o modelo `student` exibe a data sem a hora.

Last Name	First Name	Enrollment Date	
Alexander	Carson	2005-09-01	Edit   Details   Delete
Alonso	Meredith	2002-09-01	Edit   Details   Delete
Anand	Arturo	2003-09-01	Edit   Details   Delete

## O atributo StringLength

Regras de validação de dados e mensagens de erro de validação podem ser especificadas com atributos. O atributo `StringLength` especifica o tamanho mínimo e máximo de caracteres permitidos em um campo de dados. O atributo `StringLength` também fornece a validação do lado do cliente e do servidor. O valor mínimo não tem impacto sobre o esquema de banco de dados.

Atualize o modelo `Student` com o seguinte código:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models
{
    public class Student
    {
        public int ID { get; set; }
        [StringLength(50)]
        public string LastName { get; set; }
        [StringLength(50, ErrorMessage = "First name cannot be longer than 50 characters.")]  
        public string FirstMidName { get; set; }
        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        public DateTime EnrollmentDate { get; set; }

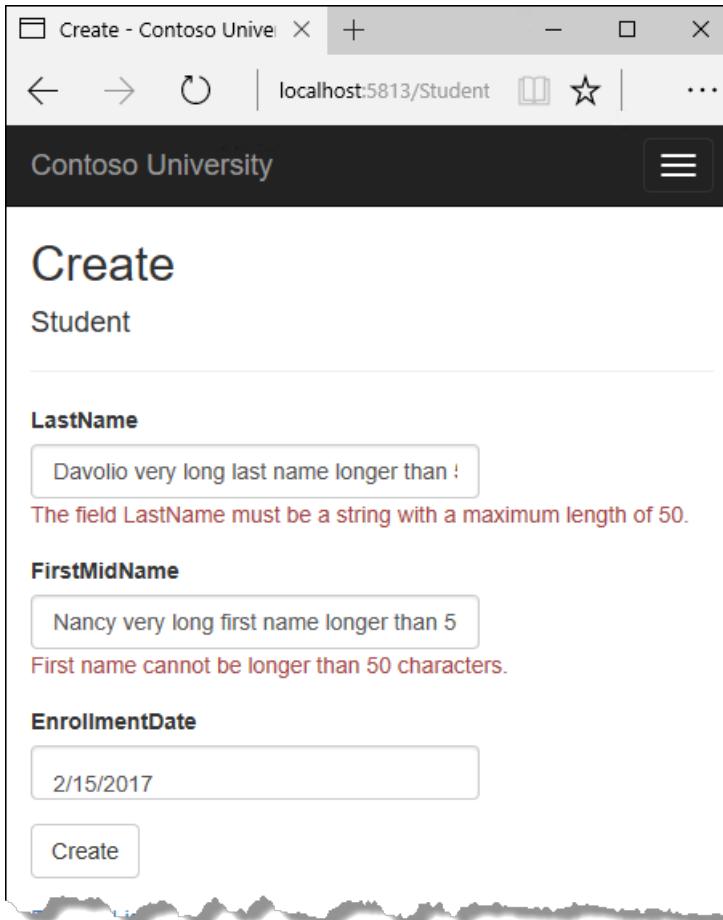
        public ICollection<Enrollment> Enrollments { get; set; }
    }
}
```

O código anterior limita os nomes a, no máximo, 50 caracteres. O atributo `StringLength` não impede que um usuário insira um espaço em branco em um nome. O atributo `RegularExpression` é usado para aplicar restrições à entrada. Por exemplo, o seguinte código exige que o primeiro caractere esteja em maiúscula e os caracteres restantes estejam em ordem alfabética:

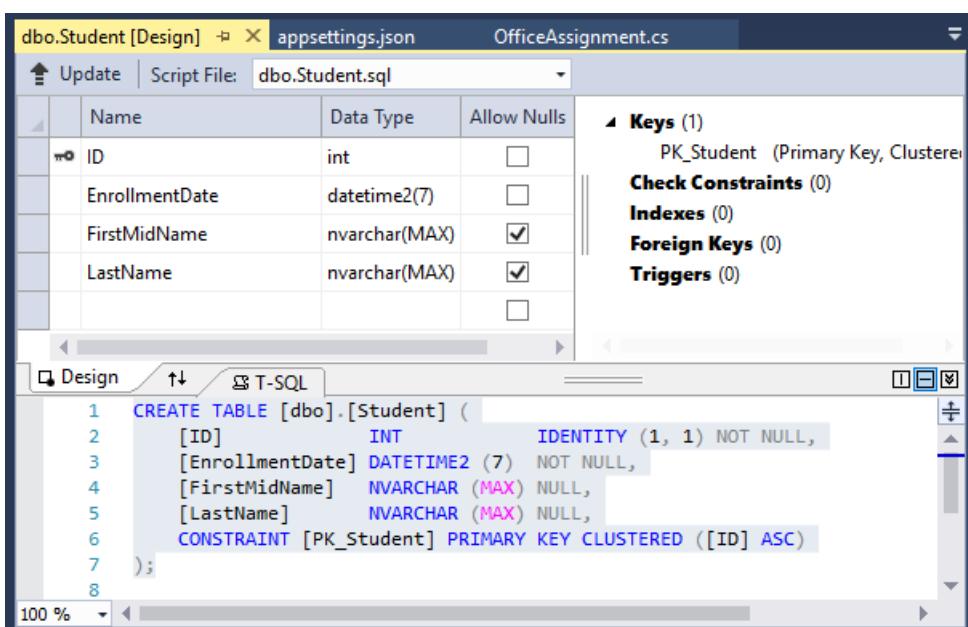
```
[RegularExpression(@"^([A-Z]+[a-zA-Z''"\s-]*)$")]
```

Execute o aplicativo:

- Navegue para a página Alunos.
- Selecione **Criar Novo** e insira um nome com mais de 50 caracteres.
- Selecione **Criar** e a validação do lado do cliente mostrará uma mensagem de erro.



No **SSOX** (Pesquisador de Objetos do SQL Server), abra o designer de tabela Aluno clicando duas vezes na tabela **Aluno**.



A imagem anterior mostra o esquema para a tabela `Student`. Os campos de nome têm o tipo `nvarchar(MAX)`

porque as migrações não foram executadas no BD. Quando as migrações forem executadas mais adiante neste tutorial, os campos de nome se tornarão `nvarchar(50)`.

## O atributo Column

Os atributos podem controlar como as classes e propriedades são mapeadas para o banco de dados. Nesta seção, o atributo `Column` é usado para mapear o nome da propriedade `FirstMidName` como "FirstName" no BD.

Quando o BD é criado, os nomes de propriedade no modelo são usados para nomes de coluna (exceto quando o atributo `Column` é usado).

O modelo `Student` usa `FirstMidName` para o campo de nome porque o campo também pode conter um sobrenome.

Atualize o arquivo `Student.cs` com o seguinte código:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Student
    {
        public int ID { get; set; }
        [StringLength(50)]
        public string LastName { get; set; }
        [StringLength(50, ErrorMessage = "First name cannot be longer than 50 characters.")]  
[Column("FirstName")]
        public string FirstMidName { get; set; }
        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        public DateTime EnrollmentDate { get; set; }

        public ICollection<Enrollment> Enrollments { get; set; }
    }
}
```

Com a alteração anterior, `Student.FirstMidName` no aplicativo é mapeado para a coluna `FirstName` da tabela `Student`.

A adição do atributo `Column` altera o modelo que dá suporte ao `SchoolContext`. O modelo que dá suporte ao `SchoolContext` não corresponde mais ao banco de dados. Se o aplicativo for executado antes da aplicação das migrações, a seguinte exceção será gerada:

```
SqlException: Invalid column name 'FirstName'.
```

Para atualizar o BD:

- Compile o projeto.
- Abra uma janela Comando na pasta do projeto. Insira os seguintes comandos para criar uma nova migração e atualizar o BD:
  - [Visual Studio](#)
  - [CLI do .NET Core](#)

```
Add-Migration ColumnFirstName  
Update-Database
```

O comando `migrations add ColumnFirstName` gera a seguinte mensagem de aviso:

```
An operation was scaffolded that may result in the loss of data.  
Please review the migration for accuracy.
```

O aviso é gerado porque os campos de nome agora estão limitados a 50 caracteres. Se um nome no BD tiver mais de 50 caracteres, o 51º caractere até o último caractere serão perdidos.

- Teste o aplicativo.

Abra a tabela Alunos no SSOX:

Name	Data Type	Allow Nulls
ID	int	<input type="checkbox"/>
EnrollmentDate	datetime2(7)	<input type="checkbox"/>
FirstName	nvarchar(50)	<input checked="" type="checkbox"/>
LastName	nvarchar(50)	<input checked="" type="checkbox"/>

**Keys (1)**  
PK\_Students (Primary Key, Clu  
**Check Constraints (0)**  
**Indexes (0)**  
**Foreign Keys (0)**  
**Triggers (0)**

```
1  CREATE TABLE [dbo].[Students] (  
2      [ID]           INT            IDENTITY (1, 1) NOT NULL,  
3      [EnrollmentDate] DATETIME2 (7) NOT NULL,  
4      [FirstName]      NVARCHAR (50) NULL,  
5      [LastName]       NVARCHAR (50) NULL,  
6      CONSTRAINT [PK_Students] PRIMARY KEY CLUSTERED ([ID] ASC)  
7  );  
8
```

Antes de a migração ser aplicada, as colunas de nome eram do tipo `nvarchar(MAX)`. As colunas de nome agora são `nvarchar(50)`. O nome da coluna foi alterado de `FirstMidName` para `FirstName`.

#### NOTE

Na seção a seguir, a criação do aplicativo em alguns estágios gera erros do compilador. As instruções especificam quando compilar o aplicativo.

## Atualização da entidade Student

Atualize `Models/Student.cs` com o seguinte código:

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Student
    {
        public int ID { get; set; }

        [Required]
        [StringLength(50)]
        [Display(Name = "Last Name")]
        public string LastName { get; set; }

        [Required]
        [StringLength(50, ErrorMessage = "First name cannot be longer than 50 characters.")]
        [Column("FirstName")]
        [Display(Name = "First Name")]
        public string FirstMidName { get; set; }

        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        [Display(Name = "Enrollment Date")]
        public DateTime EnrollmentDate { get; set; }

        [Display(Name = "Full Name")]
        public string FullName
        {
            get
            {
                return LastName + ", " + FirstMidName;
            }
        }

        public ICollection<Enrollment> Enrollments { get; set; }
    }
}

```

## O atributo Required

O atributo `Required` torna as propriedades de nome campos obrigatórios. O atributo `Required` não é necessário para tipos que não permitem valor nulo, como tipos de valor (`DateTime`, `int`, `double`, etc.). Tipos que não podem ser nulos são tratados automaticamente como campos obrigatórios.

O atributo `Required` pode ser substituído por um parâmetro de tamanho mínimo no atributo `StringLength`:

```

[Display(Name = "Last Name")]
[StringLength(50, MinimumLength=1)]
public string LastName { get; set; }

```

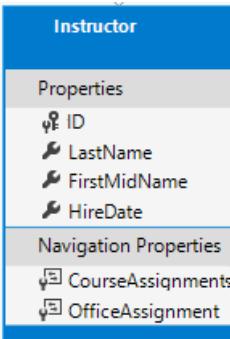
## O atributo Display

O atributo `Display` especifica que a legenda para as caixas de texto deve ser "Nome", "Sobrenome", "Nome Completo" e "Data de Registro". As legendas padrão não tinham nenhum espaço entre as palavras, por exemplo, "Lastname".

## A propriedade calculada FullName

`FullName` é uma propriedade calculada que retorna um valor criado pela concatenação de duas outras propriedades. `FullName` não pode ser definido; ele apenas tem um acessador `get`. Nenhuma coluna `FullName` é criada no banco de dados.

# Criar a entidade Instructor



Crie `Models/Instructor.cs` com o seguinte código:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Instructor
    {
        public int ID { get; set; }

        [Required]
        [Display(Name = "Last Name")]
        [StringLength(50)]
        public string LastName { get; set; }

        [Required]
        [Column("FirstName")]
        [Display(Name = "First Name")]
        [StringLength(50)]
        public string FirstMidName { get; set; }

        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        [Display(Name = "Hire Date")]
        public DateTime HireDate { get; set; }

        [Display(Name = "Full Name")]
        public string FullName
        {
            get { return LastName + ", " + FirstMidName; }
        }

        public ICollection<CourseAssignment> CourseAssignments { get; set; }
        public OfficeAssignment OfficeAssignment { get; set; }
    }
}
```

Vários atributos podem estar em uma linha. Os atributos `HireDate` podem ser escritos da seguinte maneira:

```
[DataType(DataType.Date),Display(Name = "Hire Date"),DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
```

## As propriedades de navegação `CourseAssignments` e `OfficeAssignment`

As propriedades `CourseAssignments` e `OfficeAssignment` são propriedades de navegação.

Um instrutor pode ministrar qualquer quantidade de cursos e, portanto, `CourseAssignments` é definido como uma coleção.

```
public ICollection<CourseAssignment> CourseAssignments { get; set; }
```

Se uma propriedade de navegação armazenar várias entidades:

- Ele deve ser um tipo de lista no qual as entradas possam ser adicionadas, excluídas e atualizadas.

Os tipos de propriedade de navegação incluem:

- `ICollection<T>`
- `List<T>`
- `HashSet<T>`

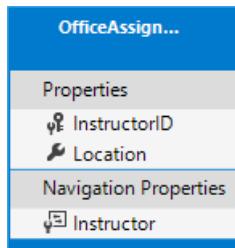
Se `ICollection<T>` for especificado, o EF Core criará uma coleção `HashSet<T>` por padrão.

A entidade `CourseAssignment` é explicada na seção sobre relações muitos para muitos.

Regras de negócio do Contoso University indicam que um instrutor pode ter, no máximo, um escritório. A propriedade `OfficeAssignment` contém uma única entidade `OfficeAssignment`. `OfficeAssignment` será nulo se nenhum escritório for atribuído.

```
public OfficeAssignment OfficeAssignment { get; set; }
```

## Criar a entidade OfficeAssignment



Crie `Models/OfficeAssignment.cs` com o seguinte código:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class OfficeAssignment
    {
        [Key]
        public int InstructorID { get; set; }
        [StringLength(50)]
        [Display(Name = "Office Location")]
        public string Location { get; set; }

        public Instructor Instructor { get; set; }
    }
}
```

### O atributo Key

O atributo `[Key]` é usado para identificar uma propriedade como a PK (chave primária) quando o nome da propriedade é algo diferente de `classnameID` ou `ID`.

Há uma relação um para zero ou um entre as entidades `Instructor` e `OfficeAssignment`. Uma atribuição de escritório existe apenas em relação ao instrutor ao qual ela é atribuída. A PK `OfficeAssignment` também é a FK

(chave estrangeira) da entidade `Instructor`. O EF Core não pode reconhecer `InstructorID` automaticamente como o PK de `OfficeAssignment` porque:

- `InstructorID` não segue a convenção de nomenclatura de ID nem de `classnameID`.

Portanto, o atributo `Key` é usado para identificar `InstructorID` como a PK:

```
[Key]  
public int InstructorID { get; set; }
```

Por padrão, o EF Core trata a chave como não gerada pelo banco de dados porque a coluna destina-se a uma relação de identificação.

### A propriedade de navegação `Instructor`

A propriedade de navegação `OfficeAssignment` da entidade `Instructor` permite valor nulo porque:

- Tipos de referência (como classes que permitem valor nulo).
- Um instrutor pode não ter uma atribuição de escritório.

A entidade `OfficeAssignment` tem uma propriedade de navegação `Instructor` que não permite valor nulo porque:

- `InstructorID` não permite valor nulo.
- Uma atribuição de escritório não pode existir sem um instrutor.

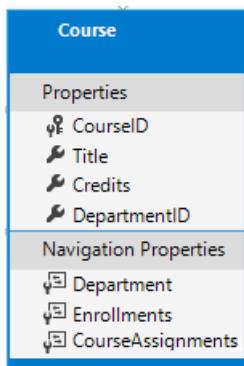
Quando uma entidade `Instructor` tem uma entidade `OfficeAssignment` relacionada, cada entidade tem uma referência à outra em sua propriedade de navegação.

O atributo `[Required]` pode ser aplicado à propriedade de navegação `Instructor`:

```
[Required]  
public Instructor Instructor { get; set; }
```

O código anterior especifica que deve haver um instrutor relacionado. O código anterior é desnecessário porque a chave estrangeira `InstructorID` (que também é a PK) não permite valor nulo.

## Modificar a entidade Course



Atualize `Models/Course.cs` com o seguinte código:

```

using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Course
    {
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        [Display(Name = "Number")]
        public int CourseID { get; set; }

        [StringLength(50, MinimumLength = 3)]
        public string Title { get; set; }

        [Range(0, 5)]
        public int Credits { get; set; }

        public int DepartmentID { get; set; }

        public Department Department { get; set; }
        public ICollection<Enrollment> Enrollments { get; set; }
        public ICollection<CourseAssignment> CourseAssignments { get; set; }
    }
}

```

A entidade `Course` tem uma propriedade de FK (chave estrangeira) `DepartmentID`. `DepartmentID` aponta para a entidade `Department` relacionada. A entidade `Course` tem uma propriedade de navegação `Department`.

O EF Core não exige uma propriedade de FK para um modelo de dados quando o modelo tem uma propriedade de navegação para uma entidade relacionada.

O EF Core cria automaticamente FKs no banco de dados sempre que forem necessárias. O EF Core cria [propriedades de sombra](#) para FKs criadas automaticamente. Ter a FK no modelo de dados pode tornar as atualizações mais simples e mais eficientes. Por exemplo, considere um modelo em que a propriedade de FK `DepartmentID` *não* é incluída. Quando uma entidade de curso é buscada para editar:

- A entidade `Department` será nula se não for carregada de forma explícita.
- Para atualizar a entidade de curso, a entidade `Department` primeiro deve ser buscada.

Quando a propriedade de FK `DepartmentID` está incluída no modelo de dados, não é necessário buscar a entidade `Department` antes de uma atualização.

### O atributo `DatabaseGenerated`

O atributo `[DatabaseGenerated(DatabaseGeneratedOption.None)]` especifica que a PK é fornecida pelo aplicativo em vez de ser gerada pelo banco de dados.

```

[DatabaseGenerated(DatabaseGeneratedOption.None)]
[Display(Name = "Number")]
public int CourseID { get; set; }

```

Por padrão, o EF Core supõe que os valores de PK sejam gerados pelo BD. Os valores de PK gerados pelo BD geralmente são a melhor abordagem. Para entidades `Course`, o usuário especifica o PK. Por exemplo, um número de curso, como uma série 1000 para o departamento de matemática e uma série 2000 para o departamento em inglês.

O atributo `DatabaseGenerated` também pode ser usado para gerar valores padrão. Por exemplo, o BD pode gerar automaticamente um campo de data para registrar a data em que uma linha foi criada ou atualizada. Para obter mais informações, consulte [Propriedades geradas](#).

## Propriedades de navegação e de chave estrangeira

As propriedades de navegação e de FK (chave estrangeira) na entidade `Course` refletem as seguintes relações:

Um curso é atribuído a um departamento; portanto, há uma FK `DepartmentID` e uma propriedade de navegação `Department`.

```
public int DepartmentID { get; set; }
public Department Department { get; set; }
```

Um curso pode ter qualquer quantidade de estudantes inscritos; portanto, a propriedade de navegação

`Enrollments` é uma coleção:

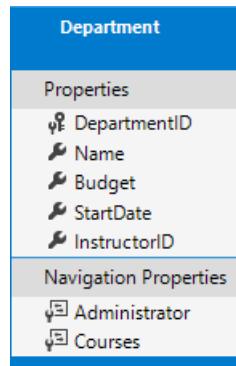
```
public ICollection<Enrollment> Enrollments { get; set; }
```

Um curso pode ser ministrado por vários instrutores; portanto, a propriedade de navegação `CourseAssignments` é uma coleção:

```
public ICollection<CourseAssignment> CourseAssignments { get; set; }
```

`CourseAssignment` é explicado [posteriormente](#).

## Criar a entidade Department



Crie `Models/Department.cs` com o seguinte código:

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Department
    {
        public int DepartmentID { get; set; }

        [StringLength(50, MinimumLength = 3)]
        public string Name { get; set; }

        [DataType(DataType.Currency)]
        [Column(TypeName = "money")]
        public decimal Budget { get; set; }

        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        [Display(Name = "Start Date")]
        public DateTime StartDate { get; set; }

        public int? InstructorID { get; set; }

        public Instructor Administrator { get; set; }
        public ICollection<Course> Courses { get; set; }
    }
}

```

## O atributo Column

Anteriormente, o atributo `Column` foi usado para alterar o mapeamento de nome de coluna. No código da entidade `Department`, o atributo `Column` é usado para alterar o mapeamento de tipo de dados SQL. A coluna `Budget` é definida usando o tipo de dinheiro do SQL Server no BD:

```

[Column(TypeName="money")]
public decimal Budget { get; set; }

```

Em geral, o mapeamento de coluna não é necessário. Em geral, o EF Core escolhe o tipo de dados do SQL Server apropriado com base no tipo CLR da propriedade. O tipo `decimal` CLR é mapeado para um tipo `decimal` SQL Server. `Budget` refere-se à moeda e o tipo de dados de dinheiro é mais apropriado para moeda.

## Propriedades de navegação e de chave estrangeira

As propriedades de navegação e de FK refletem as seguintes relações:

- Um departamento pode ou não ter um administrador.
- Um administrador é sempre um instrutor. Portanto, a propriedade `InstructorID` está incluída como a FK da entidade `Instructor`.

A propriedade de navegação é chamada `Administrator`, mas contém uma entidade `Instructor`:

```

public int? InstructorID { get; set; }
public Instructor Administrator { get; set; }

```

O ponto de interrogação (?) no código anterior especifica que a propriedade permite valor nulo.

Um departamento pode ter vários cursos e, portanto, há uma propriedade de navegação `Courses`:

```
public ICollection<Course> Courses { get; set; }
```

Observação: por convenção, o EF Core habilita a exclusão em cascata em FKs que não permitem valor nulo e em relações muitos para muitos. A exclusão em cascata pode resultar em regras de exclusão em cascata circular. As regras de exclusão em cascata circular causam uma exceção quando uma migração é adicionada.

Por exemplo, se a propriedade `Department.InstructorID` não foi definida como uma propriedade que permite valor nulo:

- O EF Core configura uma regra de exclusão em cascata para excluir o instrutor quando o departamento é excluído.
- A exclusão do instrutor quando o departamento é excluído não é o comportamento pretendido.

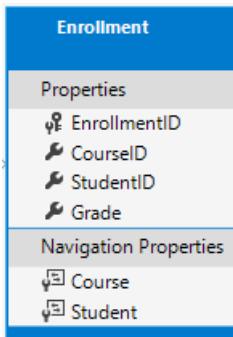
Se as regras de negócio exigirem que a propriedade `InstructorID` não permita valor nulo, use a seguinte instrução da API fluente:

```
modelBuilder.Entity<Department>()
    .HasOne(d => d.Administrator)
    .WithMany()
    .OnDelete(DeleteBehavior.Restrict)
```

O código anterior desabilita a exclusão em cascata na relação departamento-instrutor.

## Atualizar a entidade Enrollment

Um registro se refere a um curso feito por um aluno.



Atualize `Models/Enrollment.cs` com o seguinte código:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public enum Grade
    {
        A, B, C, D, F
    }

    public class Enrollment
    {
        public int EnrollmentID { get; set; }
        public int CourseID { get; set; }
        public int StudentID { get; set; }
        [DisplayFormat(NullDisplayText = "No grade")]
        public Grade? Grade { get; set; }

        public Course Course { get; set; }
        public Student Student { get; set; }
    }
}
```

## Propriedades de navegação e de chave estrangeira

As propriedades de navegação e de FK refletem as seguintes relações:

Um registro destina-se a um curso e, portanto, há uma propriedade de FK `courseID` e uma propriedade de navegação `Course`:

```
public int CourseID { get; set; }
public Course Course { get; set; }
```

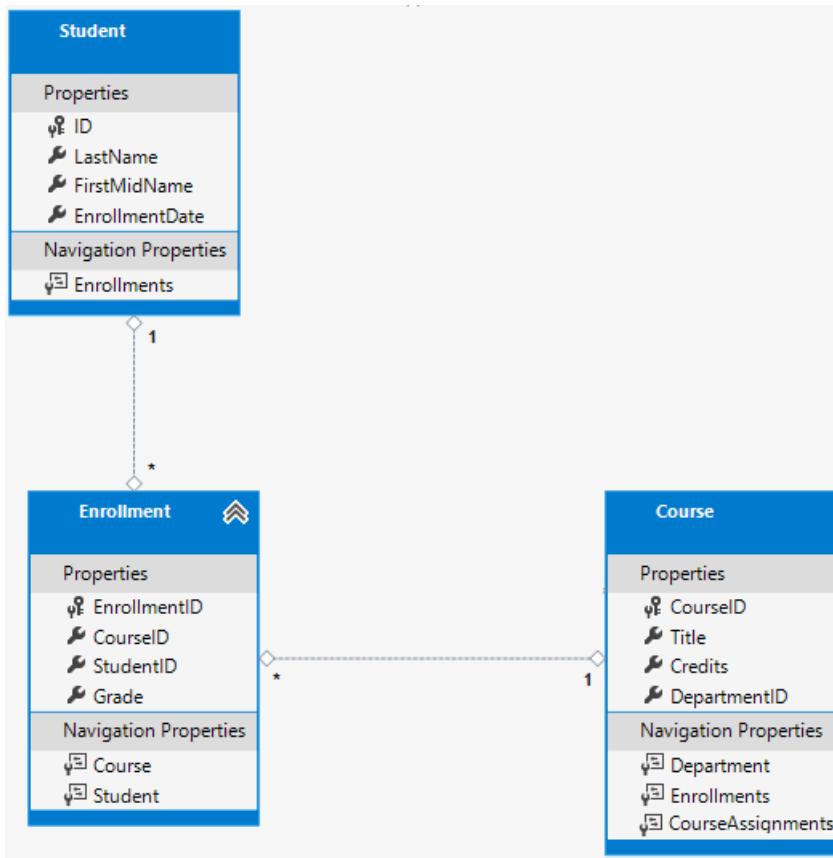
Um registro destina-se a um aluno e, portanto, há uma propriedade de FK `studentID` e uma propriedade de navegação `Student`:

```
public int StudentID { get; set; }
public Student Student { get; set; }
```

## Relações muitos para muitos

Há uma relação muitos para muitos entre as entidades `Student` e `Course`. A entidade `Enrollment` funciona como uma tabela de junção muitos para muitos *com conteúdo* no banco de dados. "Com conteúdo" significa que a tabela `Enrollment` contém dados adicionais além das FKs das tabelas unidas (nesse caso, a FK e `Grade`).

A ilustração a seguir mostra a aparência dessas relações em um diagrama de entidades. (Esse diagrama foi gerado com o [EF Power Tools](#) para EF 6.x. A criação do diagrama não faz parte do tutorial.)



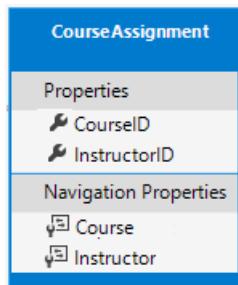
Cada linha de relação tem um 1 em uma extremidade e um asterisco (\*) na outra, indicando uma relação um para muitos.

Se a tabela `Enrollment` não incluir informações de nota, ela apenas precisará conter as duas FKs (`CourseID` e `StudentID`). Uma tabela de junção muitos para muitos sem conteúdo é às vezes chamada de PJT (uma tabela de junção pura).

As entidades `Instructor` e `Course` têm uma relação muitos para muitos usando uma tabela de junção pura.

Observação: O EF 6.x é compatível com tabelas de junção implícita para relações muitos para muitos, ao contrário do EF Core. Para obter mais informações, consulte [Relações muitos para muitos no EF Core 2.0](#).

## A entidade CourseAssignment



Crie `Models/CourseAssignment.cs` com o seguinte código:

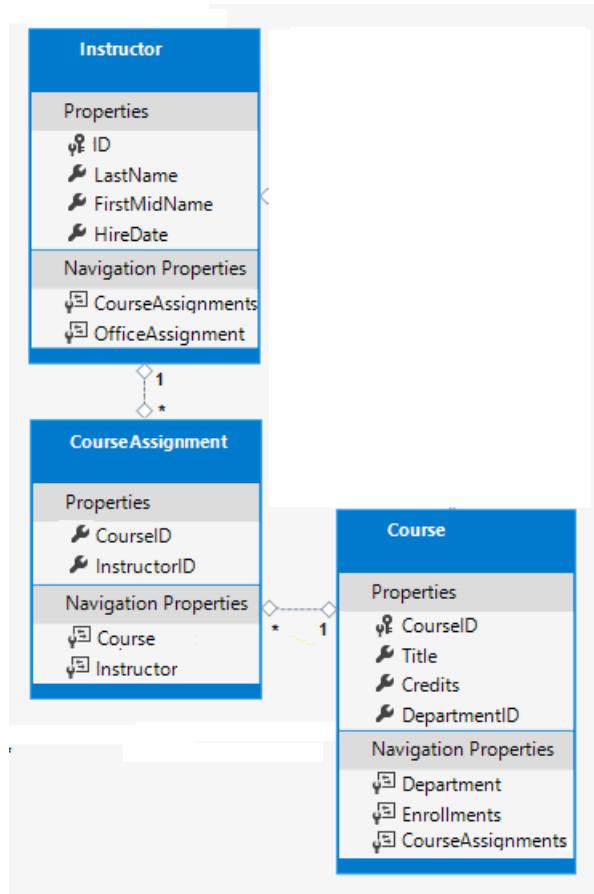
```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class CourseAssignment
    {
        public int InstructorID { get; set; }
        public int CourseID { get; set; }
        public Instructor Instructor { get; set; }
        public Course Course { get; set; }
    }
}

```

## Instrutor para Cursos



A relação muitos para muitos de Instrutor para Cursos:

- Exige que uma tabela de junção seja representada por um conjunto de entidades.
- É uma tabela de junção pura (tabela sem conteúdo).

É comum nomear uma entidade de junção `EntityName1EntityName2`. Por exemplo, a tabela de junção Instrutor para Cursos com esse padrão é `CourseInstructor`. No entanto, recomendamos que você use um nome que descreve a relação.

Modelos de dados começam simples e aumentam. PJs (junções sem conteúdo) evoluem com frequência para incluir o conteúdo. Começando com um nome descritivo de entidade, o nome não precisa ser alterado quando a tabela de junção é alterada. O ideal é que a entidade de junção tenha seu próprio nome natural (possivelmente, uma única palavra) no domínio de negócios. Por exemplo, Manuais e Clientes podem ser vinculados com uma entidade de junção chamada Ratings. Para a relação muitos para muitos de Instrutor para Cursos, `CourseAssignment` é preferível a `CourseInstructor`.

## Chave composta

As FKs não permitem valor nulo. As duas FKs em `CourseAssignment` (`InstructorID` e `CourseID`) juntas identificam exclusivamente cada linha da tabela `CourseAssignment`. `CourseAssignment` não exige um PK dedicado. As propriedades `InstructorID` e `CourseID` funcionam como uma PK composta. A única maneira de especificar PKs compostas no EF Core é com a *API fluente*. A próxima seção mostra como configurar a PK composta.

A chave composta garante:

- Várias linhas são permitidas para um curso.
- Várias linhas são permitidas para um instrutor.
- Não é permitido ter várias linhas para o mesmo instrutor e curso.

A entidade de junção `Enrollment` define sua própria PK e, portanto, duplicatas desse tipo são possíveis. Para impedir duplicatas como essas:

- Adicione um índice exclusivo nos campos de FK ou
- Configure `Enrollment` com uma chave primária composta semelhante a `CourseAssignment`. Para obter mais informações, consulte [Índices](#).

## Atualizar o contexto de BD

Adicione o seguinte código realçado a `Data/SchoolContext.cs`:

```
using ContosoUniversity.Models;
using Microsoft.EntityFrameworkCore;

namespace ContosoUniversity.Models
{
    public class SchoolContext : DbContext
    {
        public SchoolContext(DbContextOptions<SchoolContext> options) : base(options)
        {

        }

        public DbSet<Course> Courses { get; set; }
        public DbSet<Enrollment> Enrollment { get; set; }
        public DbSet<Student> Student { get; set; }
        public DbSet<Department> Departments { get; set; }
        public DbSet<Instructor> Instructors { get; set; }
        public DbSet<OfficeAssignment> OfficeAssignments { get; set; }
        public DbSet<CourseAssignment> CourseAssignments { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Course>().ToTable("Course");
            modelBuilder.Entity<Enrollment>().ToTable("Enrollment");
            modelBuilder.Entity<Student>().ToTable("Student");
            modelBuilder.Entity<Department>().ToTable("Department");
            modelBuilder.Entity<Instructor>().ToTable("Instructor");
            modelBuilder.Entity<OfficeAssignment>().ToTable("OfficeAssignment");
            modelBuilder.Entity<CourseAssignment>().ToTable("CourseAssignment");

            modelBuilder.Entity<CourseAssignment>()
                .HasKey(c => new { c.CourseID, c.InstructorID });
        }
    }
}
```

O código anterior adiciona novas entidades e configura a PK composta da entidade `CourseAssignment`.

## Alternativa de API fluente para atributos

O método `OnModelCreating` no código anterior usa a *API fluente* para configurar o comportamento do EF Core. A API é chamada "fluente" porque geralmente é usada pelo encadeamento de uma série de chamadas de método em uma única instrução. O [seguinte código](#) é um exemplo da API fluente:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .Property(b => b.Url)
        .IsRequired();
}
```

Neste tutorial, a API fluente é usada apenas para o mapeamento do BD que não pode ser feito com atributos. No entanto, a API fluente pode especificar a maioria das regras de formatação, validação e mapeamento que pode ser feita com atributos.

Alguns atributos como `MinimumLength` não podem ser aplicados com a API fluente. `MinimumLength` não altera o esquema; apenas aplica uma regra de validação de tamanho mínimo.

Alguns desenvolvedores preferem usar a API fluente exclusivamente para que possam manter suas classes de entidade "limpas". Atributos e a API fluente podem ser combinados. Há algumas configurações que apenas podem ser feitas com a API fluente (especificando uma PK composta). Há algumas configurações que apenas podem ser feitas com atributos (`MinimumLength`). A prática recomendada para uso de atributos ou da API fluente:

- Escolha uma dessas duas abordagens.
- Use a abordagem escolhida da forma mais consistente possível.

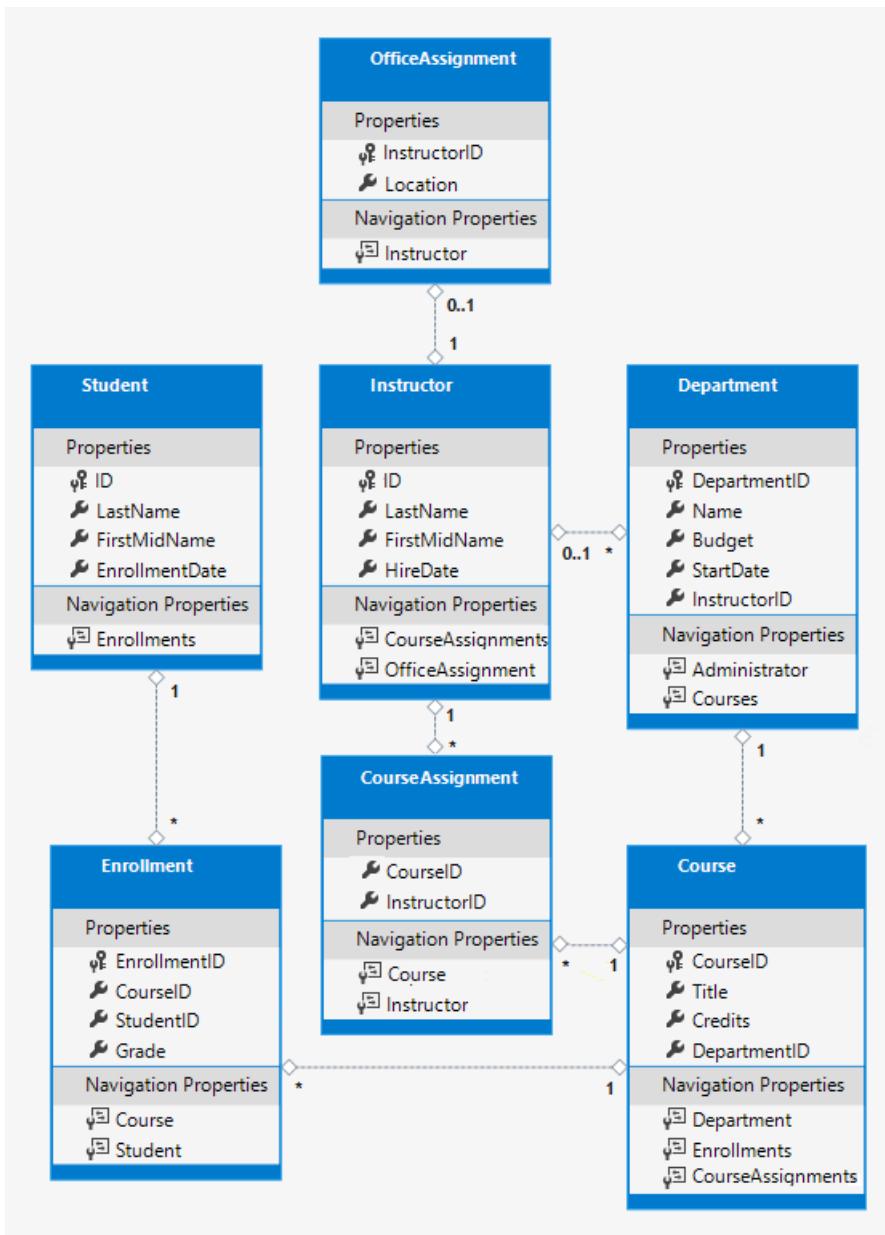
Alguns dos atributos usados neste tutorial são usados para:

- Somente validação (por exemplo, `MinimumLength`).
- Apenas configuração do EF Core (por exemplo, `HasKey`).
- Validação e configuração do EF Core (por exemplo, `[StringLength(50)]`).

Para obter mais informações sobre atributos vs. API fluente, consulte [Métodos de configuração](#).

## Diagrama de entidade mostrando relações

A ilustração a seguir mostra o diagrama criado pelo EF Power Tools para o modelo Escola concluído.



O diagrama anterior mostra:

- Várias linhas de relação um-para-muitos (1 para \*).
- A linha de relação um para zero ou um (1 para 0..1) entre as entidades `Instructor` e `OfficeAssignment`.
- A linha de relação zero-ou-um-para-muitos (0..1 para \*) entre as entidades `Instructor` e `Department`.

## Propagar o BD com os dados de teste

Atualize o código em `Data/DbInitializer.cs`:

```

using System;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using ContosoUniversity.Models;

namespace ContosoUniversity.Data
{
    public static class DbInitializer
    {
        public static void Initialize(SchoolContext context)
        {
            //context.Database.EnsureCreated();

```

```

// Look for any students.
if (context.Student.Any())
{
    return; // DB has been seeded
}

var students = new Student[]
{
    new Student { FirstMidName = "Carson", LastName = "Alexander",
        EnrollmentDate = DateTime.Parse("2010-09-01") },
    new Student { FirstMidName = "Meredith", LastName = "Alonso",
        EnrollmentDate = DateTime.Parse("2012-09-01") },
    new Student { FirstMidName = "Arturo", LastName = "Anand",
        EnrollmentDate = DateTime.Parse("2013-09-01") },
    new Student { FirstMidName = "Gytis", LastName = "Barzdukas",
        EnrollmentDate = DateTime.Parse("2012-09-01") },
    new Student { FirstMidName = "Yan", LastName = "Li",
        EnrollmentDate = DateTime.Parse("2012-09-01") },
    new Student { FirstMidName = "Peggy", LastName = "Justice",
        EnrollmentDate = DateTime.Parse("2011-09-01") },
    new Student { FirstMidName = "Laura", LastName = "Norman",
        EnrollmentDate = DateTime.Parse("2013-09-01") },
    new Student { FirstMidName = "Nino", LastName = "Olivetto",
        EnrollmentDate = DateTime.Parse("2005-09-01") }
};

foreach (Student s in students)
{
    context.Student.Add(s);
}
context.SaveChanges();

var instructors = new Instructor[]
{
    new Instructor { FirstMidName = "Kim", LastName = "Abercrombie",
        HireDate = DateTime.Parse("1995-03-11") },
    new Instructor { FirstMidName = "Fadi", LastName = "Fakhouri",
        HireDate = DateTime.Parse("2002-07-06") },
    new Instructor { FirstMidName = "Roger", LastName = "Harui",
        HireDate = DateTime.Parse("1998-07-01") },
    new Instructor { FirstMidName = "Candace", LastName = "Kapoor",
        HireDate = DateTime.Parse("2001-01-15") },
    new Instructor { FirstMidName = "Roger", LastName = "Zheng",
        HireDate = DateTime.Parse("2004-02-12") }
};

foreach (Instructor i in instructors)
{
    context.Instructors.Add(i);
}
context.SaveChanges();

var departments = new Department[]
{
    new Department { Name = "English", Budget = 350000,
        StartDate = DateTime.Parse("2007-09-01"),
        InstructorID = instructors.Single( i => i.LastName == "Abercrombie").ID },
    new Department { Name = "Mathematics", Budget = 100000,
        StartDate = DateTime.Parse("2007-09-01"),
        InstructorID = instructors.Single( i => i.LastName == "Fakhouri").ID },
    new Department { Name = "Engineering", Budget = 350000,
        StartDate = DateTime.Parse("2007-09-01"),
        InstructorID = instructors.Single( i => i.LastName == "Harui").ID },
    new Department { Name = "Economics", Budget = 100000,
        StartDate = DateTime.Parse("2007-09-01"),
        InstructorID = instructors.Single( i => i.LastName == "Kapoor").ID }
};

foreach (Department d in departments)

```

```

{
    context.Departments.Add(d);
}
context.SaveChanges();

var courses = new Course[]
{
    new Course {CourseID = 1050, Title = "Chemistry", Credits = 3,
        DepartmentID = departments.Single( s => s.Name == "Engineering").DepartmentID
    },
    new Course {CourseID = 4022, Title = "Microeconomics", Credits = 3,
        DepartmentID = departments.Single( s => s.Name == "Economics").DepartmentID
    },
    new Course {CourseID = 4041, Title = "Macroeconomics", Credits = 3,
        DepartmentID = departments.Single( s => s.Name == "Economics").DepartmentID
    },
    new Course {CourseID = 1045, Title = "Calculus", Credits = 4,
        DepartmentID = departments.Single( s => s.Name == "Mathematics").DepartmentID
    },
    new Course {CourseID = 3141, Title = "Trigonometry", Credits = 4,
        DepartmentID = departments.Single( s => s.Name == "Mathematics").DepartmentID
    },
    new Course {CourseID = 2021, Title = "Composition", Credits = 3,
        DepartmentID = departments.Single( s => s.Name == "English").DepartmentID
    },
    new Course {CourseID = 2042, Title = "Literature", Credits = 4,
        DepartmentID = departments.Single( s => s.Name == "English").DepartmentID
    },
};

foreach (Course c in courses)
{
    context.Courses.Add(c);
}
context.SaveChanges();

var officeAssignments = new OfficeAssignment[]
{
    new OfficeAssignment {
        InstructorID = instructors.Single( i => i.LastName == "Fakhouri").ID,
        Location = "Smith 17"
    },
    new OfficeAssignment {
        InstructorID = instructors.Single( i => i.LastName == "Harui").ID,
        Location = "Gowan 27"
    },
    new OfficeAssignment {
        InstructorID = instructors.Single( i => i.LastName == "Kapoor").ID,
        Location = "Thompson 304"
    };
}

foreach (OfficeAssignment o in officeAssignments)
{
    context.OfficeAssignments.Add(o);
}
context.SaveChanges();

var courseInstructors = new CourseAssignment[]
{
    new CourseAssignment {
        CourseID = courses.Single(c => c.Title == "Chemistry" ).CourseID,
        InstructorID = instructors.Single(i => i.LastName == "Kapoor").ID
    },
    new CourseAssignment {
        CourseID = courses.Single(c => c.Title == "Chemistry" ).CourseID,
        InstructorID = instructors.Single(i => i.LastName == "Harui").ID
    },
    new CourseAssignment {
        CourseID = courses.Single(c => c.Title == "Microeconomics" ).CourseID,
        InstructorID = instructors.Single(i => i.LastName == "Zheng").ID
    }
}.

```

```

        '',
new CourseAssignment {
    CourseID = courses.Single(c => c.Title == "Macroeconomics" ).CourseID,
    InstructorID = instructors.Single(i => i.LastName == "Zheng").ID
},
new CourseAssignment {
    CourseID = courses.Single(c => c.Title == "Calculus" ).CourseID,
    InstructorID = instructors.Single(i => i.LastName == "Fakhouri").ID
},
new CourseAssignment {
    CourseID = courses.Single(c => c.Title == "Trigonometry" ).CourseID,
    InstructorID = instructors.Single(i => i.LastName == "Harui").ID
},
new CourseAssignment {
    CourseID = courses.Single(c => c.Title == "Composition" ).CourseID,
    InstructorID = instructors.Single(i => i.LastName == "Abercrombie").ID
},
new CourseAssignment {
    CourseID = courses.Single(c => c.Title == "Literature" ).CourseID,
    InstructorID = instructors.Single(i => i.LastName == "Abercrombie").ID
},
};

foreach (CourseAssignment ci in courseInstructors)
{
    context.CourseAssignments.Add(ci);
}
context.SaveChanges();

var enrollments = new Enrollment[]
{
    new Enrollment {
        StudentID = students.Single(s => s.LastName == "Alexander").ID,
        CourseID = courses.Single(c => c.Title == "Chemistry" ).CourseID,
        Grade = Grade.A
    },
    new Enrollment {
        StudentID = students.Single(s => s.LastName == "Alexander").ID,
        CourseID = courses.Single(c => c.Title == "Microeconomics" ).CourseID,
        Grade = Grade.C
    },
    new Enrollment {
        StudentID = students.Single(s => s.LastName == "Alexander").ID,
        CourseID = courses.Single(c => c.Title == "Macroeconomics" ).CourseID,
        Grade = Grade.B
    },
    new Enrollment {
        StudentID = students.Single(s => s.LastName == "Alonso").ID,
        CourseID = courses.Single(c => c.Title == "Calculus" ).CourseID,
        Grade = Grade.B
    },
    new Enrollment {
        StudentID = students.Single(s => s.LastName == "Alonso").ID,
        CourseID = courses.Single(c => c.Title == "Trigonometry" ).CourseID,
        Grade = Grade.B
    },
    new Enrollment {
        StudentID = students.Single(s => s.LastName == "Alonso").ID,
        CourseID = courses.Single(c => c.Title == "Composition" ).CourseID,
        Grade = Grade.B
    },
    new Enrollment {
        StudentID = students.Single(s => s.LastName == "Anand").ID,
        CourseID = courses.Single(c => c.Title == "Chemistry" ).CourseID
    },
    new Enrollment {
        StudentID = students.Single(s => s.LastName == "Anand").ID,
        CourseID = courses.Single(c => c.Title == "Microeconomics").CourseID,
        Grade = Grade.B
    }
}

```

```

        },
        new Enrollment {
            StudentID = students.Single(s => s.LastName == "Barzdukas").ID,
            CourseID = courses.Single(c => c.Title == "Chemistry").CourseID,
            Grade = Grade.B
        },
        new Enrollment {
            StudentID = students.Single(s => s.LastName == "Li").ID,
            CourseID = courses.Single(c => c.Title == "Composition").CourseID,
            Grade = Grade.B
        },
        new Enrollment {
            StudentID = students.Single(s => s.LastName == "Justice").ID,
            CourseID = courses.Single(c => c.Title == "Literature").CourseID,
            Grade = Grade.B
        }
    }

    foreach (Enrollment e in enrollments)
    {
        var enrollmentInDataBase = context.Enrollment.Where(
            s =>
                s.Student.ID == e.StudentID &&
                s.Course.CourseID == e.CourseID).SingleOrDefault();
        if (enrollmentInDataBase == null)
        {
            context.Enrollment.Add(e);
        }
    }
    context.SaveChanges();
}
}

```

O código anterior fornece dados de semente para as novas entidades. A maioria desse código cria novos objetos de entidade e carrega dados de exemplo. Os dados de exemplo são usados para teste. Consulte [Enrollments](#) e [CourseAssignments](#) para obter exemplos de como tabelas de junção muitos para muitos podem ser propagadas.

## Adicionar uma migração

Compile o projeto.

- Visual Studio
  - CLI do .NET Core

```
Add-Migration ComplexDataModel
```

O comando anterior exibe um aviso sobre a possível perda de dados.

An operation was scaffolded that may result in the loss of data.  
Please review the migration for accuracy.  
Done. To undo this action, use 'ef migrations remove'

Se o comando `database update` é executado, o seguinte erro é produzido:

The ALTER TABLE statement conflicted with the FOREIGN KEY constraint "FK\_dbo.Course\_dbo.Department\_DepartmentID". The conflict occurred in database "ContosoUniversity", table "dbo.Department", column 'DepartmentID'.

# Aplicar a migração

Agora que você tem um banco de dados existente, precisa pensar sobre como aplicar as alterações futuras a ele. Este tutorial mostra duas abordagens:

- [Remover e recriar o banco de dados](#)
- [Aplicar a migração ao banco de dados existente](#). Embora esse método seja mais complexo e demorado, é a abordagem preferencial para ambientes de produção do mundo real. **Observação:** essa é uma seção opcional do tutorial. Você pode remover e recriar etapas e ignorar esta seção. Se você quiser seguir as etapas nesta seção, não realize as etapas de remover e recriar.

## Remover e recriar o banco de dados

O código no `DbInitializer` atualizado adiciona dados de semente às novas entidades. Para forçar o EF Core a criar um novo BD, remova e atualize o BD:

- [Visual Studio](#)
- [CLI do .NET Core](#)

No **PMC** (Console do Gerenciador de Pacotes), execute o seguinte comando:

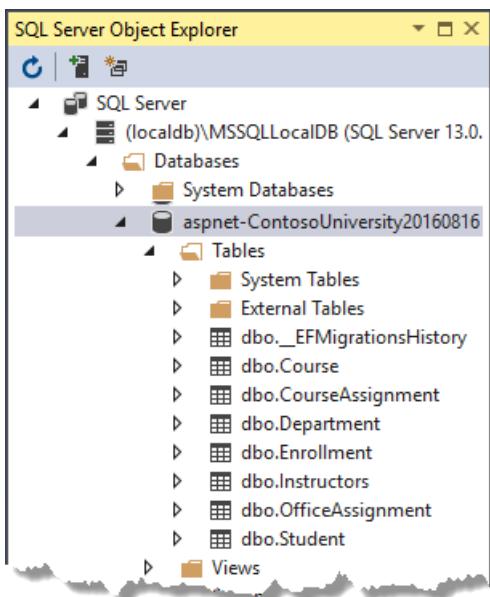
```
Drop-Database  
Update-Database
```

Execute `Get-Help about_EntityFrameworkCore` no PMC para obter informações de ajuda.

Execute o aplicativo. A execução do aplicativo executa o método `DbInitializer.Initialize`. O `DbInitializer.Initialize` popula o novo BD.

Abra o BD no SSOX:

- Se o SSOX for aberto anteriormente, clique no botão **Atualizar**.
- Expanda o nó **Tabelas**. As tabelas criadas são exibidas.



Examine a tabela **CourseAssignment**:

- Clique com o botão direito do mouse na tabela **CourseAssignment** e selecione **Exibir Dados**.
- Verifique se a tabela **CourseAssignment** contém dados.

The screenshot shows a database window titled 'ContosoUniversity' with the table 'dbo.CourseAssignment [Data]'. The table has two columns: 'CourseID' and 'InstructorID'. The data is as follows:

	CourseID	InstructorID
▶	2021	1
	2042	1
	1045	2
	1050	3
	3141	3
	1050	4
	4022	5
*	4041	5
	NULL	NULL

### Aplicar a migração ao banco de dados existente

Esta seção é opcional. Estas etapas só funcionarão se você tiver ignorado a seção [Remover e recadastrar o banco de dados](#) anterior.

Quando as migrações são executadas com os dados existentes, pode haver restrições de FK que não são atendidas com os dados existentes. Com os dados de produção, é necessário executar etapas para migrar os dados existentes. Esta seção fornece um exemplo de correção de violações de restrição de FK. Não faça essas alterações de código sem um backup. Não faça essas alterações de código se você concluir a seção anterior e atualizou o banco de dados.

O arquivo `{timestamp}_ComplexDataModel.cs` contém o seguinte código:

```
migrationBuilder.AddColumn<int>(
    name: "DepartmentID",
    table: "Course",
    type: "int",
    nullable: false,
    defaultValue: 0);
```

O código anterior adiciona uma FK `DepartmentID` que não permite valor nulo à tabela `Course`. O BD do tutorial anterior contém linhas em `Course` e, portanto, essa tabela não pode ser atualizada por migrações.

Para fazer a migração `ComplexDataModel` funcionar com os dados existentes:

- Altere o código para dar à nova coluna (`DepartmentID`) um valor padrão.
- Crie um departamento fictício chamado "Temp" para atuar como o departamento padrão.

#### Corrigir as restrições de chave estrangeira

Atualize o método `up` das classes `ComplexDataModel`:

- Abra o arquivo `{timestamp}_ComplexDataModel.cs`.
- Comente a linha de código que adiciona a coluna `DepartmentID` à tabela `Course`.

```
migrationBuilder.AlterColumn<string>(
    name: "Title",
    table: "Course",
    maxLength: 50,
    nullable: true,
    oldClrType: typeof(string),
    oldNullable: true);

//migrationBuilder.AddColumn<int>(
//    name: "DepartmentID",
//    table: "Course",
//    nullable: false,
//    defaultValue: 0);
```

Adicione o código realçado a seguir. O novo código é inserido após o bloco `.CreateTable( name: "Department" :`  
[!code-csharp]

Com as alterações anteriores, as linhas `Course` existentes estarão relacionadas ao departamento "Temp" após a execução do método `ComplexDataModel Up`.

Um aplicativo de produção:

- Inclui código ou scripts para adicionar linhas `Department` e linhas `Course` relacionadas às novas linhas `Department`.
- Não usa o departamento "Temp" nem o valor padrão para `Course.DepartmentID`.

O próximo tutorial abrange os dados relacionados.

[ANTERIOR](#)

[PRÓXIMO](#)

# Páginas Razor com o EF Core no ASP.NET Core – Ler dados relacionados – 6 de 8

30/10/2018 • 24 minutes to read • [Edit Online](#)

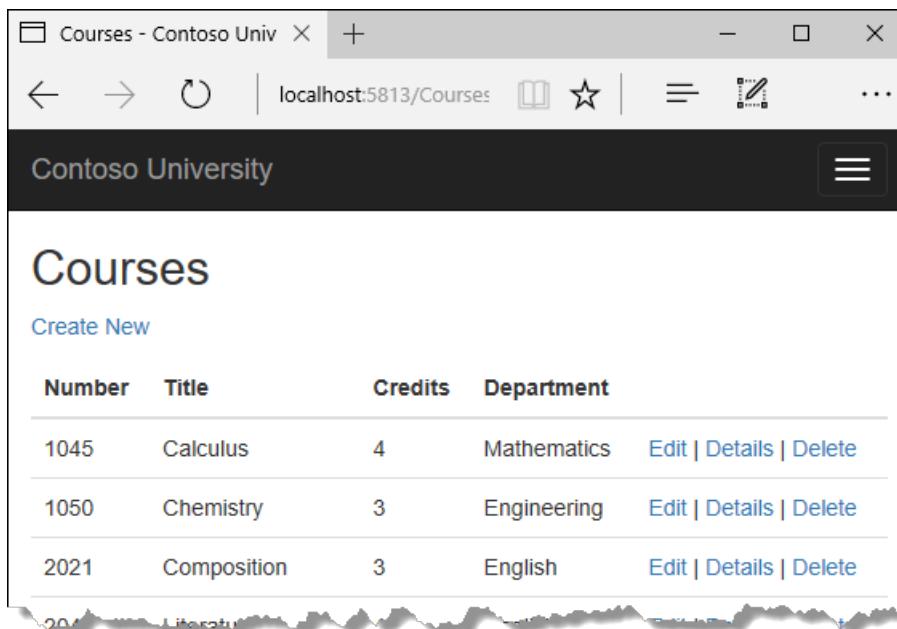
Por [Tom Dykstra](#), [Jon P Smith](#) e [Rick Anderson](#)

O aplicativo Web Contoso University demonstra como criar aplicativos Web das Páginas do Razor usando o EF Core e o Visual Studio. Para obter informações sobre a série de tutoriais, consulte [o primeiro tutorial](#).

Neste tutorial, os dados relacionados são lidos e exibidos. Dados relacionados são dados que o EF Core carrega nas propriedades de navegação.

Caso tenha problemas que não consiga resolver, [baixe ou exiba o aplicativo concluído](#). [Instruções de download](#).

As seguintes ilustrações mostram as páginas concluídas para este tutorial:



Instructors - Contoso University

localhost:1234/Instructors/3?courseID=1050&action=OnGetAsync

Contoso University

## Instructors

Create New

Last Name	First Name	Hire Date	Office	Courses	
Abercrombie	Kim	1995-03-11		2021 Composition 2042 Literature	Select   Edit   Details   Delete
Fakhouri	Fadi	2002-07-06	Smith 17	1045 Calculus	Select   Edit   Details   Delete
Harui	Roger	1998-07-01	Gowan 27	1050 Chemistry 3141 Trigonometry	Select   Edit   Details   Delete
Kapoor	Candace	2001-01-15	Thompson 304	1050 Chemistry	Select   Edit   Details   Delete
Zheng	Roger	2004-02-12		4022 Microeconomics 4041 Macroeconomics	Select   Edit   Details   Delete

### Courses Taught by Selected Instructor

	Number	Title	Department
Select	1050	Chemistry	Engineering
Select	3141	Trigonometry	Mathematics

### Students Enrolled in Selected Course

Name	Grade
Alexander, Carson	A
Anand, Arturo	No grade
Barzdukas, Gytis	B

© 2017 - Contoso University

## Carregamento adiantado, explícito e lento de dados relacionados

Há várias maneiras pelas quais o EF Core pode carregar dados relacionados nas propriedades de navegação de uma entidade:

- **Carregamento adiantado.** O carregamento adiantado é quando uma consulta para um tipo de entidade também carrega entidades relacionadas. Quando a entidade é lida, seus dados relacionados são recuperados. Normalmente, isso resulta em uma única consulta de junção que recupera todos os dados

necessários. O EF Core emitirá várias consultas para alguns tipos de carregamento adiantado. A emissão de várias consultas pode ser mais eficiente do que era o caso para algumas consultas no EF6 quando havia uma única consulta. O carregamento adiantado é especificado com os métodos `Include` e `ThenInclude`.

```
var departments = _context.Departments.Include(d => d.Courses);
foreach (Department d in departments)
{
    foreach(Course c in d.Courses)
    {
        courseList.Add(d.Name + c.Title);
    }
}
```

Query: all Department entities  
and related Course entities

O carregamento adiantado envia várias consultas quando a navegação de coleção é incluída:

- Uma consulta para a consulta principal
- Uma consulta para cada "borda" de coleção na árvore de carregamento.
- Separe consultas com `Load`: os dados podem ser recuperados em consultas separadas e o EF Core "corriga" as propriedades de navegação. "Correção" significa que o EF Core popula automaticamente as propriedades de navegação. A separação de consultas com `Load` é mais parecida com o carregamento explícito do que com o carregamento adiantado.

```
var departments = _context.Departments;
foreach (Department d in departments)
{
    _context.Courses.Where(c => c.DepartmentID == d.DepartmentID).Load();
    foreach (Course c in d.Courses)
    {
        courseList.Add(d.Name + c.Title);
    }
}
```

Query: all Department rows

Query: Course rows related to Department d

Observação: o EF Core corrige automaticamente as propriedades de navegação para outras entidades que foram carregadas anteriormente na instância do contexto. Mesmo se os dados de uma propriedade de navegação *não* foram incluídos de forma explícita, a propriedade ainda pode ser populada se algumas ou todas as entidades relacionadas foram carregadas anteriormente.

- **Carregamento explícito.** Quando a entidade é lida pela primeira vez, os dados relacionados não são recuperados. Um código precisa ser escrito para recuperar os dados relacionados quando eles forem necessários. O carregamento explícito com consultas separadas resulta no envio de várias consultas ao BD. Com o carregamento explícito, o código especifica as propriedades de navegação a serem carregadas. Use o método `Load` para fazer o carregamento explícito. Por exemplo:

```
var departments = _context.Departments;
foreach (Department d in departments)
{
    _context.Entry(d).Collection(p => p.Courses).Load();
    foreach (Course c in d.Courses)
    {
        courseList.Add(d.Name + c.Title);
    }
}
```

Query: all Department rows

Query: Course rows related to Department d

- **Carregamento lento.** O carregamento lento foi adicionado ao EF Core na versão 2.1. Quando a entidade é lida pela primeira vez, os dados relacionados não são recuperados. Na primeira vez que uma propriedade de navegação é acessada, os dados necessários para essa propriedade de navegação são recuperados automaticamente. Uma consulta é enviada para o BD sempre que uma propriedade de navegação é acessada pela primeira vez.
- O operador `Select` carrega somente os dados relacionados necessários.

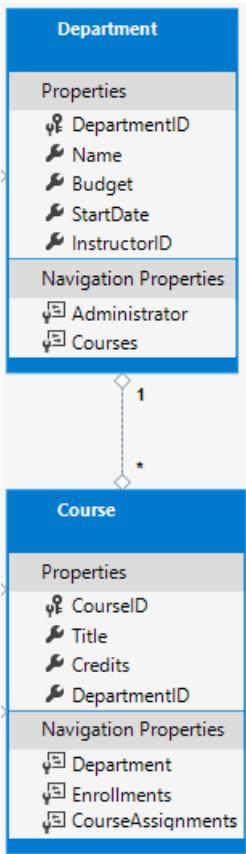
## Criar uma página Course que exibe o nome do departamento

A entidade `Course` inclui uma propriedade de navegação que contém a entidade `Department`. A entidade

`Department` contém o departamento ao qual o curso é atribuído.

Para exibir o nome do departamento atribuído em uma lista de cursos:

- Obtenha a propriedade `Name` da entidade `Department`.
- A entidade `Department` é obtida da propriedade de navegação `Course.Department`.



### Gerar o modelo Curso por scaffolding

- Visual Studio
- CLI do .NET Core

Siga as instruções em [Gere um modelo de aluno por scaffold](#) e use `Course` para a classe de modelo.

O comando anterior gera o modelo `Course` por scaffolding. Abra o projeto no Visual Studio.

Abra `Pages/Courses/Index.cshtml.cs` e examine o método `OnGetAsync`. O mecanismo de scaffolding especificou o carregamento adianteado para a propriedade de navegação `Department`. O método `Include` especifica o carregamento adianteado.

Execute o aplicativo e selecione o link **Cursos**. A coluna de departamento exibe a `DepartmentID`, que não é útil.

Atualize o método `OnGetAsync` pelo seguinte código:

```
public async Task OnGetAsync()
{
    Course = await _context.Courses
        .Include(c => c.Department)
        .AsNoTracking()
        .ToListAsync();
}
```

O código anterior adiciona `AsNoTracking`. `AsNoTracking` melhora o desempenho porque as entidades retornadas não são controladas. As entidades não são controladas porque elas não são atualizadas no contexto atual.

Atualize `Pages/Courses/Index.cshtml` com a seguinte marcação realçada:

```
@page
@model ContosoUniversity.Pages.Courses.IndexModel
 @{
    ViewData["Title"] = "Courses";
}

<h2>Courses</h2>

<p>
    <a asp-page="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Course[0].CourseID)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Course[0].Title)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Course[0].Credits)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Course[0].Department)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model.Course)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.CourseID)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Title)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Credits)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Department.Name)
                </td>
                <td>
                    <a asp-page=".Edit" asp-route-id="@item.CourseID">Edit</a> |
                    <a asp-page=".Details" asp-route-id="@item.CourseID">Details</a> |
                    <a asp-page=".Delete" asp-route-id="@item.CourseID">Delete</a>
                </td>
            </tr>
        }
    </tbody>
</table>
```

As seguintes alterações foram feitas na biblioteca gerada por código em scaffolding:

- Alterou o cabeçalho de Índice para Cursos.
- Adicionou uma coluna **Número** que mostra o valor da propriedade `CourseID`. Por padrão, as chaves primárias não são geradas por scaffolding porque normalmente não têm sentido para os usuários finais. No entanto, nesse caso, a chave primária é significativa.
- Alterou a coluna **Departamento** para que ela exiba o nome de departamento. O código exibe a

propriedade `Name` da entidade `Department` que é carregada na propriedade de navegação `Department`:

```
@Html.DisplayFor(modelItem => item.Department.Name)
```

Execute o aplicativo e selecione a guia **Cursos** para ver a lista com nomes de departamentos.

Number	Title	Credits	Department	
1045	Calculus	4	Mathematics	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
1050	Chemistry	3	Engineering	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
2021	Composition	3	English	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

### Carregando dados relacionados com Select

O método `OnGetAsync` carrega dados relacionados com o método `Include`:

```
public async Task OnGetAsync()
{
    Course = await _context.Courses
        .Include(c => c.Department)
        .AsNoTracking()
        .ToListAsync();
}
```

O operador `Select` carrega somente os dados relacionados necessários. Para itens únicos, como o `Department.Name`, ele usa um SQL INNER JOIN. Para coleções, ele usa outro acesso ao banco de dados, assim como o operador `Include` em coleções.

O seguinte código carrega dados relacionados com o método `Select`:

```
public IList<CourseViewModel> CourseVM { get; set; }

public async Task OnGetAsync()
{
    CourseVM = await _context.Courses
        .Select(p => new CourseViewModel
        {
            CourseID = p.CourseID,
            Title = p.Title,
            Credits = p.Credits,
            DepartmentName = p.Department.Name
        }).ToListAsync();
}
```

O `CourseViewModel`:

```
public class CourseViewModel
{
    public int CourseID { get; set; }
    public string Title { get; set; }
    public int Credits { get; set; }
    public string DepartmentName { get; set; }
}
```

Consulte [IndexSelect.cshtml](#) e [IndexSelect.cshtml.cs](#) para obter um exemplo completo.

## Criar uma página Instrutores que mostra Cursos e Registros

Nesta seção, a página Instrutores é criada.

Instructors - Contoso University

localhost:1234/Instructors/3?courseID=1050&action=OnGetAsync

## Instructors

Create New

Last Name	First Name	Hire Date	Office	Courses	
Abercrombie	Kim	1995-03-11		2021 Composition 2042 Literature	<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Fakhouri	Fadi	2002-07-06	Smith 17	1045 Calculus	<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Harui	Roger	1998-07-01	Gowan 27	1050 Chemistry 3141 Trigonometry	<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Kapoor	Candace	2001-01-15	Thompson 304	1050 Chemistry	<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Zheng	Roger	2004-02-12		4022 Microeconomics 4041 Macroeconomics	<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

## Courses Taught by Selected Instructor

	Number	Title	Department
<a href="#">Select</a>	1050	Chemistry	Engineering
<a href="#">Select</a>	3141	Trigonometry	Mathematics

## Students Enrolled in Selected Course

Name	Grade
Alexander, Carson	A
Anand, Arturo	No grade
Barzdukas, Gytis	B

© 2017 - Contoso University

Essa página lê e exibe dados relacionados das seguintes maneiras:

- A lista de instrutores exibe dados relacionados da entidade `OfficeAssignment` (Office na imagem anterior). As entidades `Instructor` e `OfficeAssignment` estão em uma relação um para zero ou um. O carregamento adianteado é usado para as entidades `OfficeAssignment`. O carregamento adianteado costuma ser mais eficiente quando os dados relacionados precisam ser exibidos. Nesse caso, as atribuições de escritório para os instrutores são exibidas.
- Quando o usuário seleciona um instrutor (Pedro na imagem anterior), as entidades `Course` relacionadas são

exibidas. As entidades `Instructor` e `Course` estão em uma relação muitos para muitos. O carregamento adiantado é usado para entidades `Course` e suas entidades `Department` relacionadas. Nesse caso, consultas separadas podem ser mais eficientes porque somente os cursos para o instrutor selecionado são necessários. Este exemplo mostra como usar o carregamento adiantado para propriedades de navegação em entidades que estão nas propriedades de navegação.

- Quando o usuário seleciona um curso (Química na imagem anterior), os dados relacionados da entidade `Enrollments` são exibidos. Na imagem anterior, o nome do aluno e a nota são exibidos. As entidades `Course` e `Enrollment` estão em uma relação um-para-muitos.

## Criar um modelo de exibição para a exibição Índice de Instrutor

A página Instrutores mostra dados de três tabelas diferentes. É criado um modelo de exibição que inclui as três entidades que representam as três tabelas.

Na pasta `SchoolViewModels`, crie `InstructorIndexData.cs` com o seguinte código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ContosoUniversity.Models.SchoolViewModels
{
    public class InstructorIndexData
    {
        public IEnumerable<Instructor> Instructors { get; set; }
        public IEnumerable<Course> Courses { get; set; }
        public IEnumerable<Enrollment> Enrollments { get; set; }
    }
}
```

## Gerar o modelo Instrutor por scaffolding

- [Visual Studio](#)
- [CLI do .NET Core](#)

Siga as instruções em [Gere um modelo de aluno por scaffold](#) e use `Instructor` para a classe de modelo.

O comando anterior gera o modelo `Instructor` por scaffolding. Execute o aplicativo e navegue para a página Instrutores.

Substitua `Pages/Instructors/Index.cshtml.cs` pelo seguinte código:

```

using ContosoUniversity.Models;
using ContosoUniversity.Models.SchoolViewModels; // Add VM
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using System.Threading.Tasks;

namespace ContosoUniversity.Pages.Instructors
{
    public class IndexModel : PageModel
    {
        private readonly ContosoUniversity.Data.SchoolContext _context;

        public IndexModel(ContosoUniversity.Data.SchoolContext context)
        {
            _context = context;
        }

        public InstructorIndexData Instructor { get; set; }
        public int InstructorID { get; set; }

        public async Task OnGetAsync(int? id)
        {
            Instructor = new InstructorIndexData();
            Instructor.Instructors = await _context.Instructors
                .Include(i => i.OfficeAssignment)
                .Include(i => i.CourseAssignments)
                .ThenInclude(i => i.Course)
                .AsNoTracking()
                .OrderBy(i => i.LastName)
                .ToListAsync();

            if (id != null)
            {
                InstructorID = id.Value;
            }
        }
    }
}

```

O método `OnGetAsync` aceita dados de rota opcionais para a ID do instrutor selecionado.

Examine a consulta no arquivo *Pages/Instructors/Index.cshtml*:

```

Instructor.Instructors = await _context.Instructors
    .Include(i => i.OfficeAssignment)
    .Include(i => i.CourseAssignments)
    .ThenInclude(i => i.Course)
    .AsNoTracking()
    .OrderBy(i => i.LastName)
    .ToListAsync();

```

A consulta tem duas inclusões:

- `OfficeAssignment`: exibido na [exibição de instrutores](#).
- `CourseAssignments`: que exibe os cursos ministrados.

## Atualizar a página Índice de instrutores

Atualize *Pages/Instructors/Index.cshtml* com a seguinte marcação:

```

@page "{id:int?}"
@model ContosoUniversity.Pages.Instructors.IndexModel

 @{
     ViewData["Title"] = "Instructors";
 }

<h2>Instructors</h2>

<p>
    <a asp-page="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>Last Name</th>
            <th>First Name</th>
            <th>Hire Date</th>
            <th>Office</th>
            <th>Courses</th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model.Instructor.Instructors)
        {
            string selectedRow = "";
            if (item.ID == Model.InstructorID)
            {
                selectedRow = "success";
            }
            <tr class="@selectedRow">
                <td>
                    @Html.DisplayFor(modelItem => item.LastName)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.FirstMidName)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.HireDate)
                </td>
                <td>
                    @if (item.OfficeAssignment != null)
                    {
                        @item.OfficeAssignment.Location
                    }
                </td>
                <td>
                    @{
                        foreach (var course in item.CourseAssignments)
                        {
                            @course.Course.CourseID @: @course.Course.Title <br />
                        }
                    }
                </td>
                <td>
                    <a asp-page=".Index" asp-route-id="@item.ID">Select</a> |
                    <a asp-page=".Edit" asp-route-id="@item.ID">Edit</a> |
                    <a asp-page=".Details" asp-route-id="@item.ID">Details</a> |
                    <a asp-page=".Delete" asp-route-id="@item.ID">Delete</a>
                </td>
            </tr>
        }
    </tbody>
</table>

```

A marcação anterior faz as seguintes alterações:

- Atualiza a diretiva `page` de `@page` para `@page "{id:int?}"`. `"{id:int?}"` é um modelo de rota. O modelo de rota altera cadeias de consulta de inteiro na URL para dados de rota. Por exemplo, clicar no link **Selecionar** de um o instrutor apenas com a diretiva `@page` produz uma URL semelhante à seguinte:

```
http://localhost:1234/Instructors?id=2
```

Quando a diretiva de página é `@page "{id:int?}"`, a URL anterior é:

```
http://localhost:1234/Instructors/2
```

- O título de página é **Instrutores**.
- Adicionou uma coluna **Office** que exibe `item.OfficeAssignment.Location` somente se `item.OfficeAssignment` não é nulo. Como essa é uma relação um para zero ou um, pode não haver uma entidade `OfficeAssignment` relacionada.

```
@if (item.OfficeAssignment != null)
{
    @item.OfficeAssignment.Location
}
```

- Adicionou uma coluna **Courses** que exibe os cursos ministrados por cada instrutor. Consulte [Transição de linha explícita com @:](#) para obter mais informações sobre essa sintaxe Razor.
- Adicionou um código que adiciona `class="success"` dinamicamente ao elemento `tr` do instrutor selecionado. Isso define uma cor da tela de fundo para a linha selecionada usando uma classe Bootstrap.

```
string selectedRow = "";
if (item.CourseID == Model.CourseID)
{
    selectedRow = "success";
}
<tr class="@selectedRow">
```

- Adicionou um novo hiperlink rotulado **Selecionar**. Este link envia a ID do instrutor selecionado para o método `Index` e define uma cor da tela de fundo.

```
<a asp-action="Index" asp-route-id="@item.ID">Select</a> |
```

Execute o aplicativo e selecione a guia **Instrutores**. A página exibe o `Location` (escritório) da entidade `OfficeAssignment` relacionada. Se `OfficeAssignment` é nulo, uma célula de tabela vazia é exibida.

Last Name	First Name	Hire Date	Office	Courses
Abercrombie	Kim	1995-03-11		2021 Composition 2042 Literature
Fakhouri	Fadi	2002-07-06	Smith 17	1045 Calculus

Clique no link **Selecionar**. O estilo de linha é alterado.

#### Adicionar cursos ministrados pelo instrutor selecionado

Atualize o método `OnGetAsync` em `Pages/Instructors/Index.cshtml.cs` com o seguinte código:

```
public async Task OnGetAsync(int? id, int? courseID)
{
    Instructor = new InstructorIndexData();
    Instructor.Instructors = await _context.Instructors
        .Include(i => i.OfficeAssignment)
        .Include(i => i.CourseAssignments)
        .ThenInclude(i => i.Course)
        .ThenInclude(i => i.Department)
        .AsNoTracking()
        .OrderBy(i => i.LastName)
        .ToListAsync();

    if (id != null)
    {
        InstructorID = id.Value;
        Instructor instructor = Instructor.Instructors.Where(
            i => i.ID == id.Value).Single();
        Instructor.Courses = instructor.CourseAssignments.Select(s => s.Course);
    }

    if (courseID != null)
    {
        CourseID = courseID.Value;
        Instructor.Enrollments = Instructor.Courses.Where(
            x => x.CourseID == courseID).Single().Enrollments;
    }
}
```

Adicione `public int CourseID { get; set; }`

```

public class IndexModel : PageModel
{
    private readonly ContosoUniversity.Data.SchoolContext _context;

    public IndexModel(ContosoUniversity.Data.SchoolContext context)
    {
        _context = context;
    }

    public InstructorIndexData Instructor { get; set; }
    public int InstructorID { get; set; }
    public int CourseID { get; set; }

    public async Task OnGetAsync(int? id, int? courseID)
    {
        Instructor = new InstructorIndexData();
        Instructor.Instructors = await _context.Instructors
            .Include(i => i.OfficeAssignment)
            .Include(i => i.CourseAssignments)
            .ThenInclude(i => i.Course)
            .ThenInclude(i => i.Department)
            .AsNoTracking()
            .OrderBy(i => i.LastName)
            .ToListAsync();

        if (id != null)
        {
            InstructorID = id.Value;
            Instructor.instructor = Instructor.Instructors.Where(
                i => i.ID == id.Value).Single();
            Instructor.Courses = instructor.CourseAssignments.Select(s => s.Course);
        }

        if (courseID != null)
        {
            CourseID = courseID.Value;
            Instructor.Enrollments = Instructor.Courses.Where(
                x => x.CourseID == courseID).Single().Enrollments;
        }
    }
}

```

Examine a consulta atualizada:

```

Instructor.Instructors = await _context.Instructors
    .Include(i => i.OfficeAssignment)
    .Include(i => i.CourseAssignments)
    .ThenInclude(i => i.Course)
    .ThenInclude(i => i.Department)
    .AsNoTracking()
    .OrderBy(i => i.LastName)
    .ToListAsync();

```

A consulta anterior adiciona as entidades `Department`.

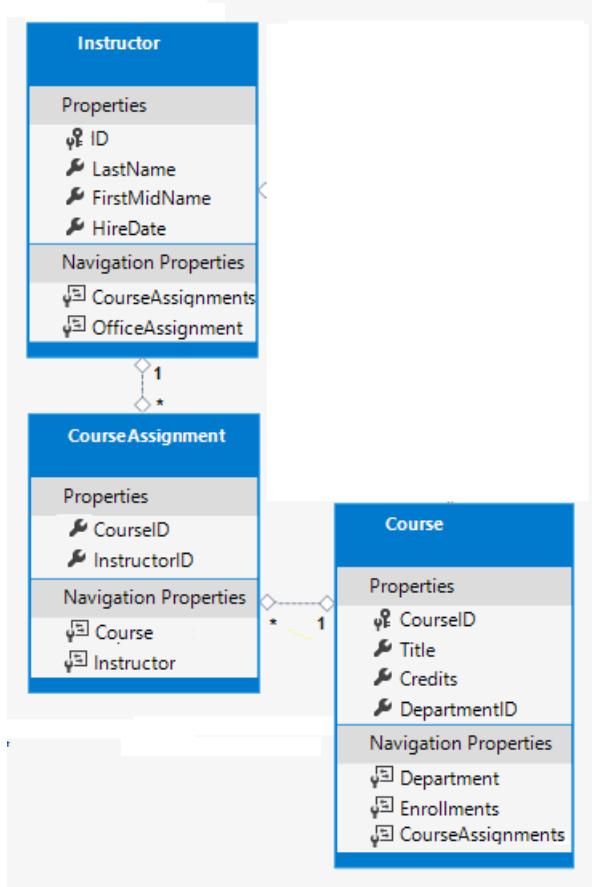
O código a seguir é executado quando o instrutor é selecionado (`id != null`). O instrutor selecionado é recuperado da lista de instrutores no modelo de exibição. Em seguida, a propriedade `Courses` do modelo de exibição é carregada com as entidades `Course` da propriedade de navegação `CourseAssignments` desse instrutor.

```

if (id != null)
{
    InstructorID = id.Value;
    Instructor instructor = Instructor.Instructors.Where(
        i => i.ID == id.Value).Single();
    Instructor.Courses = instructor.CourseAssignments.Select(s => s.Course);
}

```

O método `Where` retorna uma coleção. No método `Where` anterior, uma única entidade `Instructor` é retornada. O método `Single` converte a coleção em uma única entidade `Instructor`. A entidade `Instructor` fornece acesso à propriedade `CourseAssignments`. `CourseAssignments` fornece acesso às entidades `Course` relacionadas.



O método `Single` é usado em uma coleção quando a coleção tem apenas um item. O método `single` gera uma exceção se a coleção está vazia ou se há mais de um item. Uma alternativa é `SingleOrDefault`, que retorna um valor padrão (nulo, nesse caso) se a coleção estiver vazia. O uso de `SingleOrDefault` é uma coleção vazia:

- Resulta em uma exceção (da tentativa de encontrar uma propriedade `Courses` em uma referência nula).
- A mensagem de exceção indica menos claramente a causa do problema.

O seguinte código popula a propriedade `Enrollments` do modelo de exibição quando um curso é selecionado:

```

if (courseID != null)
{
    CourseID = courseID.Value;
    Instructor.Enrollments = Instructor.Courses.Where(
        x => x.CourseID == courseID).Single().Enrollments;
}

```

Adicione a seguinte marcação ao final do Razor Page `Pages/Courses/Index.cshtml`:

```

        <a href="#" asp-page="./Delete" asp-route-id="@item.ID">Delete</a>
    </td>
    </tr>
}
</tbody>
</table>

@if (Model.Instructor.Courses != null)
{
    <h3>Courses Taught by Selected Instructor</h3>
    <table class="table">
        <tr>
            <th></th>
            <th>Number</th>
            <th>Title</th>
            <th>Department</th>
        </tr>

@foreach (var item in Model.Instructor.Courses)
{
    string selectedRow = "";
    if (item.CourseID == Model.CourseID)
    {
        selectedRow = "success";
    }
    <tr class="@selectedRow">
        <td>
            <a href="#" asp-page="./Index" asp-route-courseID="@item.CourseID">Select</a>
        </td>
        <td>
            @item.CourseID
        </td>
        <td>
            @item.Title
        </td>
        <td>
            @item.Department.Name
        </td>
    </tr>
}

</table>
}

```

A marcação anterior exibe uma lista de cursos relacionados a um instrutor quando um instrutor é selecionado.

Teste o aplicativo. Clique em um link **Selecionar** na página Instrutores.

Last Name	First Name	Hire Date	Office	Courses
Abercrombie	Kim	1995-03-11		2021 Composition 2042 Literature
Fakhouri	Fadi	2002-07-06	Smith 17	1045 Calculus

## Courses Taught by Selected Instructor

	Number	Title	Department
Select	2021	Composition	English
Select	2042	Literature	English

## Mostrar dados de alunos

Nesta seção, o aplicativo é atualizado para mostrar os dados de alunos de um curso selecionado.

Atualize a consulta no método `OnGetAsync` em `Pages/Instructors/Index.cshtml.cs` com o seguinte código:

```
Instructor.Instructors = await _context.Instructors
    .Include(i => i.OfficeAssignment)
    .Include(i => i.CourseAssignments)
    .ThenInclude(i => i.Course)
    .ThenInclude(i => i.Department)
    .Include(i => i.CourseAssignments)
    .ThenInclude(i => i.Course)
    .ThenInclude(i => i.Enrollments)
    .ThenInclude(i => i.Student)
    .AsNoTracking()
    .OrderBy(i => i.LastName)
    .ToListAsync();
```

Atualize `Pages/Instructors/Index.cshtml`. Adicione a seguinte marcação ao final do arquivo:

```
@if (Model.Instructor.Enrollments != null)
{
    <h3>
        Students Enrolled in Selected Course
    </h3>
    <table class="table">
        <tr>
            <th>Name</th>
            <th>Grade</th>
        </tr>
        @foreach (var item in Model.Instructor.Enrollments)
        {
            <tr>
                <td>
                    @item.Student.FullName
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Grade)
                </td>
            </tr>
        }
    </table>
}
```

A marcação anterior exibe uma lista dos alunos registrados no curso selecionado.

Atualize a página e selecione um instrutor. Selecione um curso para ver a lista de alunos registrados e suas notas.

Instructors - Contoso University

localhost:1234/Instructors/3?courseID=1050&action=OnGetAsync

Contoso University

## Instructors

Create New

Last Name	First Name	Hire Date	Office	Courses	
Abercrombie	Kim	1995-03-11		2021 Composition 2042 Literature	Select   Edit   Details   Delete
Fakhouri	Fadi	2002-07-06	Smith 17	1045 Calculus	Select   Edit   Details   Delete
Harui	Roger	1998-07-01	Gowan 27	1050 Chemistry 3141 Trigonometry	Select   Edit   Details   Delete
Kapoor	Candace	2001-01-15	Thompson 304	1050 Chemistry	Select   Edit   Details   Delete
Zheng	Roger	2004-02-12		4022 Microeconomics 4041 Macroeconomics	Select   Edit   Details   Delete

### Courses Taught by Selected Instructor

	Number	Title	Department
Select	1050	Chemistry	Engineering
Select	3141	Trigonometry	Mathematics

### Students Enrolled in Selected Course

Name	Grade
Alexander, Carson	A
Anand, Arturo	No grade
Barzdukas, Gytis	B

© 2017 - Contoso University

## Usando Single

O método `Single` pode passar a condição `Where` em vez de chamar o método `Where` separadamente:

```

public async Task OnGetAsync(int? id, int? courseID)
{
    Instructor = new InstructorIndexData();

    Instructor.Instructors = await _context.Instructors
        .Include(i => i.OfficeAssignment)
        .Include(i => i.CourseAssignments)
        .ThenInclude(i => i.Course)
            .ThenInclude(i => i.Department)
        .Include(i => i.CourseAssignments)
            .ThenInclude(i => i.Course)
                .ThenInclude(i => i.Enrollments)
                    .ThenInclude(i => i.Student)
    .AsNoTracking()
    .OrderBy(i => i.LastName)
    .ToListAsync();

    if (id != null)
    {
        InstructorID = id.Value;
        Instructor instructor = Instructor.Instructors.Single(
            i => i.ID == id.Value);
        Instructor.Courses = instructor.CourseAssignments.Select(
            s => s.Course);
    }

    if (courseID != null)
    {
        CourseID = courseID.Value;
        Instructor.Enrollments = Instructor.Courses.Single(
            x => x.CourseID == courseID).Enrollments;
    }
}

```

A abordagem `Single` anterior não oferece nenhum benefício em relação ao uso de `Where`. Alguns desenvolvedores preferem o estilo de abordagem `Single`.

## Carregamento explícito

O código atual especifica o carregamento adiantado para `Enrollments` e `Students`:

```

Instructor.Instructors = await _context.Instructors
    .Include(i => i.OfficeAssignment)
    .Include(i => i.CourseAssignments)
    .ThenInclude(i => i.Course)
        .ThenInclude(i => i.Department)
    .Include(i => i.CourseAssignments)
        .ThenInclude(i => i.Course)
            .ThenInclude(i => i.Enrollments)
                .ThenInclude(i => i.Student)
    .AsNoTracking()
    .OrderBy(i => i.LastName)
    .ToListAsync();

```

Suponha que os usuários raramente desejem ver registros em um curso. Nesse caso, uma otimização será carregar apenas os dados de registro se eles forem solicitados. Nesta seção, o `OnGetAsync` é atualizado para usar o carregamento explícito de `Enrollments` e `Students`.

Atualize o `OnGetAsync` com o seguinte código:

```

public async Task OnGetAsync(int? id, int? courseID)
{
    Instructor = new InstructorIndexData();
    Instructor.Instructors = await _context.Instructors
        .Include(i => i.OfficeAssignment)
        .Include(i => i.CourseAssignments)
        .ThenInclude(i => i.Course)
        .ThenInclude(i => i.Department)
        // .Include(i => i.CourseAssignments)
        //     .ThenInclude(i => i.Course)
        //         .ThenInclude(i => i.Enrollments)
        //             .ThenInclude(i => i.Student)
        // .AsNoTracking()
        .OrderBy(i => i.LastName)
        .ToListAsync();
}

if (id != null)
{
    InstructorID = id.Value;
    Instructor instructor = Instructor.Instructors.Where(
        i => i.ID == id.Value).Single();
    Instructor.Courses = instructor.CourseAssignments.Select(s => s.Course);
}

if (courseID != null)
{
    CourseID = courseID.Value;
    var selectedCourse = Instructor.Courses.Where(x => x.CourseID == courseID).Single();
    await _context.Entry(selectedCourse).Collection(x => x.Enrollments).LoadAsync();
    foreach (Enrollment enrollment in selectedCourse.Enrollments)
    {
        await _context.Entry(enrollment).Reference(x => x.Student).LoadAsync();
    }
    Instructor.Enrollments = selectedCourse.Enrollments;
}
}

```

O código anterior remove as chamadas do método `ThenInclude` para dados de registro e de alunos. Se um curso é selecionado, o código realçado recupera:

- As entidades `Enrollment` para o curso selecionado.
- As entidades `Student` para cada `Enrollment`.

Observe que o código anterior comenta `.AsNoTracking()`. As propriedades de navegação apenas podem ser carregadas de forma explícita para entidades controladas.

Teste o aplicativo. De uma perspectiva dos usuários, o aplicativo se comporta de forma idêntica à versão anterior.

O próximo tutorial mostra como atualizar os dados relacionados.

[ANTERIOR](#)

[PRÓXIMO](#)

# Páginas Razor com o EF Core no ASP.NET Core – Atualizar dados relacionados – 7 de 8

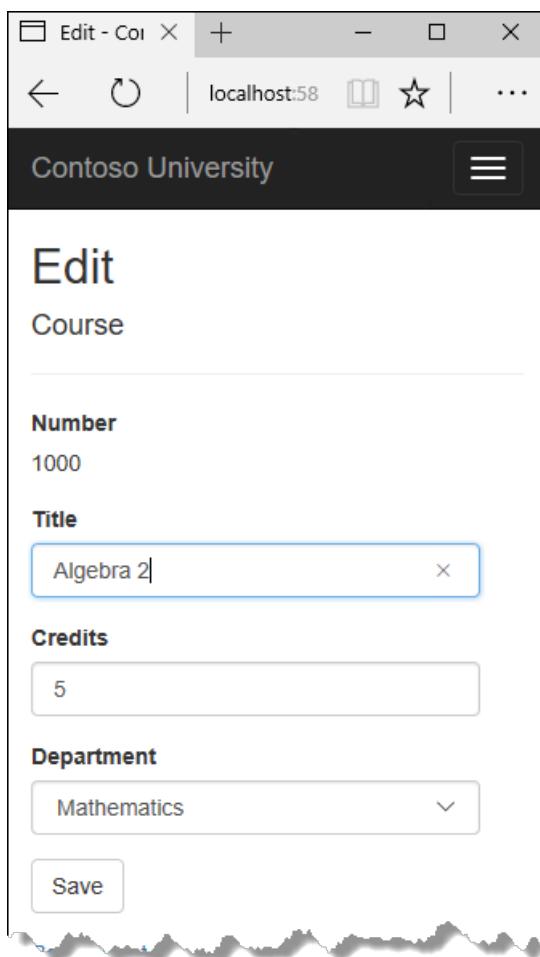
30/10/2018 • 23 minutes to read • [Edit Online](#)

Por [Tom Dykstra](#) e [Rick Anderson](#)

O aplicativo Web Contoso University demonstra como criar aplicativos Web das Páginas do Razor usando o EF Core e o Visual Studio. Para obter informações sobre a série de tutoriais, consulte [o primeiro tutorial](#).

Este tutorial demonstra como atualizar dados relacionados. Caso tenha problemas que não consiga resolver, [baixe ou exiba o aplicativo concluído](#). [Instruções de download](#).

As ilustrações a seguir mostram algumas das páginas concluídas.



The screenshot shows a web browser window titled "Edit - Contoso Universit". The address bar displays "localhost:5813/Instruct". The main content area is titled "Edit" and "Instructor". It contains several input fields: "Last Name" (Abercrombie), "First Name" (Kim), "Hire Date" (3/11/1995), and "Office Location" (44/3P). Below these are checkboxes for course selection, grouped into three columns. The first column has checkboxes for "1000 Algebra 2" and "4022 Microeconomics". The second column has checkboxes for "1045 Calculus" and "2042 Literature". The third column has checkboxes for "1050 Chemistry" and "3141 Trigonometry". The checkbox for "2021 Composition" is checked. At the bottom left is a "Save" button.

<input type="checkbox"/> 1000 Algebra 2	<input type="checkbox"/> 1045 Calculus	<input type="checkbox"/> 1050 Chemistry
<input checked="" type="checkbox"/> 2021 Composition	<input checked="" type="checkbox"/> 2042 Literature	<input type="checkbox"/> 3141 Trigonometry
<input type="checkbox"/> 4022 Microeconomics	<input type="checkbox"/> 4041 Macroeconomics	

Examine e teste as páginas de cursos Criar e Editar. Crie um novo curso. O departamento é selecionado por sua chave primária (um inteiro), não pelo nome. Edite o novo curso. Quando concluir o teste, exclua o novo curso.

## Criar uma classe base para compartilhar um código comum

As páginas Cursos/Criar e Cursos/Editar precisam de uma lista de nomes de departamentos. Crie a classe base *Pages/Courses/DepartmentNamePageModel.cshtml.cs* para as páginas Criar e Editar:

```

using ContosoUniversity.Data;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using System.Linq;

namespace ContosoUniversity.Pages.Courses
{
    public class DepartmentNamePageModel : PageModel
    {
        public SelectList DepartmentNameSL { get; set; }

        public void PopulateDepartmentsDropDownList(SchoolContext _context,
            object selectedDepartment = null)
        {
            var departmentsQuery = from d in _context.Departments
                orderby d.Name // Sort by name.
                select d;

            DepartmentNameSL = new SelectList(departmentsQuery.AsNoTracking(),
                "DepartmentID", "Name", selectedDepartment);
        }
    }
}

```

O código anterior cria uma `SelectList` para conter a lista de nomes de departamentos. Se `selectedDepartment` for especificado, esse departamento estará selecionado na `SelectList`.

As classes de modelo da página Criar e Editar serão derivadas de `DepartmentNamePageModel`.

## Personalizar as páginas Courses

Quando uma nova entidade de curso é criada, ela precisa ter uma relação com um departamento existente. Para adicionar um departamento durante a criação de um curso, a classe base para Create e Edit contém uma lista suspensa para seleção do departamento. A lista suspensa define a propriedade `Course.DepartmentID` de FK (chave estrangeira). O EF Core usa a `Course.DepartmentID` FK para carregar a propriedade de navegação `Department`.

Create DepartmentName

localhost:1234/Courses/Create

Contoso University

## Create

Course

**Number**

**Title**

**Credits**

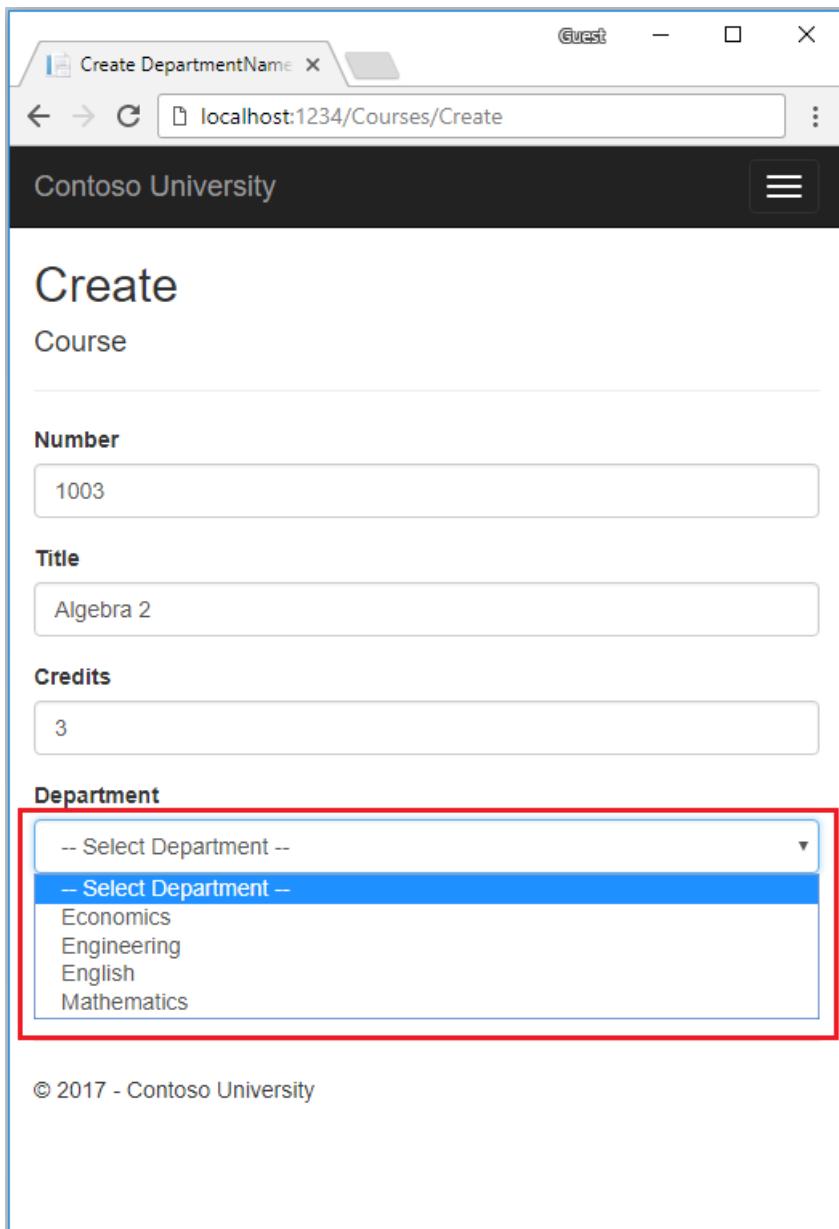
**Department**

-- Select Department --

-- Select Department --

- Economics
- Engineering
- English
- Mathematics

© 2017 - Contoso University



Atualize o modelo da página Criar com o seguinte código:

```

using ContosoUniversity.Models;
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;

namespace ContosoUniversity.Pages.Courses
{
    public class CreateModel : DepartmentNamePageModel
    {
        private readonly ContosoUniversity.Data.SchoolContext _context;

        public CreateModel(ContosoUniversity.Data.SchoolContext context)
        {
            _context = context;
        }

        public IActionResult OnGet()
        {
            PopulateDepartmentsDropDownList(_context);
            return Page();
        }

        [BindProperty]
        public Course Course { get; set; }

        public async Task<IActionResult> OnPostAsync()
        {
            if (!ModelState.IsValid)
            {
                return Page();
            }

            var emptyCourse = new Course();

            if (await TryUpdateModelAsync<Course>(
                emptyCourse,
                "course", // Prefix for form value.
                s => s.CourseID, s => s.DepartmentID, s => s.Title, s => s.Credits))
            {
                _context.Courses.Add(emptyCourse);
                await _context.SaveChangesAsync();
                return RedirectToPage("./Index");
            }

            // Select DepartmentID if TryUpdateModelAsync fails.
            PopulateDepartmentsDropDownList(_context, emptyCourse.DepartmentID);
            return Page();
        }
    }
}

```

O código anterior:

- Deriva de `DepartmentNamePageModel`.
- Usa `TryUpdateModelAsync` para impedir o [excesso de postagem](#).
- Substitui `ViewData["DepartmentID"]` por `DepartmentNameSL` (da classe base).

`ViewData["DepartmentID"]` é substituído pelo `DepartmentNameSL` fortemente tipado. Modelos fortemente tipados são preferíveis aos fracamente tipados. Para obter mais informações, consulte [Dados fracamente tipados \(ViewData e ViewBag\)](#).

## Atualizar a página Criar Cursos

Atualize `Pages/Courses/Create.cshtml` com a seguinte marcação:

```

@page
@model ContosoUniversity.Pages.Courses.CreateModel
 @{
     ViewData["Title"] = "Create Course";
 }
<h2>Create</h2>
<h4>Course</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Course.CourseID" class="control-label"></label>
                <input asp-for="Course.CourseID" class="form-control" />
                <span asp-validation-for="Course.CourseID" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Course.Title" class="control-label"></label>
                <input asp-for="Course.Title" class="form-control" />
                <span asp-validation-for="Course.Title" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Course.Credits" class="control-label"></label>
                <input asp-for="Course.Credits" class="form-control" />
                <span asp-validation-for="Course.Credits" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Course.Department" class="control-label"></label>
                <select asp-for="Course.DepartmentID" class="form-control" asp-items="@Model.DepartmentNameSL">
                    <option value="">-- Select Department --</option>
                </select>
                <span asp-validation-for="Course.DepartmentID" class="text-danger" />
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-default" />
            </div>
        </form>
    </div>
</div>
<a asp-page="Index">Back to List</a>
</div>
@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}

```

A marcação anterior faz as seguintes alterações:

- Altera a legenda de **DepartmentID** para **Departamento**.
- Substitui `"ViewBag.DepartmentID"` por `DepartmentNameSL` (da classe base).
- Adiciona a opção "Selecionar Departamento". Essa alteração renderiza "Selecionar Departamento", em vez do departamento primeiro.
- Adiciona uma mensagem de validação quando o departamento não está selecionado.

A Página do Razor usa a opção **Selecionar Auxiliar de Marcação**:

```
<div class="form-group">
    <label asp-for="Course.Department" class="control-label"></label>
    <select asp-for="Course.DepartmentID" class="form-control"
        asp-items="@Model.DepartmentNameSL">
        <option value="">-- Select Department --</option>
    </select>
    <span asp-validation-for="Course.DepartmentID" class="text-danger" />
</div>
```

Teste a página Criar. A página Criar exibe o nome do departamento em vez de a ID do departamento.

**Atualize a página Editar Cursos.**

Atualize o modelo de página de edição com o seguinte código:

```

using ContosoUniversity.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Threading.Tasks;

namespace ContosoUniversity.Pages.Courses
{
    public class EditModel : DepartmentNamePageModel
    {
        private readonly ContosoUniversity.Data.SchoolContext _context;

        public EditModel(ContosoUniversity.Data.SchoolContext context)
        {
            _context = context;
        }

        [BindProperty]
        public Course Course { get; set; }

        public async Task<IActionResult> OnGetAsync(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            Course = await _context.Courses
                .Include(c => c.Department).FirstOrDefaultAsync(m => m.CourseID == id);

            if (Course == null)
            {
                return NotFound();
            }

            // Select current DepartmentID.
            PopulateDepartmentsDropDownList(_context, Course.DepartmentID);
            return Page();
        }

        public async Task<IActionResult> OnPostAsync(int? id)
        {
            if (!ModelState.IsValid)
            {
                return Page();
            }

            var courseToUpdate = await _context.Courses.FindAsync(id);

            if (await TryUpdateModelAsync<Course>(
                courseToUpdate,
                "course", // Prefix for form value.
                c => c.Credits, c => c.DepartmentID, c => c.Title))
            {
                await _context.SaveChangesAsync();
                return RedirectToPage("./Index");
            }

            // Select DepartmentID if TryUpdateModelAsync fails.
            PopulateDepartmentsDropDownList(_context, courseToUpdate.DepartmentID);
            return Page();
        }
    }
}

```

As alterações são semelhantes às feitas no modelo da página Criar. No código anterior,

`PopulateDepartmentsDropDownList` passa a ID do departamento, que seleciona o departamento especificado na lista

suspensa.

Atualize *Pages/Courses/Edit.cshtml* com a seguinte marcação:

```
@page
@model ContosoUniversity.Pages.Courses.EditModel

@{
    ViewData["Title"] = "Edit";
}



## Edit



#### Course



---



<form method="post">
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
    <input type="hidden" asp-for="Course.CourseID" />
    <div class="form-group">
        <label asp-for="Course.CourseID" class="control-label"></label>
        <div>@Html.DisplayFor(model => model.Course.CourseID)</div>
    </div>
    <div class="form-group">
        <label asp-for="Course.Title" class="control-label"></label>
        <input asp-for="Course.Title" class="form-control" />
        <span asp-validation-for="Course.Title" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Course.Credits" class="control-label"></label>
        <input asp-for="Course.Credits" class="form-control" />
        <span asp-validation-for="Course.Credits" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Course.Department" class="control-label"></label>
        <select asp-for="Course.DepartmentID" class="form-control" asp-items="@Model.DepartmentNameSL"></select>
        <span asp-validation-for="Course.DepartmentID" class="text-danger"></span>
    </div>
    <div class="form-group">
        <input type="submit" value="Save" class="btn btn-default" />
    </div>
</form>


</div>

<div>
    <a asp-page=".~/Index">Back to List</a>
</div>

@section Scripts {
    @await Html.RenderPartialAsync("_ValidationScriptsPartial")
}


```

A marcação anterior faz as seguintes alterações:

- Exibe a ID do curso. Geralmente, a PK (chave primária) de uma entidade não é exibida. Em geral, PKs não têm sentido para os usuários. Nesse caso, o PK é o número do curso.
- Altera a legenda de **DepartmentID** para **Departamento**.
- Substitui `"ViewBag.DepartmentID"` por `DepartmentNameSL` (da classe base).

A página contém um campo oculto (`<input type="hidden">`) para o número do curso. A adição de um auxiliar de marcação `<label>` com `asp-for="Course.CourseID"` não elimina a necessidade do campo oculto.

`<input type="hidden">` é necessário para que o número seja incluído nos dados postados quando o usuário clicar em **Salvar**.

Teste o código atualizado. Crie, edite e exclua um curso.

## Adicionar `AsNoTracking` aos modelos de página Detalhes e Excluir

`AsNoTracking` pode melhorar o desempenho quando o acompanhamento não é necessário. Adicione `AsNoTracking` ao modelo de página Excluir e Detalhes. O seguinte código mostra o modelo de página Excluir atualizado:

```
public class DeleteModel : PageModel
{
    private readonly ContosoUniversity.Data.SchoolContext _context;

    public DeleteModel(ContosoUniversity.Data.SchoolContext context)
    {
        _context = context;
    }

    [BindProperty]
    public Course Course { get; set; }

    public async Task<IActionResult> OnGetAsync(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        Course = await _context.Courses
            .AsNoTracking()
            .Include(c => c.Department)
            .FirstOrDefaultAsync(m => m.CourseID == id);

        if (Course == null)
        {
            return NotFound();
        }
        return Page();
    }

    public async Task<IActionResult> OnPostAsync(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        Course = await _context.Courses
            .AsNoTracking()
            .FirstOrDefaultAsync(m => m.CourseID == id);

        if (Course != null)
        {
            _context.Courses.Remove(Course);
            await _context.SaveChangesAsync();
        }

        return RedirectToPage("./Index");
    }
}
```

Atualize o método `OnGetAsync` no arquivo *Pages/Courses/Details.cshtml.cs*:

```
public async Task<IActionResult> OnGetAsync(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    Course = await _context.Courses
        .AsNoTracking()
        .Include(c => c.Department)
        .FirstOrDefaultAsync(m => m.CourseID == id);

    if (Course == null)
    {
        return NotFound();
    }
    return Page();
}
```

## Modificar as páginas Excluir e Detalhes

Atualize a página Excluir do Razor com a seguinte marcação:

```

@page
@model ContosoUniversity.Pages.Courses.DeleteModel

 @{
     ViewData["Title"] = "Delete";
 }

<h2>Delete</h2>

<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>Course</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Course.CourseID)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Course.CourseID)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Course.Title)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Course.Title)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Course.Credits)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Course.Credits)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Course.Department)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Course.DepartmentID)
        </dd>
    </dl>

    <form method="post">
        <input type="hidden" asp-for="Course.CourseID" />
        <input type="submit" value="Delete" class="btn btn-default" /> |
        <a asp-page="./Index">Back to List</a>
    </form>
</div>

```

Faça as mesmas alterações na página Detalhes.

## Testar as páginas Curso

Teste as páginas Criar, Editar, Detalhes e Excluir.

## Atualizar as páginas do instrutor

As seções a seguir atualizam as páginas do instrutor.

### Adicionar local do escritório

Ao editar um registro de instrutor, é recomendável atualizar a atribuição de escritório do instrutor. A entidade `Instructor` tem uma relação um para zero ou um com a entidade `OfficeAssignment`. O código do instrutor precisa manipular:

- Se o usuário limpar a atribuição de escritório, exclua a entidade `OfficeAssignment`.
- Se o usuário inserir uma atribuição de escritório e ela estiver vazia, crie uma nova entidade `OfficeAssignment`.

- Se o usuário alterar a atribuição de escritório, atualize a entidade `OfficeAssignment`.

Atualize o modelo de página Editar Instrutores com o seguinte código:

```
public class EditModel : PageModel
{
    private readonly ContosoUniversity.Data.SchoolContext _context;

    public EditModel(ContosoUniversity.Data.SchoolContext context)
    {
        _context = context;
    }

    [BindProperty]
    public Instructor Instructor { get; set; }

    public async Task<IActionResult> OnGetAsync(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        Instructor = await _context.Instructors
            .Include(i => i.OfficeAssignment)
            .AsNoTracking()
            .FirstOrDefaultAsync(m => m.ID == id);

        if (Instructor == null)
        {
            return NotFound();
        }
        return Page();
    }

    public async Task<IActionResult> OnPostAsync(int? id)
    {
        if (!ModelState.IsValid)
        {
            return Page();
        }

        var instructorToUpdate = await _context.Instructors
            .Include(i => i.OfficeAssignment)
            .FirstOrDefaultAsync(s => s.ID == id);

        if (await TryUpdateModelAsync<Instructor>(
            instructorToUpdate,
            "Instructor",
            i => i.FirstMidName, i => i.LastName,
            i => i.HireDate, i => i.OfficeAssignment))
        {
            if (String.IsNullOrWhiteSpace(
                instructorToUpdate.OfficeAssignment?.Location))
            {
                instructorToUpdate.OfficeAssignment = null;
            }
            await _context.SaveChangesAsync();
        }
        return RedirectToPage("./Index");
    }
}
```

O código anterior:

- Obtém a entidade `Instructor` atual do banco de dados usando o carregamento adiantado para a propriedade de navegação `OfficeAssignment`.
- Atualiza a entidade `Instructor` recuperada com valores do associador de modelos. `TryUpdateModel` impede o [excesso de postagem](#).
- Se o local do escritório estiver em branco, `Instructor.OfficeAssignment` será definido como nulo. Quando `Instructor.OfficeAssignment` é nulo, a linha relacionada na tabela `OfficeAssignment` é excluída.

## Atualizar a página Editar Instrutor

Atualize `Pages/Instructors/Edit.cshtml` com o local do escritório:

```

@page
@model ContosoUniversity.Pages.Instructors.EditModel
 @{
    ViewData["Title"] = "Edit";
}
<h2>Edit</h2>
<h4>Instructor</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <input type="hidden" asp-for="Instructor.ID" />
            <div class="form-group">
                <label asp-for="Instructor.LastName" class="control-label"></label>
                <input asp-for="Instructor.LastName" class="form-control" />
                <span asp-validation-for="Instructor.LastName" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Instructor.FirstMidName" class="control-label"></label>
                <input asp-for="Instructor.FirstMidName" class="form-control" />
                <span asp-validation-for="Instructor.FirstMidName" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Instructor.HireDate" class="control-label"></label>
                <input asp-for="Instructor.HireDate" class="form-control" />
                <span asp-validation-for="Instructor.HireDate" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Instructor.OfficeAssignment.Location" class="control-label"></label>
                <input asp-for="Instructor.OfficeAssignment.Location" class="form-control" />
                <span asp-validation-for="Instructor.OfficeAssignment.Location" class="text-danger" />
            </div>
            <div class="form-group">
                <input type="submit" value="Save" class="btn btn-default" />
            </div>
        </form>
    </div>
</div>

<div>
    <a asp-page=".~/Index">Back to List</a>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}

```

Verifique se você pode alterar o local do escritório de um instrutor.

## Adicionar atribuições de Curso à página Editar Instrutor

Os instrutores podem ministrar a quantidade de cursos que desejarem. Nesta seção, você adiciona a capacidade

de alterar as atribuições de curso. A seguinte imagem mostra a página Editar Instrutor atualizada:

Last Name  
Abercrombie

First Name  
Kim

Hire Date  
3/11/1995

Office Location  
44/3P

1000 Algebra 2    1045 Calculus    1050 Chemistry  
 2021 Composition    2042 Literature    3141 Trigonometry  
 4022 Microeconomics    4041 Macroeconomics

Save

`Course` e `Instructor` têm uma relação muitos para muitos. Para adicionar e remover relações, adicione e remova entidades do conjunto de entidades de junção `CourseAssignments`.

As caixas de seleção permitem alterações em cursos aos quais um instrutor é atribuído. Uma caixa de seleção é exibida para cada curso no banco de dados. Os cursos aos quais o instrutor é atribuído estão marcados. O usuário pode marcar ou desmarcar as caixas de seleção para alterar as atribuições de curso. Se a quantidade de cursos for muito maior:

- Provavelmente, você usará outra interface do usuário para exibir os cursos.
- O método de manipulação de uma entidade de junção para criar ou excluir relações não será alterado.

#### Adicionar classes para dar suporte às páginas Criar e Editar Instrutor

Crie `SchoolViewModels/AssignedCourseData.cs` com o seguinte código:

```
namespace ContosoUniversity.Models.SchoolViewModels
{
    public class AssignedCourseData
    {
        public int CourseID { get; set; }
        public string Title { get; set; }
        public bool Assigned { get; set; }
    }
}
```

A classe `AssignedCourseData` contém dados para criar as caixas de seleção para os cursos atribuídos por um instrutor.

Crie a classe base `Pages/Instructors/InstructorCoursesPageModel.cshtml.cs`:

```
using ContosoUniversity.Data;
using ContosoUniversity.Models;
using ContosoUniversity.Models.SchoolViewModels;
using Microsoft.AspNetCore.Mvc.RazorPages;
using System.Collections.Generic;
using System.Linq;

namespace ContosoUniversity.Pages.Instructors
{
    public class InstructorCoursesPageModel : PageModel
    {

        public List<AssignedCourseData> AssignedCourseDataList;

        public void PopulateAssignedCourseData(SchoolContext context,
                                              Instructor instructor)
        {
            var allCourses = context.Courses;
            var instructorCourses = new HashSet<int>(
                instructor.CourseAssignments.Select(c => c.CourseID));
            AssignedCourseDataList = new List<AssignedCourseData>();
            foreach (var course in allCourses)
            {
                AssignedCourseDataList.Add(new AssignedCourseData
                {
                    CourseID = course.CourseID,
                    Title = course.Title,
                    Assigned = instructorCourses.Contains(course.CourseID)
                });
            }
        }

        public void UpdateInstructorCourses(SchoolContext context,
                                           string[] selectedCourses, Instructor instructorToUpdate)
        {
            if (selectedCourses == null)
            {
                instructorToUpdate.CourseAssignments = new List<CourseAssignment>();
                return;
            }

            var selectedCoursesHS = new HashSet<string>(selectedCourses);
            var instructorCourses = new HashSet<int>
                (instructorToUpdate.CourseAssignments.Select(c => c.Course.CourseID));
            foreach (var course in context.Courses)
            {
                if (selectedCoursesHS.Contains(course.CourseID.ToString()))
                {
                    if (!instructorCourses.Contains(course.CourseID))
                    {
                        instructorToUpdate.CourseAssignments.Add(
                            new CourseAssignment
                            {
                                InstructorID = instructorToUpdate.ID,
                                CourseID = course.CourseID
                            });
                    }
                }
                else
                {
                    if (instructorCourses.Contains(course.CourseID))
                    {

```

```
        CourseAssignment courseToRemove
        = instructorToUpdate
          .CourseAssignments
          .SingleOrDefault(i => i.CourseID == course.CourseID);
      context.Remove(courseToRemove);
    }
  }
}
```

A `InstructorCoursesPageModel` é a classe base que será usada para os modelos de página Editar e Criar. `PopulateAssignedCourseData` lê todas as entidades `Course` para popular `AssignedCourseDataList`. Para cada curso, o código define a `CourseID`, o título e se o instrutor está ou não atribuído ao curso. Um `HashSet` é usado para criar pesquisas eficientes.

### Modelo de página Editar Instrutor

Atualize o modelo de página Editar Instrutor com o seguinte código:

```

public class EditModel : InstructorCoursesPageModel
{
    private readonly ContosoUniversity.Data.SchoolContext _context;

    public EditModel(ContosoUniversity.Data.SchoolContext context)
    {
        _context = context;
    }

    [BindProperty]
    public Instructor Instructor { get; set; }

    public async Task<IActionResult> OnGetAsync(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        Instructor = await _context.Instructors
            .Include(i => i.OfficeAssignment)
            .Include(i => i.CourseAssignments).ThenInclude(i => i.Course)
            .AsNoTracking()
            .FirstOrDefaultAsync(m => m.ID == id);

        if (Instructor == null)
        {
            return NotFound();
        }
        PopulateAssignedCourseData(_context, Instructor);
        return Page();
    }

    public async Task<IActionResult> OnPostAsync(int? id, string[] selectedCourses)
    {
        if (!ModelState.IsValid)
        {
            return Page();
        }

        var instructorToUpdate = await _context.Instructors
            .Include(i => i.OfficeAssignment)
            .Include(i => i.CourseAssignments)
            .ThenInclude(i => i.Course)
            .FirstOrDefaultAsync(s => s.ID == id);

        if (await TryUpdateModelAsync<Instructor>(
            instructorToUpdate,
            "Instructor",
            i => i.FirstMidName, i => i.LastName,
            i => i.HireDate, i => i.OfficeAssignment))
        {
            if (String.IsNullOrWhiteSpace(
                instructorToUpdate.OfficeAssignment?.Location))
            {
                instructorToUpdate.OfficeAssignment = null;
            }
            UpdateInstructorCourses(_context, selectedCourses, instructorToUpdate);
            await _context.SaveChangesAsync();
            return RedirectToPage("./Index");
        }
        UpdateInstructorCourses(_context, selectedCourses, instructorToUpdate);
        PopulateAssignedCourseData(_context, instructorToUpdate);
        return Page();
    }
}

```

O código anterior manipula as alterações de atribuição de escritório.

Atualize a Exibição do Razor do instrutor:

```
@page
@model ContosoUniversity.Pages.Instructors.EditModel
 @{
    ViewData["Title"] = "Edit";
}
<h2>Edit</h2>
<h4>Instructor</h4>
<hr />


<div class="col-md-4">
        <form method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <input type="hidden" asp-for="Instructor.ID" />
            <div class="form-group">
                <label asp-for="Instructor.LastName" class="control-label"></label>
                <input asp-for="Instructor.LastName" class="form-control" />
                <span asp-validation-for="Instructor.LastName" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Instructor.FirstMidName" class="control-label"></label>
                <input asp-for="Instructor.FirstMidName" class="form-control" />
                <span asp-validation-for="Instructor.FirstMidName" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Instructor.HireDate" class="control-label"></label>
                <input asp-for="Instructor.HireDate" class="form-control" />
                <span asp-validation-for="Instructor.HireDate" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Instructor.OfficeAssignment.Location" class="control-label"></label>
                <input asp-for="Instructor.OfficeAssignment.Location" class="form-control" />
                <span asp-validation-for="Instructor.OfficeAssignment.Location" class="text-danger"></span>
            </div>
            <div class="form-group">
                <div class="col-md-offset-2 col-md-10">
                    <table>
                        <tr>
                            @{
                                int cnt = 0;

                                foreach (var course in Model.AssignedCourseDataList)
                                {
                                    if (cnt++ % 3 == 0)
                                    {
                                        @:</tr><tr>
                                    }
                                    @:<td>
                                        <input type="checkbox"
                                                name="selectedCourses"
                                                value="@course.CourseID"
                                                @(Html.Raw(course.Assigned ? "checked=\"checked\" : "")) />
                                            @course.CourseID @: @course.Title
                                    @:</td>
                                }
                                @:</tr>
                            }
                        </table>
                </div>
            <div class="form-group">
                <input type="submit" value="Save" class="btn btn-default" />
            </div>
        </form>
    </div>


```

```

</div>

<div>
    <a asp-page=".~/Index">Back to List</a>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}

```

#### NOTE

Quando você cola o código no Visual Studio, as quebras de linha são alteradas de uma forma que divide o código. Pressione Ctrl+Z uma vez para desfazer a formatação automática. A tecla de atalho Ctrl+Z corrige as quebras de linha para que elas se pareçam com o que você vê aqui. O recuo não precisa ser perfeito, mas cada uma das linhas `@</tr><tr>`, `@:<td>`, `@:</td>` e `@:</tr>` precisa estar em uma única linha, conforme mostrado. Com o bloco de novo código selecionado, pressione Tab três vezes para alinhar o novo código com o código existente. Vote ou examine o status deste bug [com este link](#).

Esse código anterior cria uma tabela HTML que contém três colunas. Cada coluna tem uma caixa de seleção e uma legenda que contém o número e o título do curso. Todas as caixas de seleção têm o mesmo nome ("selectedCourses"). O uso do mesmo nome instrui o associador de modelos a tratá-las como um grupo. O atributo de valor de cada caixa de seleção é definido como `CourseID`. Quando a página é postada, o associador de modelos passa uma matriz que consiste nos valores `CourseID` para apenas as caixas de seleção marcadas.

Quando as caixas de seleção são inicialmente renderizadas, os cursos atribuídos ao instrutor têm atributos marcados.

Execute o aplicativo e teste a página Editar Instrutor atualizada. Altere algumas atribuições de curso. As alterações são refletidas na página Índice.

Observação: a abordagem usada aqui para editar os dados de curso do instrutor funciona bem quando há uma quantidade limitada de cursos. Para coleções muito maiores, uma interface do usuário e um método de atualização diferentes são mais utilizáveis e eficientes.

#### Atualizar a página Criar Instrutor

Atualize o modelo de página Criar instrutor com o seguinte código:

```

using ContosoUniversity.Models;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace ContosoUniversity.Pages.Instructors
{
    public class CreateModel : InstructorCoursesPageModel
    {
        private readonly ContosoUniversity.Data.SchoolContext _context;

        public CreateModel(ContosoUniversity.Data.SchoolContext context)
        {
            _context = context;
        }

        public IActionResult OnGet()
        {
            var instructor = new Instructor();
            instructor.CourseAssignments = new List<CourseAssignment>();

            // Provides an empty collection for the foreach loop

```

```

        // foreach (var course in Model.AssignedCourseDataList)
        // in the Create Razor page.
        PopulateAssignedCourseData(_context, instructor);
        return Page();
    }

    [BindProperty]
    public Instructor Instructor { get; set; }

    public async Task<IActionResult> OnPostAsync(string[] selectedCourses)
    {
        if (!ModelState.IsValid)
        {
            return Page();
        }

        var newInstructor = new Instructor();
        if (selectedCourses != null)
        {
            newInstructor.CourseAssignments = new List<CourseAssignment>();
            foreach (var course in selectedCourses)
            {
                var courseToAdd = new CourseAssignment
                {
                    CourseID = int.Parse(course)
                };
                newInstructor.CourseAssignments.Add(courseToAdd);
            }
        }

        if (await TryUpdateModelAsync<Instructor>(
            newInstructor,
            "Instructor",
            i => i.FirstMidName, i => i.LastName,
            i => i.HireDate, i => i.OfficeAssignment))
        {
            _context.Instructors.Add(newInstructor);
            await _context.SaveChangesAsync();
            return RedirectToPage("./Index");
        }
        PopulateAssignedCourseData(_context, newInstructor);
        return Page();
    }
}

```

O código anterior é semelhante ao código de *Pages/Instructors/Edit.cshtml.cs*.

Atualize a página Criar instrutor do Razor com a seguinte marcação:

```

@page
@model ContosoUniversity.Pages.Instructors.CreateModel

 @{
    ViewData["Title"] = "Create";
}

<h2>Create</h2>

<h4>Instructor</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Instructor.LastName" class="control-label"></label>

```

```

        <input asp-for="Instructor.LastName" class="form-control" />
        <span asp-validation-for="Instructor.LastName" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Instructor.FirstMidName" class="control-label"></label>
        <input asp-for="Instructor.FirstMidName" class="form-control" />
        <span asp-validation-for="Instructor.FirstMidName" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Instructor.HireDate" class="control-label"></label>
        <input asp-for="Instructor.HireDate" class="form-control" />
        <span asp-validation-for="Instructor.HireDate" class="text-danger"></span>
    </div>

    <div class="form-group">
        <label asp-for="Instructor.OfficeAssignment.Location" class="control-label"></label>
        <input asp-for="Instructor.OfficeAssignment.Location" class="form-control" />
        <span asp-validation-for="Instructor.OfficeAssignment.Location" class="text-danger" />
    </div>
    <div class="form-group">
        <div class="col-md-offset-2 col-md-10">
            <table>
                <tr>
                    @{
                        int cnt = 0;

                        foreach (var course in Model.AssignedCourseDataList)
                        {
                            if (cnt++ % 3 == 0)
                            {
                                @:</tr><tr>
                            }
                            @:<td>
                                <input type="checkbox"
                                       name="selectedCourses"
                                       value="@course.CourseID"
                                       @(Html.Raw(course.Assigned ? "checked=\"checked\"" : ""))
                                       @course.CourseID @: @course.Title
                                @:</td>
                            }
                            @:</tr>
                        }
                    </table>
                </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-default" />
            </div>
        </form>
    </div>
</div>

<div>
    <a asp-page="Index">Back to List</a>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}

```

Teste a página Criar Instrutor.

## Atualizar a página Excluir

Atualize o modelo da página Excluir com o seguinte código:

```

using ContosoUniversity.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using System.Threading.Tasks;

namespace ContosoUniversity.Pages.Instructors
{
    public class DeleteModel : PageModel
    {
        private readonly ContosoUniversity.Data.SchoolContext _context;

        public DeleteModel(ContosoUniversity.Data.SchoolContext context)
        {
            _context = context;
        }

        [BindProperty]
        public Instructor Instructor { get; set; }

        public async Task<IActionResult> OnGetAsync(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            Instructor = await _context.Instructors.SingleAsync(m => m.ID == id);

            if (Instructor == null)
            {
                return NotFound();
            }
            return Page();
        }

        public async Task<IActionResult> OnPostAsync(int id)
        {
            Instructor instructor = await _context.Instructors
                .Include(i => i.CourseAssignments)
                .SingleAsync(i => i.ID == id);

            var departments = await _context.Departments
                .Where(d => d.InstructorID == id)
                .ToListAsync();
            departments.ForEach(d => d.InstructorID = null);

            _context.Instructors.Remove(instructor);

            await _context.SaveChangesAsync();
            return RedirectToPage("./Index");
        }
    }
}

```

O código anterior faz as seguintes alterações:

- Usa o carregamento adiantado para a propriedade de navegação `CourseAssignments`. `CourseAssignments` deve ser incluído ou eles não são excluídos quando o instrutor é excluído. Para evitar a necessidade de lê-las, configure a exclusão em cascata no banco de dados.
- Se o instrutor a ser excluído é atribuído como administrador de qualquer departamento, remove a atribuição de instrutor desse departamento.

[ANTERIOR](#)

[PRÓXIMO](#)

# Páginas Razor com o EF Core no ASP.NET Core – Simultaneidade – 8 de 8

10/01/2019 • 26 minutes to read • [Edit Online](#)

Por [Rick Anderson](#), [Tom Dykstra](#) e [Jon P Smith](#)

O aplicativo Web Contoso University demonstra como criar aplicativos Web das Páginas do Razor usando o EF Core e o Visual Studio. Para obter informações sobre a série de tutoriais, consulte [o primeiro tutorial](#).

Este tutorial mostra como lidar com conflitos quando os mesmos usuários atualizam uma entidade simultaneamente. Caso tenha problemas que não consiga resolver, [baixe ou exiba o aplicativo concluído](#). [Instruções de download](#).

## Conflitos de simultaneidade

Um conflito de simultaneidade ocorre quando:

- Um usuário navega para a página de edição de uma entidade.
- Outro usuário atualiza a mesma entidade antes que a primeira alteração do usuário seja gravada no BD.

Se a detecção de simultaneidade não estiver habilitada, quando ocorrerem atualizações simultâneas:

- A última atualização vencerá. Ou seja, os últimos valores de atualização serão salvos no BD.
- A primeira das atualizações atuais será perdida.

### Simultaneidade otimista

A simultaneidade otimista permite que conflitos de simultaneidade ocorram e, em seguida, responde adequadamente quando ocorrem. Por exemplo, Alice visita a página Editar Departamento e altera o orçamento para o departamento de inglês de US\$ 350.000,00 para US\$ 0,00.

The screenshot shows a web browser window with the title "Edit - Contoso University". The address bar displays "localhost:581". The main content area is titled "Edit" and "Department". It contains several input fields: "Budget" (value: 0), "Administrator" (dropdown menu showing "Abercrombie, Kim"), "Name" (text input: "English"), and "Start Date" (text input: "9/1/2007"). A "Save" button is at the bottom. A red box highlights the "Budget" input field.

Budget

0

Administrator

Abercrombie, Kim

Name

English

Start Date

9/1/2007

Save

Antes que Alice clique em **Salvar**, Julio visita a mesma página e altera o campo Data de Início de 1/9/2007 para 1/9/2013.

Budget  
350000.00

Administrator  
Abercrombie, Kim

Name  
English

**Start Date**  
9/1/2013

Save

Alice clica em **Salvar** primeiro e vê a alteração quando o navegador exibe a página Índice.

Name	Budget	Administrator	Start Date	
English	\$0.00	Abercrombie, Kim	2007-09-01	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Mathematics	\$100000.00	Eekhouri, Radi	2008-09-01	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

Julio clica em **Salvar** em uma página Editar que ainda mostra um orçamento de US\$ 350.000,00. O que acontece em seguida é determinado pela forma como você lida com conflitos de simultaneidade.

A simultaneidade otimista inclui as seguintes opções:

- Controle qual propriedade um usuário modificou e atualize apenas as colunas correspondentes no BD.

No cenário, não haverá perda de dados. Propriedades diferentes foram atualizadas pelos dois usuários. Na próxima vez que alguém navegar no departamento de inglês, verá as alterações de Alice e Julio. Esse método de atualização pode reduzir o número de conflitos que podem resultar em perda de dados. Essa abordagem:

- Não poderá evitar a perda de dados se forem feitas alterações concorrentes na mesma propriedade.

- Geralmente, não é prática em um aplicativo Web. Ela exige um estado de manutenção significativo para controlar todos os valores buscados e novos valores. Manter grandes quantidades de estado pode afetar o desempenho do aplicativo.
- Pode aumentar a complexidade do aplicativo comparado à detecção de simultaneidade em uma entidade.
- Você não pode deixar a alteração de Julio substituir a alteração de Alice.

Na próxima vez que alguém navegar pelo departamento de inglês, verá 1/9/2013 e o valor de US\$ 350.000,00 buscado. Essa abordagem é chamada de um cenário *O cliente vence* ou *O último vence*. (Todos os valores do cliente têm precedência sobre o conteúdo do armazenamento de dados.) Se você não fizer nenhuma codificação para a manipulação de simultaneidade, o cenário *O cliente vence* ocorrerá automaticamente.

- Você pode impedir que as alterações de Julio sejam atualizadas no BD. Normalmente, o aplicativo:

- Exibe uma mensagem de erro.
- Mostra o estado atual dos dados.
- Permite ao usuário aplicar as alterações novamente.

Isso é chamado de um cenário *O armazenamento vence*. (Os valores do armazenamento de dados têm precedência sobre os valores enviados pelo cliente.) Você implementará o cenário *O armazenamento vence* neste tutorial. Esse método garante que nenhuma alteração é substituída sem que um usuário seja alertado.

## Tratamento de simultaneidade

Quando uma propriedade é configurada como um [token de simultaneidade](#):

- O EF Core verifica se a propriedade não foi modificada depois que foi buscada. A verificação ocorre quando `SaveChanges` ou `SaveChangesAsync` é chamado.
- Se a propriedade tiver sido alterada depois que ela foi buscada, uma `DbUpdateConcurrencyException` será gerada.

O BD e o modelo de dados precisam ser configurados para dar suporte à geração de `DbUpdateConcurrencyException`.

### Detectando conflitos de simultaneidade em uma propriedade

Os conflitos de simultaneidade podem ser detectados no nível do propriedade com o atributo [ConcurrencyCheck](#). O atributo pode ser aplicado a várias propriedades no modelo. Para obter mais informações, consulte [Data Annotations-ConcurrencyCheck](#).

O atributo `[ConcurrencyCheck]` não é usado neste tutorial.

### Detectando conflitos de simultaneidade em uma linha

Para detectar conflitos de simultaneidade, uma coluna de controle de [rowversion](#) é adicionada ao modelo.

`rowversion` :

- É específico ao SQL Server. Outros bancos de dados podem não fornecer um recurso semelhante.
- É usado para determinar se uma entidade não foi alterada desde que foi buscada no BD.

O BD gera um número `rowversion` sequencial que é incrementado sempre que a linha é atualizada. Em um comando `Update` ou `Delete`, a cláusula `Where` inclui o valor buscado de `rowversion`. Se a linha que está sendo atualizada foi alterada:

- `rowversion` não corresponde ao valor buscado.
- Os comandos `Update` ou `Delete` não encontram uma linha porque a cláusula `Where` inclui a `rowversion`

buscada.

- Uma `DbUpdateConcurrencyException` é gerada.

No EF Core, quando nenhuma linha é atualizada por um comando `Update` ou `Delete`, uma exceção de simultaneidade é gerada.

### Adicionar uma propriedade de controle à entidade Department

Em `Models/Department.cs`, adicione uma propriedade de controle chamada `RowVersion`:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Department
    {
        public int DepartmentID { get; set; }

        [StringLength(50, MinimumLength = 3)]
        public string Name { get; set; }

        [DataType(DataType.Currency)]
        [Column(TypeName = "money")]
        public decimal Budget { get; set; }

        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        [Display(Name = "Start Date")]
        public DateTime StartDate { get; set; }

        public int? InstructorID { get; set; }

        [Timestamp]
        public byte[] RowVersion { get; set; }

        public Instructor Administrator { get; set; }
        public ICollection<Course> Courses { get; set; }
    }
}
```

O atributo `Timestamp` especifica que essa coluna é incluída na cláusula `Where` dos comandos `Update` e `Delete`.

O atributo é chamado `Timestamp` porque as versões anteriores do SQL Server usavam um tipo de dados

`timestamp` do SQL antes de o tipo `rowversion` SQL substituí-lo.

A API fluente também pode especificar a propriedade de controle:

```
modelBuilder.Entity<Department>()
    .Property<byte[]>("RowVersion")
    .IsRowVersion();
```

O seguinte código mostra uma parte do T-SQL gerado pelo EF Core quando o nome do Departamento é atualizado:

```
SET NOCOUNT ON;
UPDATE [Department] SET [Name] = @p0
WHERE [DepartmentID] = @p1 AND [RowVersion] = @p2;
SELECT [RowVersion]
FROM [Department]
WHERE @@ROWCOUNT = 1 AND [DepartmentID] = @p1;
```

O código anterior realçado mostra a cláusula `WHERE` que contém `RowVersion`. Se o BD `RowVersion` não for igual ao parâmetro `RowVersion` (`@p2`), nenhuma linha será atualizada.

O seguinte código realçado mostra o T-SQL que verifica exatamente se uma linha foi atualizada:

```
SET NOCOUNT ON;
UPDATE [Department] SET [Name] = @p0
WHERE [DepartmentID] = @p1 AND [RowVersion] = @p2;
SELECT [RowVersion]
FROM [Department]
WHERE @@ROWCOUNT = 1 AND [DepartmentID] = @p1;
```

`@@ROWCOUNT` retorna o número de linhas afetadas pela última instrução. Quando nenhuma linha é atualizada, o EF Core gera uma `DbUpdateConcurrencyException`.

Veja o T-SQL gerado pelo EF Core na janela de Saída do Visual Studio.

## Atualizar o BD

A adição da propriedade `RowVersion` altera o modelo de BD, o que exige uma migração.

Compile o projeto. Insira o seguinte em uma janela Comando:

```
dotnet ef migrations add RowVersion
dotnet ef database update
```

Os comandos anteriores:

- Adicionam o arquivo de migração `Migrations/{time stamp}_RowVersion.cs`.
- Atualizam o arquivo `Migrations/SchoolContextModelSnapshot.cs`. A atualização adiciona o seguinte código realçado ao método `BuildModel`:

```

modelBuilder.Entity("ContosoUniversity.Models.Department", b =>
{
    b.Property<int>"("DepartmentID")
        .ValueGeneratedOnAdd();

    b.Property<decimal>"("Budget")
        .HasColumnType("money");

    b.Property<int?>"("InstructorID");

    b.Property<string>"("Name")
        .HasMaxLength(50);

    b.Property<byte[]>"("RowVersion")
        .IsConcurrencyToken()
        .ValueGeneratedOnAddOrUpdate();

    b.Property<DateTime>"("StartDate");

    b.HasKey("DepartmentID");

    b.HasIndex("InstructorID");

    b.ToTable("Department");
});

```

- Executam migrações para atualizar o BD.

## Gerar o modelo Departamentos por scaffolding

- [Visual Studio](#)
- [CLI do .NET Core](#)

Siga as instruções em [Gere um modelo de aluno por scaffold](#) e use `Department` para a classe de modelo.

O comando anterior gera o modelo `Department` por scaffolding. Abra o projeto no Visual Studio.

Compile o projeto.

### Atualizar a página Índice de Departamentos

O mecanismo de scaffolding criou uma coluna `RowVersion` para a página Índice, mas esse campo não deve ser exibido. Neste tutorial, o último byte da `RowVersion` é exibido para ajudar a entender a simultaneidade. O último byte não tem garantia de ser exclusivo. Um aplicativo real não exibe `RowVersion` ou o último byte de `RowVersion`.

Atualize a página Índice:

- Substitua Índice por Departamentos.
- Substitua a marcação que contém `RowVersion` pelo último byte de `RowVersion`.
- Substitua `FirstMidName` por `FullName`.

A seguinte marcação mostra a página atualizada:

```

@page
@model ContosoUniversity.Pages.Departments.IndexModel

 @{
    ViewData["Title"] = "Departments";
}

<h2>Departments</h2>

<p>
    <a asp-page="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Department[0].Name)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Department[0].Budget)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Department[0].StartDate)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Department[0].Administrator)
            </th>
            <th>
                RowVersion
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model.Department) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Name)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Budget)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.StartDate)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Administrator.FullName)
            </td>
            <td>
                @item.RowVersion[7]
            </td>
            <td>
                <a asp-page=".Edit" asp-route-id="@item.DepartmentID">Edit</a> |
                <a asp-page=".Details" asp-route-id="@item.DepartmentID">Details</a> |
                <a asp-page=".Delete" asp-route-id="@item.DepartmentID">Delete</a>
            </td>
        </tr>
}
    </tbody>
</table>

```

[Atualizar o modelo da página](#) [Editar](#)

Atualize `pages\departments\edit.cshtml.cs` com o seguinte código:

```
using ContosoUniversity.Data;
```

```
using ContosoUniversity.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using System.Threading.Tasks;

namespace ContosoUniversity.Pages.Departments
{
    public class EditModel : PageModel
    {
        private readonly ContosoUniversity.Data.SchoolContext _context;

        public EditModel(ContosoUniversity.Data.SchoolContext context)
        {
            _context = context;
        }

        [BindProperty]
        public Department Department { get; set; }
        // Replace ViewData["InstructorID"]
        public SelectList InstructorNameSL { get; set; }

        public async Task<IActionResult> OnGetAsync(int id)
        {
            Department = await _context.Departments
                .Include(d => d.Administrator) // eager loading
                .AsNoTracking() // tracking not required
                .FirstOrDefaultAsync(m => m.DepartmentID == id);

            if (Department == null)
            {
                return NotFound();
            }

            // Use strongly typed data rather than ViewData.
            InstructorNameSL = new SelectList(_context.Instructors,
                "ID", "FirstMidName");

            return Page();
        }

        public async Task<IActionResult> OnPostAsync(int id)
        {
            if (!ModelState.IsValid)
            {
                return Page();
            }

            var departmentToUpdate = await _context.Departments
                .Include(i => i.Administrator)
                .FirstOrDefaultAsync(m => m.DepartmentID == id);

            // null means Department was deleted by another user.
            if (departmentToUpdate == null)
            {
                return await HandleDeletedDepartment();
            }

            // Update the RowVersion to the value when this entity was
            // fetched. If the entity has been updated after it was
            // fetched, RowVersion won't match the DB RowVersion and
            // a DbUpdateConcurrencyException is thrown.
            // A second postback will make them match, unless a new
            // concurrency issue happens.
            _context.Entry(departmentToUpdate)
                .Property("RowVersion").OriginalValue = Department.RowVersion;
        }
    }
}
```

```

        if (await TryUpdateModelAsync<Department>(
            departmentToUpdate,
            "Department",
            s => s.Name, s => s.StartDate, s => s.Budget, s => s.InstructorID))
    {
        try
        {
            await _context.SaveChangesAsync();
            return RedirectToPage("./Index");
        }
        catch (DbUpdateConcurrencyException ex)
        {
            var exceptionEntry = ex.Entries.Single();
            var clientValues = (Department)exceptionEntry.Entity;
            var databaseEntry = exceptionEntry.GetDatabaseValues();
            if (databaseEntry == null)
            {
                ModelState.AddModelError(string.Empty, "Unable to save. " +
                    "The department was deleted by another user.");
                return Page();
            }

            var dbValues = (Department)databaseEntry.ToObject();
            await setDbErrorMessage(dbValues, clientValues, _context);

            // Save the current RowVersion so next postback
            // matches unless an new concurrency issue happens.
            Department.RowVersion = (byte[])dbValues.RowVersion;
            // Must clear the model error for the next postback.
            ModelState.Remove("Department.RowVersion");
        }
    }

    InstructorNameSL = new SelectList(_context.Instructors,
        "ID", "FullName", departmentToUpdate.InstructorID);

    return Page();
}

private async Task<IActionResult> HandleDeletedDepartment()
{
    Department deletedDepartment = new Department();
    // ModelState contains the posted data because of the deletion error and will override the
    Department instance values when displaying Page().
    ModelState.AddModelError(string.Empty,
        "Unable to save. The department was deleted by another user.");
    InstructorNameSL = new SelectList(_context.Instructors, "ID", "FullName",
    Department.InstructorID);
    return Page();
}

private async Task setDbErrorMessage(Department dbValues,
    Department clientValues, SchoolContext context)
{

    if (dbValues.Name != clientValues.Name)
    {
        ModelState.AddModelError("Department.Name",
            $"Current value: {dbValues.Name}");
    }
    if (dbValues.Budget != clientValues.Budget)
    {
        ModelState.AddModelError("Department.Budget",
            $"Current value: {dbValues.Budget:c}");
    }
    if (dbValues.StartDate != clientValues.StartDate)
    {
        ModelState.AddModelError("Department.StartDate",
            $"Current value: {dbValues.StartDate:d}");
    }
}

```

```

        }

        if (dbValues.InstructorID != clientValues.InstructorID)
        {
            Instructor dbInstructor = await _context.Instructors
                .FindAsync(dbValues.InstructorID);
            ModelState.AddModelError("Department.InstructorID",
                $"Current value: {dbInstructor?.FullName}");
        }

        ModelState.AddModelError(string.Empty,
            "The record you attempted to edit "
            + "was modified by another user after you. The "
            + "edit operation was canceled and the current values in the database "
            + "have been displayed. If you still want to edit this record, click "
            + "the Save button again.");
    }
}
}
}

```

Para detectar um problema de simultaneidade, o `OriginalValue` é atualizado com o valor `rowVersion` da entidade que foi buscada. O EF Core gera um comando SQL UPDATE com uma cláusula WHERE que contém o valor `RowVersion` original. Se nenhuma linha for afetada pelo comando UPDATE (nenhuma linha tem o valor `RowVersion` original), uma exceção `DbUpdateConcurrencyException` será gerada.

```

public async Task<IActionResult> OnPostAsync(int id)
{
    if (!ModelState.IsValid)
    {
        return Page();
    }

    var departmentToUpdate = await _context.Departments
        .Include(i => i.Administrator)
        .FirstOrDefaultAsync(m => m.DepartmentID == id);

    // null means Department was deleted by another user.
    if (departmentToUpdate == null)
    {
        return await HandleDeletedDepartment();
    }

    // Update the RowVersion to the value when this entity was
    // fetched. If the entity has been updated after it was
    // fetched, RowVersion won't match the DB RowVersion and
    // a DbUpdateConcurrencyException is thrown.
    // A second postback will make them match, unless a new
    // concurrency issue happens.
    _context.Entry(departmentToUpdate)
        .Property("RowVersion").OriginalValue = Department.RowVersion;
}

```

No código anterior, `Department.RowVersion` é o valor quando a entidade foi buscada. `OriginalValue` é o valor no BD quando `FirstOrDefaultAsync` foi chamado nesse método.

O seguinte código obtém os valores de cliente (os valores postados nesse método) e os valores do BD:

```

try
{
    await _context.SaveChangesAsync();
    return RedirectToPage("./Index");
}
catch (DbUpdateConcurrencyException ex)
{
    var exceptionEntry = ex.Entries.Single();
    var clientValues = (Department)exceptionEntry.Entity;
    var databaseEntry = exceptionEntry.GetDatabaseValues();
    if (databaseEntry == null)
    {
        ModelState.AddModelError(string.Empty, "Unable to save. " +
            "The department was deleted by another user.");
        return Page();
    }

    var dbValues = (Department)databaseEntry.ToObject();
    await setDbErrorMessage(dbValues, clientValues, _context);

    // Save the current RowVersion so next postback
    // matches unless a new concurrency issue happens.
    Department.RowVersion = (byte[])dbValues.RowVersion;
    // Must clear the model error for the next postback.
    ModelState.Remove("Department.RowVersion");
}

```

O seguinte código adiciona uma mensagem de erro personalizada a cada coluna que tem valores de BD diferentes daqueles que foram postados em `onPostAsync`:

```

private async Task setDbErrorMessage(Department dbValues,
    Department clientValues, SchoolContext context)
{

    if (dbValues.Name != clientValues.Name)
    {
        ModelState.AddModelError("Department.Name",
            $"Current value: {dbValues.Name}");
    }
    if (dbValues.Budget != clientValues.Budget)
    {
        ModelState.AddModelError("Department.Budget",
            $"Current value: {dbValues.Budget:c}");
    }
    if (dbValues.StartDate != clientValues.StartDate)
    {
        ModelState.AddModelError("Department.StartDate",
            $"Current value: {dbValues.StartDate:d}");
    }
    if (dbValues.InstructorID != clientValues.InstructorID)
    {
        Instructor dbInstructor = await _context.Instructors
            .FindAsync(dbValues.InstructorID);
        ModelState.AddModelError("Department.InstructorID",
            $"Current value: {dbInstructor?.FullName}");
    }

    ModelState.AddModelError(string.Empty,
        "The record you attempted to edit "
        + "was modified by another user after you. The "
        + "edit operation was canceled and the current values in the database "
        + "have been displayed. If you still want to edit this record, click "
        + "the Save button again.");
}

```

O código realçado a seguir define o valor `RowVersion` com o novo valor recuperado do BD. Na próxima vez que o usuário clicar em **Salvar**, somente os erros de simultaneidade que ocorrerem desde a última exibição da página Editar serão capturados.

```
try
{
    await _context.SaveChangesAsync();
    return RedirectToPage("./Index");
}
catch (DbUpdateConcurrencyException ex)
{
    var exceptionEntry = ex.Entries.Single();
    var clientValues = (Department)exceptionEntry.Entity;
    var databaseEntry = exceptionEntry.GetDatabaseValues();
    if (databaseEntry == null)
    {
        ModelState.AddModelError(string.Empty, "Unable to save. " +
            "The department was deleted by another user.");
        return Page();
    }

    var dbValues = (Department)databaseEntry.ToObject();
    await setDbErrorMessage(dbValues, clientValues, _context);

    // Save the current RowVersion so next postback
    // matches unless a new concurrency issue happens.
    Department.RowVersion = (byte[])dbValues.RowVersion;
    // Must clear the model error for the next postback.
    ModelState.Remove("Department.RowVersion");
}
```

A instrução `ModelState.Remove` é obrigatória porque `ModelState` tem o valor `RowVersion` antigo. Na Página do Razor, o valor `ModelState` de um campo tem precedência sobre os valores de propriedade do modelo, quando ambos estão presentes.

## Atualizar a página Editar

Atualize `Pages/Departments/Edit.cshtml` com a seguinte marcação:

```

@page "{id:int}"
@model ContosoUniversity.Pages.Departments.EditModel
 @{
     ViewData["Title"] = "Edit";
 }
<h2>Edit</h2>
<h4>Department</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <input type="hidden" asp-for="Department.DepartmentID" />
            <input type="hidden" asp-for="Department.RowVersion" />
            <div class="form-group">
                <label>RowVersion</label>
                @Model.Department.RowVersion[7]
            </div>
            <div class="form-group">
                <label asp-for="Department.Name" class="control-label"></label>
                <input asp-for="Department.Name" class="form-control" />
                <span asp-validation-for="Department.Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Department.Budget" class="control-label"></label>
                <input asp-for="Department.Budget" class="form-control" />
                <span asp-validation-for="Department.Budget" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Department.StartDate" class="control-label"></label>
                <input asp-for="Department.StartDate" class="form-control" />
                <span asp-validation-for="Department.StartDate" class="text-danger">
                    </span>
            </div>
            <div class="form-group">
                <label class="control-label">Instructor</label>
                <select asp-for="Department.InstructorID" class="form-control" asp-items="@Model.InstructorNameSL"></select>
                <span asp-validation-for="Department.InstructorID" class="text-danger">
                    </span>
            </div>
            <div class="form-group">
                <input type="submit" value="Save" class="btn btn-default" />
            </div>
        </form>
    </div>
</div>
<div>
    <a asp-page=".~/Index">Back to List</a>
</div>
@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}

```

A marcação anterior:

- Atualiza a diretiva `page` de `@page` para `@page "{id:int}"`.
- Adiciona uma versão de linha oculta. `RowVersion` deve ser adicionado para que o postback associe o valor.
- Exibe o último byte de `RowVersion` para fins de depuração.
- Substitui `ViewData` pelo `InstructorNameSL` fortemente tipado.

## Testar conflitos de simultaneidade com a página Editar

Abra duas instâncias de navegadores de Editar no departamento de inglês:

- Execute o aplicativo e selecione Departamentos.
- Clique com o botão direito do mouse no hiperlink **Editar** do departamento de inglês e selecione **Abrir em uma nova guia**.
- Na primeira guia, clique no hiperlink **Editar** do departamento de inglês.

As duas guias do navegador exibem as mesmas informações.

Altere o nome na primeira guia do navegador e clique em **Salvar**.

The screenshot shows a browser window with two tabs. Both tabs have the same URL: <localhost:1234/Departments/Edit/1>.  
The left tab (foreground) displays the 'Edit Department' form. It includes fields for Name (containing 'Languages'), Budget (containing '350000.00'), Start Date ('09/01/2007'), Instructor ('Kim'), and a 'Save' button.  
The right tab (background) also displays the 'Edit Department' form, showing the updated value 'Languages' in the Name field, indicating a successful postback from the first tab.

O navegador mostra a página de Índice com o valor alterado e o indicador de rowVersion atualizado. Observe o indicador de rowVersion atualizado: ele é exibido no segundo postback na outra guia.

Altere outro campo na segunda guia do navegador.

Screenshot of a web browser showing the 'Edit - Contos' page for a department. The browser title bar says 'Edit - Contos'. The address bar shows 'localhost:1234/Departments/Edit/1'. The page header says 'Contoso University'. The main content area has a heading 'Edit Department'. Below it, there is a 'RowVersion 209' label. The form fields include:

- Name**: English
- Budget**: 5000000 (highlighted with a red box)
- Start Date**: 09/01/2007
- Instructor**: Kim

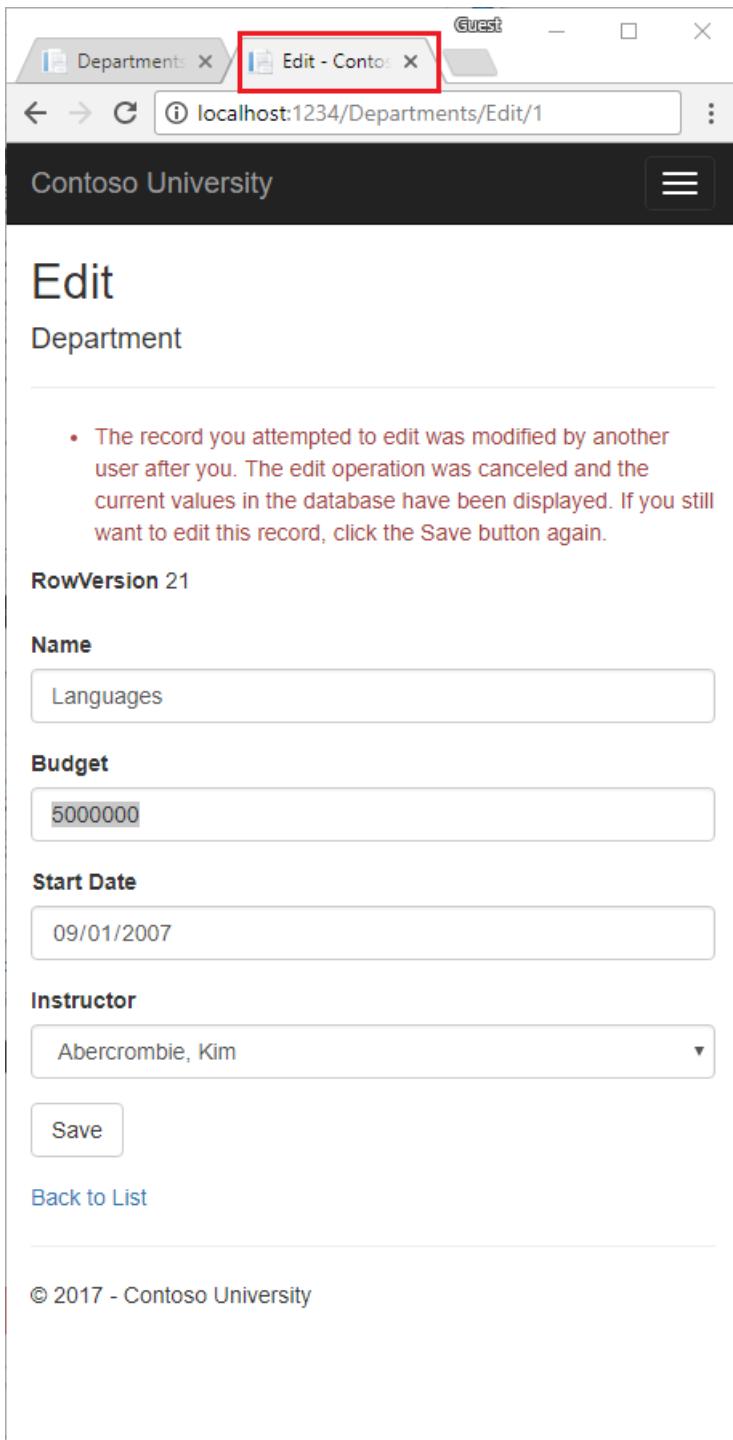
At the bottom left is a 'Save' button, and at the bottom center is a link 'Back to List'. At the very bottom is a copyright notice: '© 2017 - Contoso University'.

Clique em **Salvar**. Você verá mensagens de erro em todos os campos que não correspondem aos valores do BD:

Screenshot of a browser window showing the 'Edit - Contoso' page for a department. The URL is localhost:1234/Departments/Edit/1. The page title is 'Contoso University'. The main content area shows an error message: 'The record you attempted to edit was modified by another user after you. The edit operation was canceled and the current values in the database have been displayed. If you still want to edit this record, click the Save button again.' Below this, there are fields for 'Name' (set to 'English'), 'Budget' (set to '5000000'), 'Start Date' (set to '09/01/2007'), and 'Instructor' (set to 'Abercrombie, Kim'). A 'Save' button is visible at the bottom left.

The browser title bar shows 'Guest' and the address bar shows 'localhost:1234/Departments/Edit/1'. The page header says 'Contoso University'. The main content area has a heading 'Edit' and a sub-heading 'Department'. A note states: 'The record you attempted to edit was modified by another user after you. The edit operation was canceled and the current values in the database have been displayed. If you still want to edit this record, click the Save button again.' Below this, there are four form fields: 'Name' (value: English), 'Budget' (value: 5000000), 'Start Date' (value: 09/01/2007), and 'Instructor' (value: Abercrombie, Kim). At the bottom left is a 'Save' button, and at the bottom right is a link 'Back to List'. The footer contains the copyright notice '© 2017 - Contoso University'.

Essa janela do navegador não pretendia alterar o campo Name. Copie e cole o valor atual (Languages) para o campo Name. Saída da guia. A validação do lado do cliente remove a mensagem de erro.



Clique em **Salvar** novamente. O valor inserido na segunda guia do navegador foi salvo. Você verá os valores salvos na página Índice.

## Atualizar a página Excluir

Atualize o modelo da página Excluir com o seguinte código:

```
using ContosoUniversity.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using System.Threading.Tasks;

namespace ContosoUniversity.Pages.Departments
{
    public class DeleteModel : PageModel
    {
        private readonly ContosoUniversity.Data.SchoolContext _context;
```

```

public DeleteModel(ContosoUniversity.Data.SchoolContext context)
{
    _context = context;
}

[BindProperty]
public Department Department { get; set; }
public string ConcurrencyErrorMessage { get; set; }

public async Task<IActionResult> OnGetAsync(int id, bool? concurrencyError)
{
    Department = await _context.Departments
        .Include(d => d.Administrator)
        .AsNoTracking()
        .FirstOrDefaultAsync(m => m.DepartmentID == id);

    if (Department == null)
    {
        return NotFound();
    }

    if (concurrencyError.GetValueOrDefault())
    {
        ConcurrencyErrorMessage = "The record you attempted to delete "
            + "was modified by another user after you selected delete. "
            + "The delete operation was canceled and the current values in the "
            + "database have been displayed. If you still want to delete this "
            + "record, click the Delete button again.";
    }
    return Page();
}

public async Task<IActionResult> OnPostAsync(int id)
{
    try
    {
        if (await _context.Departments.AnyAsync(
            m => m.DepartmentID == id))
        {
            // Department.rowVersion value is from when the entity
            // was fetched. If it doesn't match the DB, a
            // DbUpdateConcurrencyException exception is thrown.
            _context.Departments.Remove(Department);
            await _context.SaveChangesAsync();
        }
        return RedirectToPage("./Index");
    }
    catch (DbUpdateConcurrencyException)
    {
        return RedirectToPage("./Delete",
            new { concurrencyError = true, id = id });
    }
}
}

```

A página Excluir detectou conflitos de simultaneidade quando a entidade foi alterada depois de ser buscada. `Department.RowVersion` é a versão de linha quando a entidade foi buscada. Quando o EF Core cria o comando SQL DELETE, ele inclui uma cláusula WHERE com `RowVersion`. Se o comando SQL DELETE não resultar em nenhuma linha afetada:

- A `RowVersion` no comando SQL DELETE não corresponderá a `RowVersion` no BD.
  - Uma exceção `DbUpdateConcurrencyException` é gerada.
  - `OnGetAsync` é chamado com o `concurrencyError`.

## Atualizar a página Excluir

Atualize *Pages/Departments/Delete.cshtml* com o seguinte código:

```
@page "{id:int}"
@model ContosoUniversity.Pages.Departments.DeleteModel

@{
    ViewData["Title"] = "Delete";
}



## Delete



@Model.ConcurrencyErrorMessage



### Are you sure you want to delete this?



#### Department



---



<dt>
            @Html.DisplayNameFor(model => model.Department.Name)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Department.Name)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Department.Budget)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Department.Budget)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Department.StartDate)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Department.StartDate)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Department.RowVersion)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Department.RowVersion[7])
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Department.Administrator)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Department.Administrator.FullName)
        </dd>


    <form method="post">
        <input type="hidden" asp-for="Department.DepartmentID" />
        <input type="hidden" asp-for="Department.RowVersion" />
        <div class="form-actions no-color">
            <input type="submit" value="Delete" class="btn btn-default" /> |
            <a asp-page=".~/Index">Back to List</a>
        </div>
    </form>


```

A marcação anterior faz as seguintes alterações:

- Atualiza a diretiva `page` de `@page` para `@page "{id:int}"`.
- Adiciona uma mensagem de erro.

- Substitua FirstMidName por FullName no campo **Administrador**.
- Altere `RowVersion` para exibir o último byte.
- Adiciona uma versão de linha oculta. `RowVersion` deve ser adicionado para que o postback associe o valor.

## Testar conflitos de simultaneidade com a página Excluir

Crie um departamento de teste.

Abra duas instâncias dos navegadores de Excluir no departamento de teste:

- Execute o aplicativo e selecione Departamentos.
- Clique com o botão direito do mouse no hiperlink **Excluir** do departamento de teste e selecione **Abrir em uma nova guia**.
- Clique no hiperlink **Editar** do departamento de teste.

As duas guias do navegador exibem as mesmas informações.

Altere o orçamento na primeira guia do navegador e clique em **Salvar**.

O navegador mostra a página de Índice com o valor alterado e o indicador de rowVersion atualizado. Observe o indicador de rowVersion atualizado: ele é exibido no segundo postback na outra guia.

Exclua o departamento de teste na segunda guia. Um erro de simultaneidade é exibido com os valores atuais do BD. Clicar em **Excluir** exclui a entidade, a menos que `RowVersion` tenha sido atualizada e o departamento tenha sido excluído.

Consulte [Herança](#) para saber como herdar um modelo de dados.

## Recursos adicionais

- [Tokens de simultaneidade no EF Core](#)
- [Lidar com a simultaneidade no EF Core](#)

[ANTERIOR](#)

# ASP.NET Core MVC com EF Core – série de tutoriais

21/06/2018 • 2 minutes to read • [Edit Online](#)

Este tutorial ensina a usar o ASP.NET Core MVC e o Entity Framework Core com controladores e exibições. As Páginas Razor é uma nova alternativa no ASP.NET Core 2.0, um modelo de programação baseado em página que torna a criação da interface do usuário da Web mais fácil e produtiva. É recomendável tentar o tutorial das [Páginas Razor](#) na versão do MVC. O tutorial Páginas do Razor:

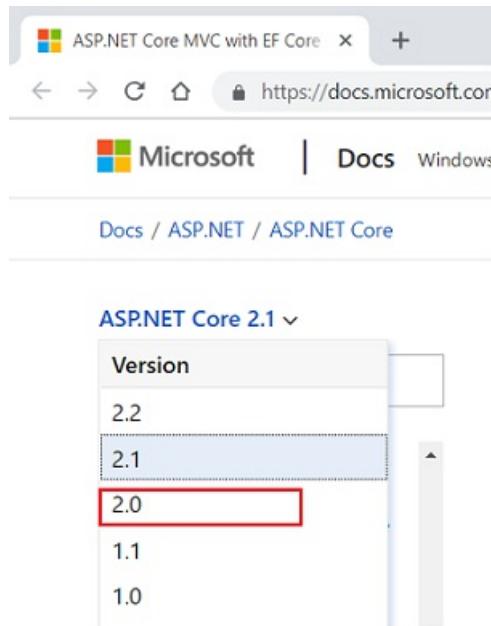
- É mais fácil de acompanhar.
- Fornece mais práticas recomendadas do EF Core.
- Usa consultas mais eficientes.
- É mais atual com a API mais recente.
- Aborda mais recursos.

1. [Introdução](#)
2. [Operações Create, Read, Update e Delete](#)
3. [Classificação, filtragem, paginação e agrupamento](#)
4. [Migrações](#)
5. [Criar um modelo de dados complexo](#)
6. [Lendo dados relacionados](#)
7. [Atualizando dados relacionados](#)
8. [Tratar conflitos de simultaneidade](#)
9. [Herança](#)
10. [Tópicos avançados](#)

# ASP.NET Core MVC com o Entity Framework Core – tutorial – 1 de 10

10/01/2019 • 40 minutes to read • [Edit Online](#)

Este tutorial não foi atualizado para o ASP.NET Core 2.1. A versão deste tutorial para o ASP.NET Core 2.0 pode ser consultada selecionando **ASP.NET Core 2.0** acima do sumário ou na parte superior da página:



The screenshot shows a browser window with the URL <https://docs.microsoft.com>. The page title is "ASP.NET Core MVC with EF Core". Below the title, there's a Microsoft logo and a "Docs" link. The main content area has a dropdown menu titled "ASP.NET Core 2.1" with the following options: Version, 2.2, 2.1, 2.0 (which is highlighted with a red border), 1.1, and 1.0.

A versão deste tutorial do [Razor Pages](#) para o ASP.NET Core 2.1 conta com várias melhorias em relação à versão 2.0.

O tutorial do 2.0 ensina a usar o ASP.NET Core MVC e o Entity Framework Core com controladores e exibições. O tutorial do Razor Pages foi atualizado com os seguintes aprimoramentos:

- É mais fácil de acompanhar. Por exemplo, o código de scaffolding foi bastante simplificado.
- Fornece mais práticas recomendadas do EF Core.
- Usa consultas mais eficientes.
- Usa a API do EF Core mais recente.

Por [Tom Dykstra](#) e [Rick Anderson](#)

Este tutorial ensina a usar o ASP.NET Core MVC e o Entity Framework Core com controladores e exibições. As Páginas Razor é uma nova alternativa no ASP.NET Core 2.0, um modelo de programação baseado em página que torna a criação da interface do usuário da Web mais fácil e produtiva. É recomendável tentar o tutorial das [Páginas Razor](#) na versão do MVC. O tutorial Páginas do Razor:

- É mais fácil de acompanhar.
- Fornece mais práticas recomendadas do EF Core.
- Usa consultas mais eficientes.
- É mais atual com a API mais recente.
- Aborda mais recursos.

O aplicativo Web de exemplo Contoso University demonstra como criar aplicativos Web ASP.NET Core 2.0 MVC usando o EF (Entity Framework) Core 2.0 e o Visual Studio 2017.

O aplicativo de exemplo é um site de uma Contoso University fictícia. Ele inclui funcionalidades como admissão de alunos, criação de cursos e atribuições de instrutor. Este é o primeiro de uma série de tutoriais que explica como criar o aplicativo de exemplo Contoso University do zero.

[Baixe ou exiba o aplicativo concluído.](#)

O EF Core 2.0 é a última versão do EF, mas ainda não tem todos os recursos do EF 6.x. Para obter informações sobre como escolher entre o EF 6.x e o EF Core, consulte [EF Core vs. EF6.x](#). Se você escolher o EF 6.x, confira [a versão anterior desta série de tutoriais](#).

#### NOTE

Para obter a versão ASP.NET Core 1.1 deste tutorial, confira a [versão VS 2017 Atualização 2](#) deste tutorial em formato PDF.

## Pré-requisitos

Clique em **uma** das seguintes opções:

- Ferramentas da CLI: Windows, Linux ou macOS: [SDK 2.0 ou posterior do .NET Core](#)
- Ferramentas de editor/IDE
  - Windows: [Visual Studio para Windows](#)
    - Carga de trabalho **ASP.NET e desenvolvimento para a Web**
    - Carga de trabalho de **desenvolvimento multiplataforma do .NET Core**
  - Linux: [Visual Studio Code](#)
  - macOS: [Visual Studio para Mac](#)

## Solução de problemas

Caso tenha um problema que não consiga resolver, em geral, você poderá encontrar a solução comparando o código com o [projeto concluído](#). Para obter uma lista de erros comuns e como resolvê-los, consulte [a seção Solução de problemas do último tutorial da série](#). Caso não encontre o que precisa na seção, poste uma pergunta no StackOverflow.com sobre o [ASP.NET Core](#) ou o [EF Core](#).

#### TIP

Esta é uma série de dez tutoriais, cada um se baseando no que é feito nos tutoriais anteriores. Considere a possibilidade de salvar uma cópia do projeto após a conclusão bem-sucedida de cada tutorial. Caso tenha problemas, comece novamente no tutorial anterior em vez de voltar ao início de toda a série.

## O aplicativo Web Contoso University

O aplicativo que você criará nestes tutoriais é um site simples de uma universidade.

Os usuários podem exibir e atualizar informações de alunos, cursos e instrutores. Estas são algumas das telas que você criará.

LastName	FirstMidName	EnrollmentDate	
Alexander	Carson	9/1/2005 12:00:00 AM	Edit   Details   Delete
Alonso	Meredith	9/1/2002 12:00:00 AM	Edit   Details   Delete
Anand	Arturo	9/1/2003 12:00:00 AM	Edit   Details   Delete
Barzdukas	Gytis	9/1/2002 12:00:00 AM	Edit   Details   Delete

**Edit**

Student

---

**EnrollmentDate**

**FirstMidName**

**LastNames**

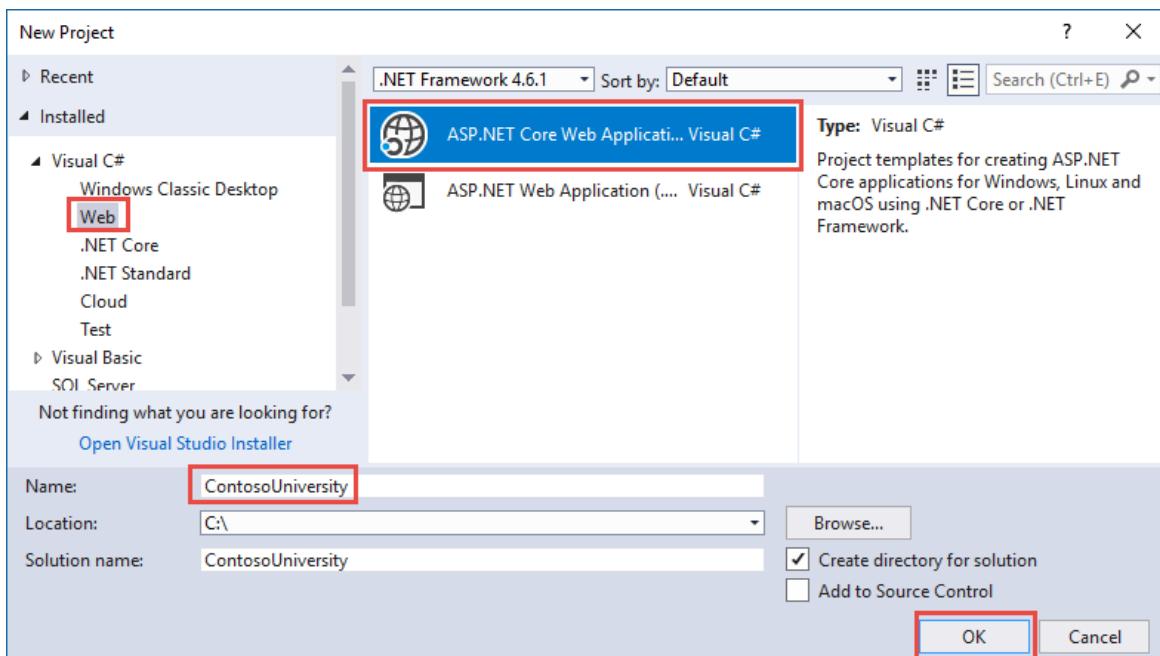
**Save**

O estilo de interface do usuário desse site foi mantido perto do que é gerado pelos modelos internos, de modo que o tutorial possa se concentrar principalmente em como usar o Entity Framework.

## Criar um aplicativo Web ASP.NET Core MVC

Abra o Visual Studio e crie um novo projeto Web ASP.NET Core C# chamado "ContosoUniversity".

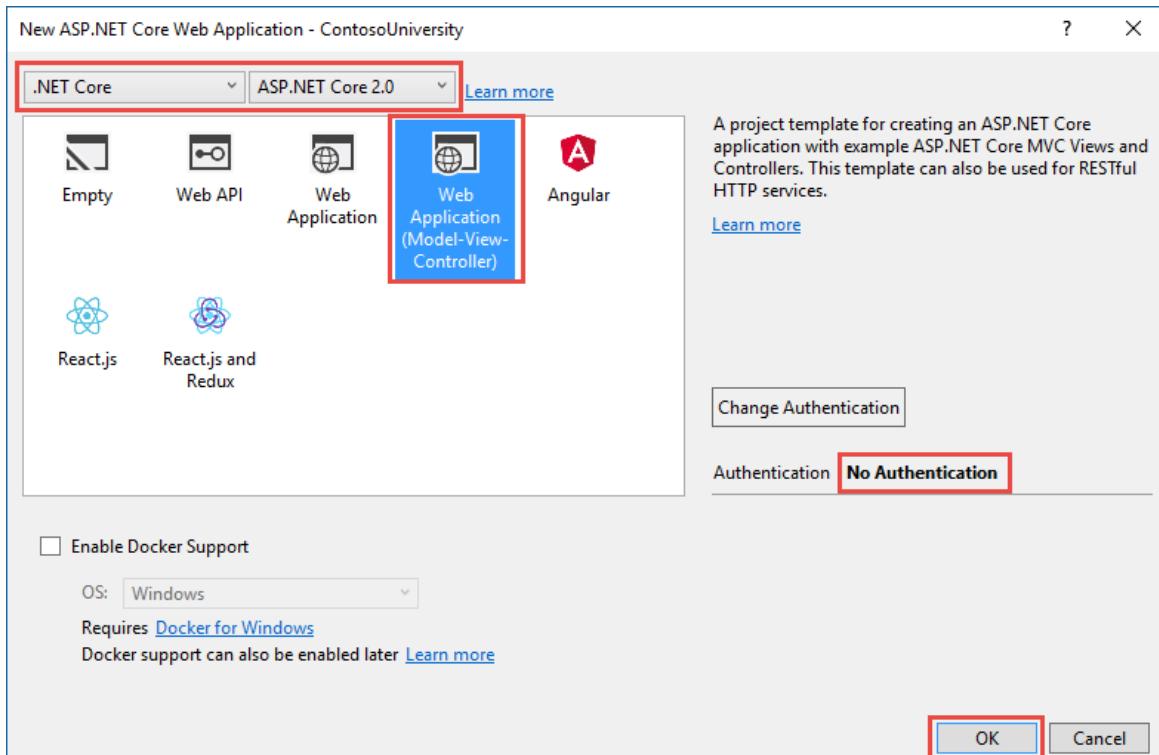
- No menu **Arquivo**, selecione **Novo > Projeto**.
- No painel esquerdo, selecione **Instalado > Visual C# > Web**.
- Selecione o modelo de projeto **Aplicativo Web ASP.NET Core**.
- Insira **ContosoUniversity** como o nome e clique em **OK**.



- Aguarde a caixa de diálogo **Novo Aplicativo Web ASP.NET Core (.NET Core)** ser exibida
- Selecione **ASP.NET Core 2.0** e o modelo **Aplicativo Web (Model-View-Controller)**.

**Observação:** este tutorial exige o ASP.NET Core 2.0 e o EF Core 2.0 ou posterior – verifique se o **ASP.NET Core 1.1** não está selecionado.

- Verifique se a opção **Autenticação** está definida como **Sem Autenticação**.
- Clique em **OK**



## Configurar o estilo do site

Algumas alterações simples configurarão o menu do site, o layout e a home page.

Abra *Views/Shared/\_Layout.cshtml* e faça as seguintes alterações:

- Altere cada ocorrência de "ContosoUniversity" para "Contoso University". Há três ocorrências.

- Adicione entradas de menu para **Alunos, Cursos, Instrutores e Departamentos** e exclua a entrada de menu **Contato**.

As alterações são realçadas.

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - Contoso University</title>

    <environment names="Development">
        <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
        <link rel="stylesheet" href("~/css/site.css" />
    </environment>
    <environment names="Staging,Production">
        <link rel="stylesheet" href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/css/bootstrap.min.css"
              asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"
              asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-test-
              value="absolute" />
        <link rel="stylesheet" href "~/css/site.min.css" asp-append-version="true" />
    </environment>

</head>
<body>
    <nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-
collapse">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a asp-area="" asp-controller="Home" asp-action="Index" class="navbar-brand">Contoso
University</a>
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li><a asp-area="" asp-controller="Home" asp-action="Index">Home</a></li>
                    <li><a asp-area="" asp-controller="Home" asp-action="About">About</a></li>
                    <li><a asp-area="" asp-controller="Students" asp-action="Index">Students</a></li>
                    <li><a asp-area="" asp-controller="Courses" asp-action="Index">Courses</a></li>
                    <li><a asp-area="" asp-controller="Instructors" asp-action="Index">Instructors</a></li>
                    <li><a asp-area="" asp-controller="Departments" asp-action="Index">Departments</a></li>
                </ul>
            </div>
        </div>
    </nav>
    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; 2017 - Contoso University</p>
        </footer>
    </div>

    <environment names="Development">
        <script src="~/lib/jquery/dist/jquery.js"></script>
        <script src="~/lib/bootstrap/dist/js/bootstrap.js"></script>
        <script src "~/js/site.js" asp-append-version="true"></script>
    </environment>
    <environment names="Staging,Production">
        <script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-2.2.0.min.js"
              asp-fallback-src="~/lib/jquery/dist/jquery.min.js"
              asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-test-
              value="absolute" />
    </environment>

```

```

        asp-fallback-test="window.jQuery"
        crossorigin="anonymous"
        integrity="sha384-K+ctZQ+LL8q6tP7I94W+qzQsfRV2a+AfHIi9k8z819ggpc8X+Ytst4yBo/hH+8Fk">
    </script>
    <script src="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/bootstrap.min.js"
        asp-fallback-src="~/lib/bootstrap/dist/js/bootstrap.min.js"
        asp-fallback-test="window.jQuery && window.jQuery.fn && window.jQuery.fn.modal"
        crossorigin="anonymous"
        integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfWVxZxUPnCJA7l2mCWNIPG9mGCD8wGNICPD7Txa">
    </script>
    <script src="~/js/site.min.js" asp-append-version="true"></script>
</environment>

    @RenderSection("Scripts", required: false)
</body>
</html>

```

Em *Views/Home/Index.cshtml*, substitua o conteúdo do arquivo pelo seguinte código para substituir o texto sobre o ASP.NET e MVC pelo texto sobre este aplicativo:

```

@{
    ViewData["Title"] = "Home Page";
}

<div class="jumbotron">
    <h1>Contoso University</h1>
</div>
<div class="row">
    <div class="col-md-4">
        <h2>Welcome to Contoso University</h2>
        <p>
            Contoso University is a sample application that
            demonstrates how to use Entity Framework Core in an
            ASP.NET Core MVC web application.
        </p>
    </div>
    <div class="col-md-4">
        <h2>Build it from scratch</h2>
        <p>You can build the application by following the steps in a series of tutorials.</p>
        <p><a class="btn btn-default" href="https://docs.asp.net/en/latest/data/ef-mvc/intro.html">See the
        tutorial &raquo;</a></p>
    </div>
    <div class="col-md-4">
        <h2>Download it</h2>
        <p>You can download the completed project from GitHub.</p>
        <p><a class="btn btn-default" href="https://github.com/aspnet/Docs/tree/master/aspnetcore/data/ef-
        mvc/intro/samples/cu-final">See project source code &raquo;</a></p>
    </div>
</div>

```

Pressione CTRL+F5 para executar o projeto ou escolha **Depurar > Iniciar sem Depuração** no menu. Você verá a home page com guias para as páginas que você criará nestes tutoriais.

The screenshot shows a web browser window with the title bar "Home Page - Contoso L X". The address bar shows "localhost:5813". The page content is the "Contoso University" home page. At the top, there is a navigation bar with links for "Contoso University", "Home", "About", "Students", "Courses", "Instructors", and "Departments". Below the navigation bar, the main heading "Contoso University" is displayed in a large, bold font. Underneath the heading, the subheading "Welcome to Contoso University" is shown in a large, bold font. A descriptive text follows: "Contoso University is a sample application that demonstrates how to use Entity Framework Core 1.0 in an ASP.NET Core MVC 1.0 web application." Below this, the section "Build it from scratch" is introduced with the text "You can build the application by following the steps in a series of tutorials." A button labeled "See the tutorial »" is present. The next section, "Download it", is introduced with the text "You can download the completed project from GitHub." A button labeled "See project source code »" is present. At the bottom of the page, a copyright notice reads "© 2016 - Contoso University".

## Pacotes NuGet do Entity Framework Core

Para adicionar o suporte do EF Core a um projeto, instale o provedor de banco de dados que você deseja ter como destino. Este tutorial usa o SQL Server e o pacote de provedor é

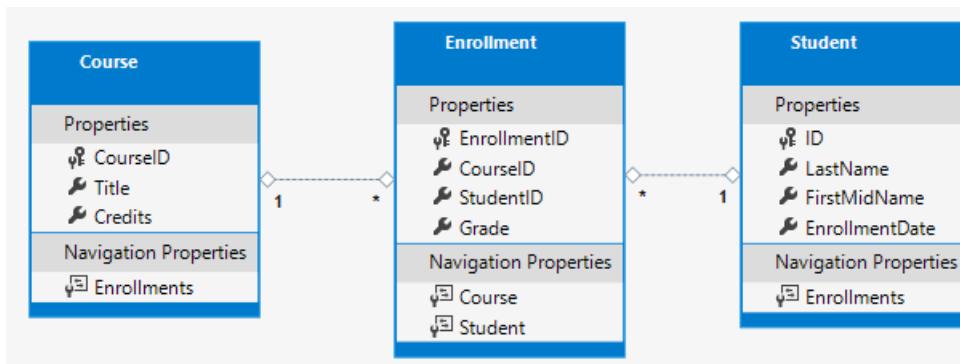
[Microsoft.EntityFrameworkCore.SqlServer](#). Esse pacote está incluído no [metapacote Microsoft.AspNetCore.App](#), portanto você não precisa referenciar o pacote se o aplicativo tem uma referência de pacote ao pacote [Microsoft.AspNetCore.App](#).

Este pacote e suas dependências ([Microsoft.EntityFrameworkCore](#) e [Microsoft.EntityFrameworkCore.Relational](#)) fornecem suporte de tempo de execução para o EF. Você adicionará um pacote de ferramentas posteriormente, no tutorial [Migrações](#).

Para obter informações sobre outros provedores de banco de dados que estão disponíveis para o Entity Framework Core, consulte [Provedores de banco de dados](#).

## Criar o modelo de dados

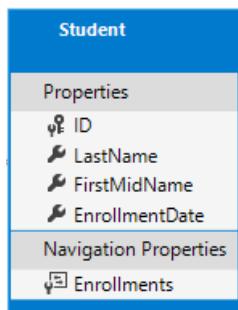
Em seguida, você criará as classes de entidade para o aplicativo Contoso University. Você começará com as três entidades a seguir.



Há uma relação um-para-muitos entre as entidades `Student` e `Enrollment`, e uma relação um-para-muitos entre as entidades `Course` e `Enrollment`. Em outras palavras, um aluno pode ser registrado em qualquer quantidade de cursos e um curso pode ter qualquer quantidade de alunos registrados.

Nas seções a seguir, você criará uma classe para cada uma dessas entidades.

### A entidade Student



Na pasta `Models`, crie um arquivo de classe chamado `Student.cs` e substitua o código de modelo pelo código a seguir.

```
using System;
using System.Collections.Generic;

namespace ContosoUniversity.Models
{
    public class Student
    {
        public int ID { get; set; }
        public string LastName { get; set; }
        public string FirstMidName { get; set; }
        public DateTime EnrollmentDate { get; set; }

        public ICollection<Enrollment> Enrollments { get; set; }
    }
}
```

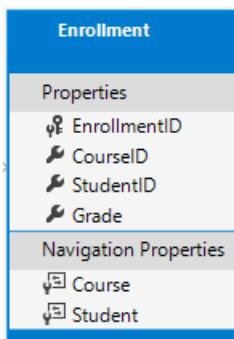
A propriedade `ID` se tornará a coluna de chave primária da tabela de banco de dados que corresponde a essa classe. Por padrão, o Entity Framework interpreta uma propriedade nomeada `ID` ou `classnameID` como a chave primária.

A propriedade `Enrollments` é uma [propriedade de navegação](#). As propriedades de navegação armazenam outras entidades que estão relacionadas a essa entidade. Nesse caso, a propriedade `Enrollments` de uma `Student entity` armazenará todas as entidades `Enrollment` relacionadas a essa entidade `Student`. Em outras palavras, se determinada linha Aluno no banco de dados tiver duas linhas Registro relacionadas (linhas que contêm o valor de chave primária do aluno na coluna de chave estrangeira StudentID), a propriedade de navegação `Enrollments` dessa entidade `Student` conterá as duas entidades `Enrollment`.

Se uma propriedade de navegação pode armazenar várias entidades (como em relações muitos para muitos ou

um-para-muitos), o tipo precisa ser uma lista na qual entradas podem ser adicionadas, excluídas e atualizadas, como `ICollection<T>`. Especifique `ICollection<T>` ou um tipo, como `List<T>` ou `HashSet<T>`. Se você especificar `ICollection<T>`, o EF criará uma coleção `HashSet<T>` por padrão.

## A entidade Enrollment



Na pasta `Models`, crie `Enrollment.cs` e substitua o código existente pelo seguinte código:

```
namespace ContosoUniversity.Models
{
    public enum Grade
    {
        A, B, C, D, F
    }

    public class Enrollment
    {
        public int EnrollmentID { get; set; }
        public int CourseID { get; set; }
        public int StudentID { get; set; }
        public Grade? Grade { get; set; }

        public Course Course { get; set; }
        public Student Student { get; set; }
    }
}
```

A propriedade `EnrollmentID` será a chave primária; essa entidade usa o padrão `classnameID` em vez de `ID` por si só, como você viu na entidade `student`. Normalmente, você escolhe um padrão e usa-o em todo o modelo de dados. Aqui, a variação ilustra que você pode usar qualquer um dos padrões. Em um [tutorial posterior](#), você verá como usar uma ID sem nome de classe facilita a implementação da herança no modelo de dados.

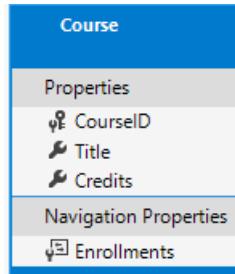
A propriedade `Grade` é um `enum`. O ponto de interrogação após a declaração de tipo `Grade` indica que a propriedade `Grade` permite valor nulo. Uma nota nula é diferente de uma nota zero – nulo significa que uma nota não é conhecida ou que ainda não foi atribuída.

A propriedade `StudentID` é uma chave estrangeira e a propriedade de navegação correspondente é `Student`. Uma entidade `Enrollment` é associada a uma entidade `Student`, de modo que a propriedade possa armazenar apenas uma única entidade `Student` (ao contrário da propriedade de navegação `Student.Enrollments` que você viu anteriormente, que pode armazenar várias entidades `Enrollment`).

A propriedade `CourseID` é uma chave estrangeira e a propriedade de navegação correspondente é `Course`. Uma entidade `Enrollment` está associada a uma entidade `Course`.

O Entity Framework interpreta uma propriedade como uma propriedade de chave estrangeira se ela é nomeada `<navigation property name><primary key property name>` (por exemplo, `StudentID` para a propriedade de navegação `Student`, pois a chave primária da entidade `Student` é `ID`). As propriedades de chave estrangeira também podem ser nomeadas apenas `<primary key property name>` (por exemplo, `CourseID`, pois a chave primária da entidade `Course` é `CourseID`).

## A entidade Course



Na pasta *Models*, crie *Course.cs* e substitua o código existente pelo seguinte código:

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Course
    {
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int CourseID { get; set; }
        public string Title { get; set; }
        public int Credits { get; set; }

        public ICollection<Enrollment> Enrollments { get; set; }
    }
}
```

A propriedade `Enrollments` é uma propriedade de navegação. Uma entidade `Course` pode estar relacionada a qualquer quantidade de entidades `Enrollment`.

Falaremos mais sobre o atributo `DatabaseGenerated` em um [tutorial posterior](#) desta série. Basicamente, esse atributo permite que você insira a chave primária do curso, em vez de fazer com que ela seja gerada pelo banco de dados.

## Criar o contexto de banco de dados

A classe principal que coordena a funcionalidade do Entity Framework para determinado modelo de dados é a classe de contexto de banco de dados. Você cria essa classe derivando-a da classe

`Microsoft.EntityFrameworkCore.DbContext`. No código, especifique quais entidades são incluídas no modelo de dados. Também personalize o comportamento específico do Entity Framework. Neste projeto, a classe é chamada `SchoolContext`.

Na pasta do projeto, crie uma pasta chamada *Dados*.

Na pasta *Dados*, crie um novo arquivo de classe chamado *SchoolContext.cs* e substitua o código de modelo pelo seguinte código:

```

using ContosoUniversity.Models;
using Microsoft.EntityFrameworkCore;

namespace ContosoUniversity.Data
{
    public class SchoolContext : DbContext
    {
        public SchoolContext(DbContextOptions<SchoolContext> options) : base(options)
        {
        }

        public DbSet<Course> Courses { get; set; }
        public DbSet<Enrollment> Enrollments { get; set; }
        public DbSet<Student> Students { get; set; }
    }
}

```

Esse código cria uma propriedade `DbSet` para cada conjunto de entidades. Na terminologia do Entity Framework, um conjunto de entidades normalmente corresponde a uma tabela de banco de dados, enquanto uma entidade corresponde a uma linha na tabela.

Você pode omitir as instruções `DbSet<Enrollment>` e `DbSet<Course>` e elas funcionarão da mesma maneira. O Entity Framework inclui-os de forma implícita porque a entidade `Student` referencia a entidade `Enrollment` e a entidade `Enrollment` referencia a entidade `Course`.

Quando o banco de dados é criado, o EF cria tabelas que têm nomes iguais aos nomes de propriedade `DbSet`. Em geral, os nomes de propriedade de coleções são plurais (Alunos em vez de Aluno), mas os desenvolvedores não concordam sobre se os nomes de tabela devem ser pluralizados ou não. Para esses tutoriais, você substituirá o comportamento padrão especificando nomes singulares de tabela no `DbContext`. Para fazer isso, adicione o código realçado a seguir após a última propriedade `DbSet`.

```

using ContosoUniversity.Models;
using Microsoft.EntityFrameworkCore;

namespace ContosoUniversity.Data
{
    public class SchoolContext : DbContext
    {
        public SchoolContext(DbContextOptions<SchoolContext> options) : base(options)
        {
        }

        public DbSet<Course> Courses { get; set; }
        public DbSet<Enrollment> Enrollments { get; set; }
        public DbSet<Student> Students { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Course>().ToTable("Course");
            modelBuilder.Entity<Enrollment>().ToTable("Enrollment");
            modelBuilder.Entity<Student>().ToTable("Student");
        }
    }
}

```

## Registrar o contexto com a injeção de dependência

O ASP.NET Core implementa a [injeção de dependência](#) por padrão. Serviços (como o contexto de banco de dados do EF) são registrados com injeção de dependência durante a inicialização do aplicativo. Os componentes que exigem esses serviços (como controladores MVC) recebem esses serviços por meio de

parâmetros do construtor. Você verá o código de construtor do controlador que obtém uma instância de contexto mais adiante neste tutorial.

Para registrar `SchoolContext` como um serviço, abra `Startup.cs` e adicione as linhas realçadas ao método `ConfigureServices`.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<SchoolContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddMvc();
}
```

O nome da cadeia de conexão é passado para o contexto com a chamada de um método em um objeto `DbContextOptionsBuilder`. Para o desenvolvimento local, o [sistema de configuração do ASP.NET Core](#) lê a cadeia de conexão do arquivo `appsettings.json`.

Adicione instruções `using` aos namespaces `ContosoUniversity.Data` e `Microsoft.EntityFrameworkCore` e, em seguida, compile o projeto.

```
using ContosoUniversity.Data;
using Microsoft.EntityFrameworkCore;
```

Abra o arquivo `appsettings.json` e adicione uma cadeia de conexão, conforme mostrado no exemplo a seguir.

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=ContosoUniversity1;Trusted_Connection=True;MultipleActiveResultSets=true"
  },
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Warning"
    }
  }
}
```

## SQL Server Express LocalDB

A cadeia de conexão especifica um banco de dados LocalDB do SQL Server. LocalDB é uma versão leve do Mecanismo de Banco de Dados do SQL Server Express destinado ao desenvolvimento de aplicativos, e não ao uso em produção. O LocalDB é iniciado sob demanda e executado no modo de usuário e, portanto, não há nenhuma configuração complexa. Por padrão, o LocalDB cria arquivos de banco de dados `.mdf` no diretório `C:/Users/<user>`.

## Adicionar um código para inicializar o banco de dados com os dados de teste

O Entity Framework criará um banco de dados vazio para você. Nesta seção, você escreve um método que é chamado depois que o banco de dados é criado para populá-lo com os dados de teste.

Aqui, você usará o método `EnsureCreated` para criar o banco de dados automaticamente. Em um [tutorial posterior](#), você verá como manipular as alterações do modelo usando as Migrações do Code First para alterar o esquema de banco de dados, em vez de remover e recriar o banco de dados.

Na pasta *Data*, crie um novo arquivo de classe chamado *DbInitializer.cs* e substitua o código de modelo pelo código a seguir, que faz com que um banco de dados seja criado, quando necessário, e carrega dados de teste no novo banco de dados.

```
using ContosoUniversity.Models;
using System;
using System.Linq;

namespace ContosoUniversity.Data
{
    public static class DbInitializer
    {
        public static void Initialize(SchoolContext context)
        {
            context.Database.EnsureCreated();

            // Look for any students.
            if (context.Students.Any())
            {
                return; // DB has been seeded
            }

            var students = new Student[]
            {
                new Student{FirstMidName="Carson", LastName="Alexander", EnrollmentDate=DateTime.Parse("2005-09-01")},
                new Student{FirstMidName="Meredith", LastName="Alonso", EnrollmentDate=DateTime.Parse("2002-09-01")},
                new Student{FirstMidName="Arturo", LastName="Anand", EnrollmentDate=DateTime.Parse("2003-09-01")},
                new Student{FirstMidName="Gytis", LastName="Barzdukas", EnrollmentDate=DateTime.Parse("2002-09-01")},
                new Student{FirstMidName="Yan", LastName="Li", EnrollmentDate=DateTime.Parse("2002-09-01")},
                new Student{FirstMidName="Peggy", LastName="Justice", EnrollmentDate=DateTime.Parse("2001-09-01")},
                new Student{FirstMidName="Laura", LastName="Norman", EnrollmentDate=DateTime.Parse("2003-09-01")},
                new Student{FirstMidName="Nino", LastName="Olivetto", EnrollmentDate=DateTime.Parse("2005-09-01")}
            };
            foreach (Student s in students)
            {
                context.Students.Add(s);
            }
            context.SaveChanges();

            var courses = new Course[]
            {
                new Course{CourseID=1050, Title="Chemistry", Credits=3},
                new Course{CourseID=4022, Title="Microeconomics", Credits=3},
                new Course{CourseID=4041, Title="Macroeconomics", Credits=3},
                new Course{CourseID=1045, Title="Calculus", Credits=4},
                new Course{CourseID=3141, Title="Trigonometry", Credits=4},
                new Course{CourseID=2021, Title="Composition", Credits=3},
                new Course{CourseID=2042, Title="Literature", Credits=4}
            };
            foreach (Course c in courses)
            {
                context.Courses.Add(c);
            }
            context.SaveChanges();

            var enrollments = new Enrollment[]
            {
                new Enrollment{StudentID=1, CourseID=1050, Grade=Grade.A},
                new Enrollment{StudentID=1, CourseID=4022, Grade=Grade.C},
                new Enrollment{StudentID=1, CourseID=4041, Grade=Grade.B},
            };
        }
    }
}
```

```

        new Enrollment{StudentID=2,CourseID=1045,Grade=Grade.B},
        new Enrollment{StudentID=2,CourseID=3141,Grade=Grade.F},
        new Enrollment{StudentID=2,CourseID=2021,Grade=Grade.F},
        new Enrollment{StudentID=3,CourseID=1050},
        new Enrollment{StudentID=4,CourseID=1050},
        new Enrollment{StudentID=4,CourseID=4022,Grade=Grade.F},
        new Enrollment{StudentID=5,CourseID=4041,Grade=Grade.C},
        new Enrollment{StudentID=6,CourseID=1045},
        new Enrollment{StudentID=7,CourseID=3141,Grade=Grade.A},
    };
    foreach (Enrollment e in enrollments)
    {
        context.Enrollments.Add(e);
    }
    context.SaveChanges();
}
}
}

```

O código verifica se há alunos no banco de dados e, se não há, ele pressupõe que o banco de dados é novo e precisa ser propagado com os dados de teste. Ele carrega os dados de teste em matrizes em vez de em coleções `List<T>` para otimizar o desempenho.

Em `Program.cs`, modifique o método `Main` para fazer o seguinte na inicialização do aplicativo:

- Obtenha uma instância de contexto de banco de dados do contêiner de injecção de dependência.
- Chame o método de semente passando a ele o contexto.
- Descarte o contexto quando o método de semente for concluído.

```

public static void Main(string[] args)
{
    var host = BuildWebHost(args);

    using (var scope = host.Services.CreateScope())
    {
        var services = scope.ServiceProvider;
        try
        {
            var context = services.GetRequiredService<SchoolContext>();
            DbInitializer.Initialize(context);
        }
        catch (Exception ex)
        {
            var logger = services.GetRequiredService<ILogger<Program>>();
            logger.LogError(ex, "An error occurred while seeding the database.");
        }
    }

    host.Run();
}

```

Adicione instruções `using`:

```

using Microsoft.Extensions.DependencyInjection;
using ContosoUniversity.Data;

```

Nos tutoriais mais antigos, você poderá ver um código semelhante no método `Configure` em `Startup.cs`.

Recomendamos que você use o método `Configure` apenas para configurar o pipeline de solicitação. O código de inicialização do aplicativo pertence ao método `Main`.

Agora, na primeira vez que você executar o aplicativo, o banco de dados será criado e propagado com os dados

de teste. Sempre que você alterar o modelo de dados, exclua o banco de dados, atualize o método de semente e comece novamente com um novo banco de dados da mesma maneira. Nos próximos tutoriais, você verá como modificar o banco de dados quando o modelo de dados for alterado, sem excluí-lo e recriá-lo.

## Criar um controlador e exibições

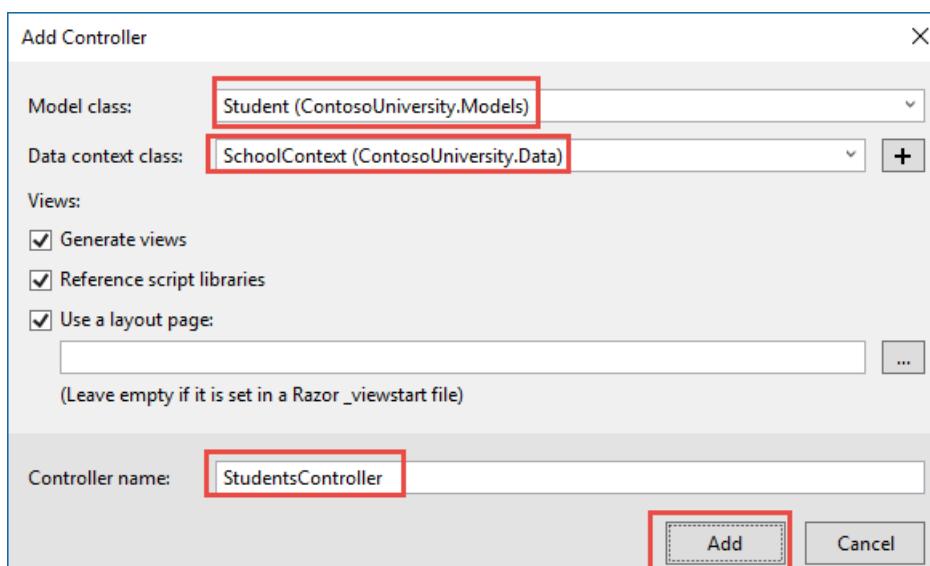
Em seguida, você usará o mecanismo de scaffolding no Visual Studio para adicionar um controlador MVC e exibições que usam o EF para consultar e salvar dados.

A criação automática de métodos de ação CRUD e exibições é conhecida como scaffolding. O scaffolding difere da geração de código, em que o código gerado por scaffolding é um ponto de partida que você pode modificar de acordo com seus requisitos, enquanto que normalmente o código gerado não é modificado. Quando precisar personalizar o código gerado, use classes parciais ou regenere o código quando as coisas mudarem.

- Clique com o botão direito do mouse na pasta **Controladores** no **Gerenciador de Soluções** e selecione **Adicionar > Novo Item Gerado por Scaffolding**.

Se a caixa de diálogo **Adicionar Dependências do MVC** for exibida:

- [Atualize o Visual Studio para a última versão](#). Versões do Visual Studio anteriores a 15.5 mostram essa caixa de diálogo.
- Se não puder atualizar, selecione **ADICIONAR** e, em seguida, siga as etapas de adição do controlador novamente.
- Na caixa de diálogo **Adicionar Scaffolding**:
  - Selecione **Controlador MVC com exibições, usando o Entity Framework**.
  - Clique em **Adicionar**.
- Na caixa de diálogo **Adicionar Controlador**:
  - Na **classe Model**, selecione **Aluno**.
  - Na **Classe de contexto de dados** selecione **SchoolContext**.
  - Aceite o **StudentsController** padrão como o nome.
  - Clique em **Adicionar**.



Quando você clica em **Adicionar**, o mecanismo de scaffolding do Visual Studio cria um arquivo *StudentsController.cs* e um conjunto de exibições (arquivos *.cshtml*) que funcionam com o controlador.

(O mecanismo de scaffolding também poderá criar o contexto de banco de dados para você se não criá-lo manualmente primeiro como fez anteriormente para este tutorial. Especifique uma nova classe de contexto na caixa **Adicionar Controlador** clicando no sinal de adição à direita de **Classe de contexto de dados**. Em seguida, o Visual Studio criará a classe `DbContext`, bem como o controlador e as exibições.)

Você observará que o controlador usa um `SchoolContext` como parâmetro de construtor.

```
namespace ContosoUniversity.Controllers
{
    public class StudentsController : Controller
    {
        private readonly SchoolContext _context;

        public StudentsController(SchoolContext context)
        {
            _context = context;
        }
    }
}
```

A injeção de dependência do ASP.NET Core é responsável por passar uma instância de `SchoolContext` para o controlador. Você configurou isso no arquivo `Startup.cs` anteriormente.

O controlador contém um método de ação `Index`, que exibe todos os alunos no banco de dados. O método obtém uma lista de alunos do conjunto de entidades `Students` pela leitura da propriedade `Students` da instância de contexto de banco de dados:

```
public async Task<IActionResult> Index()
{
    return View(await _context.Students.ToListAsync());
}
```

Você aprenderá sobre os elementos de programação assíncronos nesse código mais adiante no tutorial.

A exibição `Views/Students/Index.cshtml` mostra esta lista em uma tabela:

```

@model IEnumerable<ContosoUniversity.Models.Student>

 @{
     ViewData["Title"] = "Index";
 }

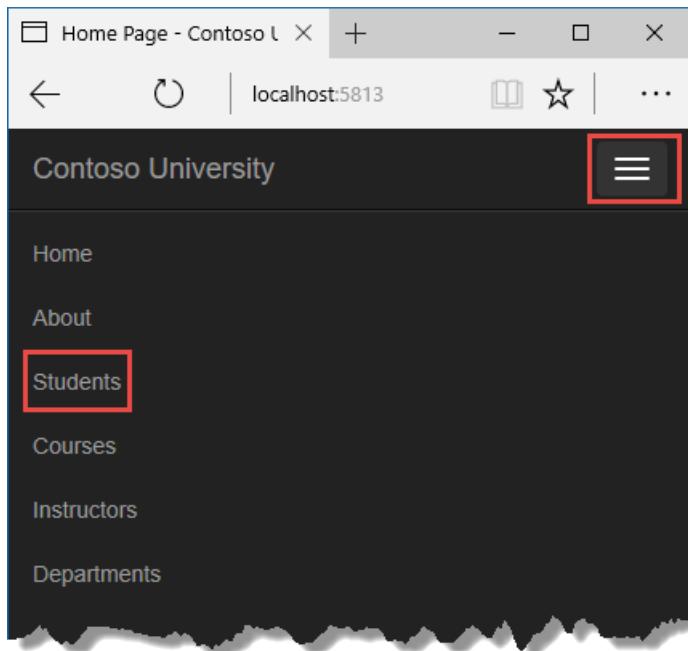
<h2>Index</h2>

<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.LastName)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.FirstMidName)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.EnrollmentDate)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.LastName)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.FirstMidName)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.EnrollmentDate)
            </td>
            <td>
                <a asp-action="Edit" asp-route-id="@item.ID">Edit</a> |
                <a asp-action="Details" asp-route-id="@item.ID">Details</a> |
                <a asp-action="Delete" asp-route-id="@item.ID">Delete</a>
            </td>
        </tr>
}
    </tbody>
</table>

```

Pressione CTRL+F5 para executar o projeto ou escolha **Depurar > Iniciar sem Depuração** no menu.

Clique na guia Alunos para ver os dados de teste inserido pelo método `DbInitializer.Initialize`. Dependendo da largura da janela do navegador, você verá o link da guia `Student` na parte superior da página ou precisará clicar no ícone de navegação no canto superior direito para ver o link.



Last Name	First Mid Name	Enrollment Date	
Alexander	Carson	9/1/2005 12:00:00 AM	Edit   Details   Delete
Alonso	Meredith	9/1/2002 12:00:00 AM	Edit   Details   Delete
Anand	Arturo	9/1/2003 12:00:00 AM	Edit   Details   Delete
Barzdukas	Gytis	9/1/2002 12:00:00 AM	Edit   Details   Delete

## Exibir o banco de dados

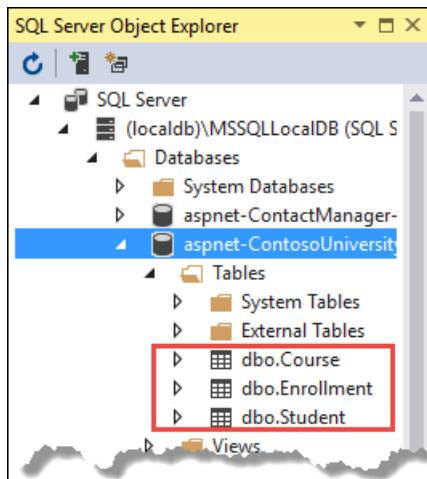
Quando você iniciou o aplicativo, o método `DbInitializer.Initialize` chamou `EnsureCreated`. O EF observou que não havia nenhum banco de dados e, portanto, ele criou um; em seguida, o restante do código do método `Initialize` populou o banco de dados com os dados. Use o **SSOX** (Pesquisador de Objetos do SQL Server) para exibir o banco de dados no Visual Studio.

Feche o navegador.

Se a janela do SSOX ainda não estiver aberta, selecione-a no menu **Exibir** do Visual Studio.

No SSOX, clique em **(localdb)\MSSQLLocalDB > Bancos de Dados** e, em seguida, clique na entrada do nome do banco de dados que está na cadeia de conexão no arquivo `appsettings.json`.

Expanda o nó **Tabelas** para ver as tabelas no banco de dados.



Clique com o botão direito do mouse na tabela **Aluno** e clique em **Exibir Dados** para ver as colunas criadas e as linhas que foram inseridas na tabela.

	ID	EnrollmentDate	FirstMidName	LastName
▶	1	9/1/2005 12:00:...	Carson	Alexander
	2	9/1/2002 12:00:...	Meredith	Alonso
	3	9/1/2003 12:00:...	Arturo	Anand
	4	9/1/2002 12:00:...	Gytis	Barzdukas
	5	9/1/2002 12:00:...	Yan	Li

Os arquivos de banco de dados **.mdf** e **.ldf** estão na pasta **C:\Users\**.

Como você está chamando `EnsureCreated` no método inicializador executado na inicialização do aplicativo, agora você pode fazer uma alteração na classe `Student`, excluir o banco de dados, executar novamente o aplicativo e o banco de dados será recriado automaticamente para que ele corresponda à alteração. Por exemplo, se você adicionar uma propriedade `EmailAddress` à classe `Student`, verá uma nova coluna `EmailAddress` na tabela recriada.

## Convenções

A quantidade de código feita para que o Entity Framework possa criar um banco de dados completo para você é mínima, devido ao uso de convenções ou de suposições feitas pelo Entity Framework.

- Os nomes de propriedades `DbSet` são usadas como nomes de tabela. Para entidades não referenciadas por uma propriedade `DbSet`, os nomes de classe de entidade são usados como nomes de tabela.
- Os nomes de propriedade de entidade são usados para nomes de coluna.
- As propriedades de entidade que são nomeadas ID ou classnameID são reconhecidas como propriedades de chave primária.
- Uma propriedade é interpretada como uma propriedade de chave estrangeira se ela é nomeada (por exemplo, `StudentID` para a propriedade de navegação `Student`, pois a chave primária da entidade `Student` é `ID`). As propriedades de chave estrangeira também podem ser nomeadas apenas (por exemplo, `EnrollmentID`, pois a chave primária da entidade `Enrollment` é `EnrollmentID`).

O comportamento convencional pode ser substituído. Por exemplo, você pode especificar os nomes de tabela de forma explícita, conforme visto anteriormente neste tutorial. Além disso, você pode definir nomes de coluna e qualquer propriedade como a chave primária ou chave estrangeira, como você verá em um [tutorial posterior](#).

desta série.

## Código assíncrono

A programação assíncrona é o modo padrão do ASP.NET Core e EF Core.

Um servidor Web tem um número limitado de threads disponíveis e, em situações de alta carga, todos os threads disponíveis podem estar em uso. Quando isso acontece, o servidor não pode processar novas solicitações até que os threads são liberados. Com um código síncrono, muitos threads podem ser vinculados enquanto realmente não estão fazendo nenhum trabalho porque estão aguardando a conclusão da E/S. Com um código assíncrono, quando um processo está aguardando a conclusão da E/S, seu thread é liberado para o servidor para ser usado para processar outras solicitações. Como resultado, o código assíncrono permite que os recursos do servidor sejam usados com mais eficiência, e o servidor fica capacitado a manipular mais tráfego sem atrasos.

O código assíncrono introduz uma pequena quantidade de sobrecarga em tempo de execução, mas para situações de baixo tráfego, o impacto no desempenho é insignificante, ao passo que, em situações de alto tráfego, a melhoria de desempenho potencial é significativa.

No código a seguir, a palavra-chave `async`, o valor retornado `Task<T>`, a palavra-chave `await` e o método `ToListAsync` fazem o código ser executado de forma assíncrona.

```
public async Task<IActionResult> Index()
{
    return View(await _context.Students.ToListAsync());
}
```

- A palavra-chave `async` instrui o compilador a gerar retornos de chamada para partes do corpo do método e a criar automaticamente o objeto `Task<IActionResult>` que é retornado.
- O tipo de retorno `Task<IActionResult>` representa um trabalho em andamento com um resultado do tipo `IActionResult`.
- A palavra-chave `await` faz com que o compilador divida o método em duas partes. A primeira parte termina com a operação que é iniciada de forma assíncrona. A segunda parte é colocada em um método de retorno de chamada que é chamado quando a operação é concluída.
- `ToListAsync` é a versão assíncrona do método de extensão `ToList`.

Algumas coisas a serem consideradas quando você estiver escrevendo um código assíncrono que usa o Entity Framework:

- Somente instruções que fazem com que consultas ou comandos sejam enviados ao banco de dados são executadas de forma assíncrona. Isso inclui, por exemplo, `ToListAsync`, `SingleOrDefaultAsync` e `SaveChangesAsync`. Isso não inclui, por exemplo, instruções que apenas alteram um `IQueryable`, como `var students = context.Students.Where(s => s.LastName == "Davolio")`.
- Um contexto do EF não é thread-safe: não tente realizar várias operações em paralelo. Quando você chamar qualquer método assíncrono do EF, sempre use a palavra-chave `await`.
- Se desejar aproveitar os benefícios de desempenho do código assíncrono, verifique se os pacotes de biblioteca que você está usando (por exemplo, para paginação) também usam o código assíncrono se eles chamam métodos do Entity Framework que fazem com que consultas sejam enviadas ao banco de dados.

Para obter mais informações sobre a programação assíncrona no .NET, consulte [Visão geral da programação assíncrona](#).

## Resumo

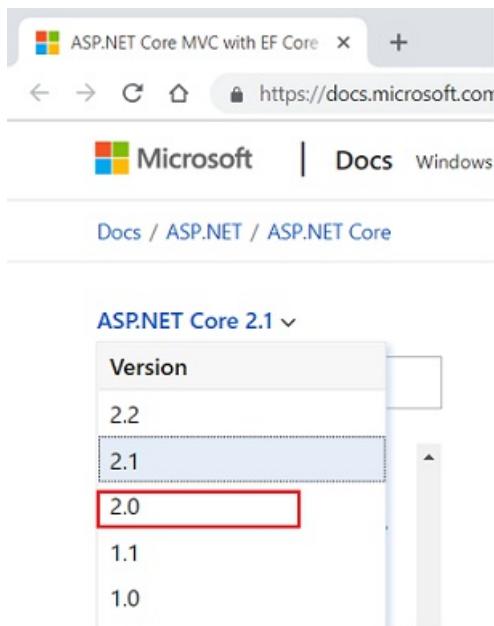
Agora, você criou um aplicativo simples que usa o Entity Framework Core e o LocalDB do SQL Server Express para armazenar e exibir dados. No tutorial a seguir, você aprenderá a executar operações CRUD (criar, ler, atualizar e excluir) básicas.

[AVANÇAR](#)

# ASP.NET Core MVC com EF Core – CRUD – 2 de 10

01/11/2018 • 38 minutes to read • [Edit Online](#)

Este tutorial não foi atualizado para o ASP.NET Core 2.1. A versão deste tutorial para o ASP.NET Core 2.0 pode ser consultada selecionando **ASP.NET Core 2.0** acima do sumário ou na parte superior da página:



A versão deste tutorial do [Razor Pages](#) para o ASP.NET Core 2.1 conta com várias melhorias em relação à versão 2.0.

O tutorial do 2.0 ensina a usar o ASP.NET Core MVC e o Entity Framework Core com controladores e exibições. O tutorial do Razor Pages foi atualizado com os seguintes aprimoramentos:

- É mais fácil de acompanhar. Por exemplo, o código de scaffolding foi bastante simplificado.
- Fornece mais práticas recomendadas do EF Core.
- Usa consultas mais eficientes.
- Usa a API do EF Core mais recente.

Por [Tom Dykstra](#) e [Rick Anderson](#)

O aplicativo web de exemplo Contoso University demonstra como criar aplicativos web do ASP.NET Core MVC usando o Entity Framework Core e o Visual Studio. Para obter informações sobre a série de tutoriais, consulte [o primeiro tutorial da série](#).

No tutorial anterior, você criou um aplicativo MVC que armazena e exibe dados usando o Entity Framework e o LocalDB do SQL Server. Neste tutorial, você examinará e personalizará o código CRUD (criar, ler, atualizar e excluir) que o scaffolding do MVC cria automaticamente para você em controladores e exibições.

## NOTE

É uma prática comum implementar o padrão de repositório para criar uma camada de abstração entre o controlador e a camada de acesso a dados. Para manter esses tutoriais simples e com foco no ensino de como usar o Entity Framework em si, eles não usam repositórios. Para obter informações sobre repositórios com o EF, consulte [o último tutorial desta série](#).

Neste tutorial, você trabalhará com as seguintes páginas da Web:

Details - Contoso Univ × +

localhost:5813/Students/Details/1

Contoso University Home About Students Courses Instructors Departments Register Log in

## Student

Last Name	Alexander	
First/Middle Name	Carson	
Enrollment Date	9/1/2005 12:00:00 AM	
Enrollments	Course Title	Grade
	Chemistry	A
	Microeconomics	C
	Macroeconomics	B

Create × + - □ ×

s/Create

Contoso University

## Create

### Student

Last Name	<input type="text"/>
First/Middle Name	<input type="text"/>
Enrollment Date	<input type="text"/>
<input type="button" value="Create"/>	

The screenshot shows a web browser window titled "Edit - c" with the URL "localhost". The main content area is titled "Edit" and has a sub-section "Student". It contains three text input fields: "LastName" with the value "Alexander", "FirstMidName" with the value "Carson", and "EnrollmentDate" with the value "2005-09-01T00:00:00.000". Below these fields is a "Save" button.

The screenshot shows a web browser window titled "Delete" with the URL "localhost". The main content area is titled "Delete" and has a sub-section "Student". It displays the same three fields as the previous screenshot: "LastName" (Alexander), "FirstMidName" (Carson), and "EnrollmentDate" (9/2/2005 12:00:00 AM). At the bottom are two buttons: "Delete" and "Back to List".

## Personalizar a página Detalhes

O código gerado por scaffolding da página Índice de Alunos omitiu a propriedade `Enrollments`, porque essa propriedade contém uma coleção. Na página **Detalhes**, você exibirá o conteúdo da coleção em uma tabela HTML.

Em `Controllers/StudentsController.cs`, o método de ação para a exibição Detalhes usa o método

`SingleOrDefaultAsync` para recuperar uma única entidade `Student`. Adicione um código que chama `Include`. Os métodos `ThenInclude` e `AsNoTracking`, conforme mostrado no código realçado a seguir.

```
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var student = await _context.Students
        .Include(s => s.Enrollments)
        .ThenInclude(e => e.Course)
        .AsNoTracking()
        .SingleOrDefaultAsync(m => m.ID == id);

    if (student == null)
    {
        return NotFound();
    }

    return View(student);
}
```

Os métodos `Include` e `ThenInclude` fazem com que o contexto carregue a propriedade de navegação `Student.Enrollments` e, dentro de cada registro, a propriedade de navegação `Enrollment.Course`. Você aprenderá mais sobre esses métodos no tutorial [Ler dados relacionados](#).

O método `AsNoTracking` melhora o desempenho em cenários em que as entidades retornadas não serão atualizadas no tempo de vida do contexto atual. Você aprenderá mais sobre `AsNoTracking` ao final deste tutorial.

## Dados de rota

O valor de chave que é passado para o método `Details` é obtido dos *dados de rota*. Dados de rota são dados que o associador de modelos encontrou em um segmento da URL. Por exemplo, a rota padrão especifica os segmentos de controlador, ação e ID:

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});
```

Na URL a seguir, a rota padrão mapeia Instructor como o controlador, Index como a ação e 1 como a ID; esses são valores de dados de rota.

```
http://localhost:1230/Instructor/Index/1?courseID=2021
```

A última parte da URL ("?courseID=2021") é um valor de cadeia de caracteres de consulta. O associador de modelos passará o valor da ID para o parâmetro `id` do método `Details` se você passá-lo como um valor de cadeia de caracteres de consulta:

```
http://localhost:1230/Instructor/Index?id=1&CourseID=2021
```

Na página Índice, as URLs de hiperlinks são criadas por instruções de auxiliar de marcação na exibição do Razor. No código Razor a seguir, o parâmetro `id` corresponde à rota padrão e, portanto, `id` é adicionada aos dados de

rota.

```
<a asp-action="Edit" asp-route-id="@item.ID">Edit</a>
```

Isso gera o seguinte HTML quando `item.ID` é 6:

```
<a href="/Students/Edit/6">Edit</a>
```

No código a seguir Razor, `studentID` não corresponde a um parâmetro na rota padrão e, portanto, ela é adicionada como uma cadeia de caracteres de consulta.

```
<a asp-action="Edit" asp-route-studentID="@item.ID">Edit</a>
```

Isso gera o seguinte HTML quando `item.ID` é 6:

```
<a href="/Students/Edit?studentID=6">Edit</a>
```

Para obter mais informações sobre os auxiliares de marca, confira [Auxiliares de Marca no ASP.NET Core](#).

### Adicionar registros à exibição Detalhes

Abra `Views/Students/Details.cshtml`. Cada campo é exibido usando auxiliares `DisplayNameFor` e `DisplayFor`, conforme mostrado no seguinte exemplo:

```
<dt>
    @Html.DisplayNameFor(model => model.LastName)
</dt>
<dd>
    @Html.DisplayFor(model => model.LastName)
</dd>
```

Após o último campo e imediatamente antes da marcação `</dd>` de fechamento, adicione o seguinte código para exibir uma lista de registros:

```
<dt>
    @Html.DisplayNameFor(model => model.Enrollments)
</dt>
<dd>
    <table class="table">
        <tr>
            <th>Course Title</th>
            <th>Grade</th>
        </tr>
        @foreach (var item in Model.Enrollments)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.Course.Title)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Grade)
                </td>
            </tr>
        }
    </table>
</dd>
```

Se o recuo do código estiver incorreto depois de colar o código, pressione CTRL-K-D para corrigi-lo.

Esse código percorre as entidades na propriedade de navegação `Enrollments`. Para cada registro, ele exibe o nome do curso e a nota. O título do curso é recuperado da entidade `Course`, que é armazenada na propriedade de navegação `course` da entidade `Enrollments`.

Execute o aplicativo, selecione a guia **Alunos** e clique no link **Detalhes** de um aluno. Você verá a lista de cursos e notas do aluno selecionado:

The screenshot shows a web browser window titled "Details - Contoso Univ". The address bar displays "localhost:5813/Students/Details/1". The page header includes the Contoso University logo and navigation links for Home, About, Students, Courses, Instructors, and Departments, along with Register and Log in buttons. The main content area is titled "Student". It shows a table with student information and enrollment details. The student's last name is Alexander and first/middle name is Carson, with an enrollment date of 9/1/2005 12:00:00 AM. The student is enrolled in three courses: Chemistry (Grade A), Microeconomics (Grade C), and Macroeconomics (Grade B).

Last Name	Alexander
First/Middle Name	Carson
Enrollment Date	9/1/2005 12:00:00 AM
Enrollments	
Course Title	Grade
Chemistry	A
Microeconomics	C
Macroeconomics	B

## Atualizar a página Criar

Em `StudentsController.cs`, modifique o método `HttpPost [Create]` adicionando um bloco try-catch e removendo a ID do atributo `Bind`.

```

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create(
    [Bind("EnrollmentDate,FirstMidName,LastName")] Student student)
{
    try
    {
        if (ModelState.IsValid)
        {
            _context.Add(student);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
    }
    catch (DbUpdateException /* ex */)
    {
        //Log the error (uncomment ex variable name and write a log.
        ModelState.AddModelError("", "Unable to save changes. " +
            "Try again, and if the problem persists " +
            "see your system administrator.");
    }
    return View(student);
}

```

Esse código adiciona a entidade Student criada pelo associador de modelos do ASP.NET Core MVC ao conjunto de entidades Student e, em seguida, salva as alterações no banco de dados. (Associador de modelos refere-se à funcionalidade do ASP.NET Core MVC que facilita o trabalho com os dados enviados por um formulário. Um associador de modelos converte os valores de formulário postados em tipos CLR e passa-os para o método de ação em parâmetros. Nesse caso, o associador de modelos cria uma instância de uma entidade Student usando valores de propriedade da coleção Form.)

Você removeu `ID` do atributo `Bind` porque a ID é o valor de chave primária que o SQL Server definirá automaticamente quando a linha for inserida. A entrada do usuário não define o valor da ID.

Além do atributo `Bind`, o bloco try-catch é a única alteração que você fez no código gerado por scaffolding. Se uma exceção que é derivada de `DbUpdateException` é capturada enquanto as alterações estão sendo salvas, uma mensagem de erro genérica é exibida. Às vezes, as exceções `DbUpdateException` são causadas por algo externo ao aplicativo, em vez de por um erro de programação e, portanto, o usuário é aconselhado a tentar novamente. Embora não implementado nesta amostra, um aplicativo de qualidade de produção registrará a exceção em log. Para obter mais informações, consulte a seção **Log para informações** em [Monitoramento e telemetria \(criando aplicativos de nuvem do mundo real com o Azure\)](#).

O atributo `ValidateAntiForgeryToken` ajuda a impedir ataques CSRF (solicitação intersite forjada). O token é injetado automaticamente na exibição pelo [FormTagHelper](#) e é incluído quando o formulário é enviado pelo usuário. O token é validado pelo atributo `ValidateAntiForgeryToken`. Para obter mais informações sobre o CSRF, consulte [Falsificação antissolicitação](#).

### **Observação de segurança sobre o excesso de postagem**

O atributo `Bind` que o código gerado por scaffolding inclui no método `Create` é uma maneira de proteger contra o excesso de postagem em cenários de criação. Por exemplo, suponha que a entidade Student inclua uma propriedade `Secret` que você não deseja que essa página da Web defina.

```

public class Student
{
    public int ID { get; set; }
    public string LastName { get; set; }
    public string FirstMidName { get; set; }
    public DateTime EnrollmentDate { get; set; }
    public string Secret { get; set; }
}

```

Mesmo se você não tiver um campo `Secret` na página da Web, um invasor poderá usar uma ferramenta como o Fiddler ou escrever um JavaScript para postar um valor de formulário `Secret`. Sem o atributo `Bind` limitando os campos que o associador de modelos usa quando ele cria uma instância de `Student`, o associador de modelos seleciona esse valor de formulário `Secret` e usa-o para criar a instância da entidade `Student`. Em seguida, seja qual for o valor que o invasor especificou para o campo de formulário `Secret`, ele é atualizado no banco de dados. A imagem a seguir mostra a ferramenta Fiddler adicionando o campo `Secret` (com o valor "OverPost") aos valores de formulário postados.

Em seguida, o valor "OverPost" é adicionado com êxito à propriedade `Secret` da linha inserida, embora você nunca desejou que a página da Web pudesse definir essa propriedade.

Impeça o excesso de postagem em cenários de edição lendo a entidade do banco de dados primeiro e, em seguida, chamando `TryUpdateModel`, passando uma lista explícita de propriedades permitidas. Esse é o método usado nestes tutoriais.

Uma maneira alternativa de impedir o excesso de postagem preferida por muitos desenvolvedores é usar modelos de exibição em vez de classes de entidade com o model binding. Inclua apenas as propriedades que você deseja atualizar no modelo de exibição. Quando o associador de modelos MVC tiver concluído, copie as propriedades do modelo de exibição para a instância da entidade, opcionalmente usando uma ferramenta como o AutoMapper. Use `_context.Entry` na instância de entidade para definir seu estado como `Unchanged` e, em seguida, defina `Property("PropertyName").IsModified` como verdadeiro em cada propriedade da entidade incluída no modelo de exibição. Esse método funciona nos cenários de edição e criação.

### Testar a página Criar

O código em `Views/Students/Create.cshtml` usa os auxiliares de marcação `label`, `input` e `span` (para mensagens de validação) para cada campo.

Execute o aplicativo, selecione a guia **Alunos** e, em seguida, clique em **Criar Novo**.

Insira nomes e uma data. Tente inserir uma data inválida se o navegador permitir fazer isso. (Alguns navegadores forçam o uso de um seletor de data.) Em seguida, clique em **Criar** para ver a mensagem de erro.

The screenshot shows a browser window with the title 'Create' and the URL 'localhost'. The page is titled 'Contoso University'. The 'Create' form for a 'Student' is displayed. It has three input fields: 'LastName' (value: 'Davolio'), 'FirstMidName' (value: 'Nancy'), and 'EnrollmentDate' (value: '09/31/2016'). Below the 'EnrollmentDate' field, an error message is shown: 'The value '09/31/2016' is not valid for EnrollmentDate.'. At the bottom of the form is a 'Create' button.

Essa é a validação do lado do servidor que você obtém por padrão; em um tutorial posterior, você verá como adicionar atributos que gerarão o código para a validação do lado do cliente também. O código realçado a seguir mostra a verificação de validação de modelo no método `Create`.

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create(
    [Bind("EnrollmentDate,FirstMidName,LastName")] Student student)
{
    try
    {
        if (ModelState.IsValid)
        {
            _context.Add(student);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
    }
    catch (DbUpdateException /* ex */)
    {
        //Log the error (uncomment ex variable name and write a log.
        ModelState.AddModelError("", "Unable to save changes. " +
            "Try again, and if the problem persists " +
            "see your system administrator.");
    }
    return View(student);
}
```

Altere a data para um valor válido e clique em **Criar** para ver o novo aluno ser exibido na página **Índice**.

## Atualizar a página Editar

Em `StudentController.cs`, o método `HttpGet` `Edit` (aquele sem o atributo `HttpPost`) usa o método `SingleOrDefaultAsync` para recuperar a entidade Student selecionada, como você viu no método `Details`. Não é necessário alterar esse método.

### Código `HttpPost` `Edit` recomendado: ler e atualizar

Substitua o método de ação `HttpPost` `Edit` pelo código a seguir.

```
[HttpPost, ActionName("Edit")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> EditPost(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
    var studentToUpdate = await _context.Students.SingleOrDefaultAsync(s => s.ID == id);
    if (await TryUpdateModelAsync<Student>(
        studentToUpdate,
        "",
        s => s.FirstMidName, s => s.LastName, s => s.EnrollmentDate))
    {
        try
        {
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        catch (DbUpdateException /* ex */)
        {
            //Log the error (uncomment ex variable name and write a log.)
            ModelState.AddModelError("", "Unable to save changes. " +
                "Try again, and if the problem persists, " +
                "see your system administrator.");
        }
    }
    return View(studentToUpdate);
}
```

Essas alterações implementam uma melhor prática de segurança para evitar o excesso de postagem. O scaffolder gerou um atributo `Bind` e adicionou a entidade criada pelo associador de modelos ao conjunto de entidades com um sinalizador `Modified`. Esse código não é recomendado para muitos cenários porque o atributo `Bind` limpa os dados pré-existentes nos campos não listados no parâmetro `Include`.

O novo código lê a entidade existente e chama `TryUpdateModel` para atualizar os campos na entidade recuperada **com base na entrada do usuário nos dados de formulário postados**. O controle automático de alterações do Entity Framework define o sinalizador `Modified` nos campos alterados pela entrada de formulário. Quando o método `SaveChanges` é chamado, o Entity Framework cria instruções SQL para atualizar a linha de banco de dados. Os conflitos de simultaneidade são ignorados e somente as colunas da tabela que foram atualizadas pelo usuário são atualizadas no banco de dados. (Um tutorial posterior mostra como lidar com conflitos de simultaneidade.)

Como uma melhor prática para evitar o excesso de postagem, os campos que você deseja que sejam atualizáveis pela página **Editar** estão na lista de permissões nos parâmetros `TryUpdateModel`. (A cadeia de caracteres vazia antes da lista de campos na lista de parâmetros destina-se ao uso de um prefixo com os nomes de campos de formulário.) Atualmente, não há nenhum campo extra que está sendo protegido, mas listar os campos que você deseja que o associador de modelos associe garante que, se você adicionar campos ao modelo de dados no

futuro, eles serão automaticamente protegidos até que você adicione-os aqui de forma explícita.

Como resultado dessas alterações, a assinatura do método `HttpPost` [Edit](#) é a mesma do método `HttpGet` [Edit](#); portanto, você já renomeou o método [EditPost](#).

### Código `HttpPost Edit` alternativo: criar e anexar

O código de edição `HttpPost` recomendado garante que apenas as colunas alteradas sejam atualizadas e preserva os dados nas propriedades que você não deseja incluir para o model binding. No entanto, a abordagem de primeira leitura exige uma leitura de banco de dados extra e pode resultar em um código mais complexo para lidar com conflitos de simultaneidade. Uma alternativa é anexar uma entidade criada pelo associador de modelos ao contexto do EF e marcá-la como modificada. (Não atualize o projeto com esse código; ele é mostrado somente para ilustrar uma abordagem opcional.)

```
public async Task<IActionResult> Edit(int id, [Bind("ID,EnrollmentDate,FirstMidName,LastName")] Student student)
{
    if (id != student.ID)
    {
        return NotFound();
    }
    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(student);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        catch (DbUpdateException /* ex */)
        {
            //Log the error (uncomment ex variable name and write a log.)
            ModelState.AddModelError("", "Unable to save changes. " +
                "Try again, and if the problem persists, " +
                "see your system administrator.");
        }
    }
    return View(student);
}
```

Use essa abordagem quando a interface do usuário da página da Web incluir todos os campos na entidade e puder atualizar qualquer um deles.

O código gerado por scaffolding usa a abordagem "criar e anexar", mas apenas captura exceções `DbUpdateConcurrencyException` e retorna códigos de erro 404. O exemplo mostrado captura qualquer exceção de atualização de banco de dados e exibe uma mensagem de erro.

### Estados da entidade

O contexto de banco de dados controla se as entidades em memória estão em sincronia com suas linhas correspondentes no banco de dados, e essas informações determinam o que acontece quando você chama o método `SaveChanges`. Por exemplo, quando você passa uma nova entidade para o método `Add`, o estado dessa entidade é definido como `Added`. Em seguida, quando você chama o método `SaveChanges`, o contexto de banco de dados emite um comando SQL INSERT.

Uma entidade pode estar em um dos seguintes estados:

- `Added`. A entidade ainda não existe no banco de dados. O método `SaveChanges` emite uma instrução INSERT.
- `Unchanged`. Nada precisa ser feito com essa entidade pelo método `SaveChanges`. Ao ler uma entidade do banco de dados, a entidade começa com esse status.

- `Modified`. Alguns ou todos os valores de propriedade da entidade foram modificados. O método `SaveChanges` emite uma instrução UPDATE.
- `Deleted`. A entidade foi marcada para exclusão. O método `SaveChanges` emite uma instrução DELETE.
- `Detached`. A entidade não está sendo controlada pelo contexto de banco de dados.

Em um aplicativo da área de trabalho, em geral, as alterações de estado são definidas automaticamente. Você lê uma entidade e faz alterações em alguns de seus valores de propriedade. Isso faz com que seu estado da entidade seja alterado automaticamente para `Modified`. Em seguida, quando você chama `SaveChanges`, o Entity Framework gera uma instrução SQL UPDATE que atualiza apenas as propriedades reais que você alterou.

Em um aplicativo Web, o `DbContext` que inicialmente lê uma entidade e exibe seus dados a serem editados é descartado depois que uma página é renderizada. Quando o método de ação `HttpPost Edit` é chamado, é feita uma nova solicitação da Web e você tem uma nova instância do `DbContext`. Se você ler novamente a entidade nesse novo contexto, simulará o processamento da área de trabalho.

Mas se você não desejar fazer a operação de leitura extra, precisará usar o objeto de entidade criado pelo associador de modelos. A maneira mais simples de fazer isso é definir o estado da entidade como Modificado, como é feito no código `HttpPost Edit` alternativo mostrado anteriormente. Em seguida, quando você chama `SaveChanges`, o Entity Framework atualiza todas as colunas da linha de banco de dados, porque o contexto não tem como saber quais propriedades foram alteradas.

Caso deseje evitar a abordagem de primeira leitura, mas também deseje que a instrução SQL UPDATE atualize somente os campos que o usuário realmente alterar, o código será mais complexo. É necessário salvar os valores originais de alguma forma (por exemplo, usando campos ocultos) para que eles estejam disponíveis quando o método `HttpPost Edit` for chamado. Em seguida, você pode criar uma entidade `Student` usando os valores originais, chamar o método `Attach` com a versão original da entidade, atualizar os valores da entidade para os novos valores e, em seguida, chamar `SaveChanges`.

## Testar a página Editar

Execute o aplicativo, selecione a guia **Alunos** e, em seguida, clique em um hiperlink **Editar**.

The screenshot shows a web browser window with the following details:

- Header:** The title bar says "Edit - c" and the address bar shows "localhost".
- Page Title:** "Contoso University"
- Content:**
  - Edit:** The main title of the form.
  - Student:** Subtitle or section of the form.
  - Fields:**
    - Last Name:** Input field containing "Alexander".
    - First Mid Name:** Input field containing "Carson".
    - Enrollment Date:** Input field containing "2005-09-01T00:00:00.000".
  - Buttons:** A "Save" button at the bottom of the form.

Altere alguns dos dados e clique em **Salvar**. A página **Índice** será aberta e você verá os dados alterados.

## Atualizar a página Excluir

Em `StudentController.cs`, o código de modelo para o método `HttpGet Delete` usa o método `SingleOrDefaultAsync` para recuperar a entidade Student selecionada, como você viu nos métodos Details e Edit. No entanto, para implementar uma mensagem de erro personalizada quando a chamada a `SaveChanges` falhar, você adicionará uma funcionalidade a esse método e à sua exibição correspondente.

Como você viu para operações de atualização e criação, as operações de exclusão exigem dois métodos de ação. O método chamado em resposta a uma solicitação GET mostra uma exibição que dá ao usuário uma oportunidade de aprovar ou cancelar a operação de exclusão. Se o usuário aprová-la, uma solicitação POST será criada. Quando isso acontece, o método `HttpPost Delete` é chamado e, em seguida, esse método executa, de fato, a operação de exclusão.

Você adicionará um bloco try-catch ao método `HttpPost Delete` para tratar os erros que podem ocorrer quando o banco de dados é atualizado. Se ocorrer um erro, o método `HttpPost Delete` chamará o método `HttpGet Delete`, passando a ele um parâmetro que indica que ocorreu um erro. Em seguida, o método `HttpGet Delete` exibe novamente a página de confirmação, junto com a mensagem de erro, dando ao usuário a oportunidade de cancelar ou tentar novamente.

Substitua o método de ação `HttpGet Delete` pelo código a seguir, que gerencia o relatório de erros.

```
public async Task<IActionResult> Delete(int? id, bool? saveChangesError = false)
{
    if (id == null)
    {
        return NotFound();
    }

    var student = await _context.Students
        .AsNoTracking()
        .SingleOrDefaultAsync(m => m.ID == id);
    if (student == null)
    {
        return NotFound();
    }

    if (saveChangesError.GetValueOrDefault())
    {
        ViewData["ErrorMessage"] =
            "Delete failed. Try again, and if the problem persists " +
            "see your system administrator.";
    }

    return View(student);
}
```

Este código aceita um parâmetro opcional que indica se o método foi chamado após uma falha ao salvar as alterações. Esse parâmetro é falso quando o método `HttpGet Delete` é chamado sem uma falha anterior. Quando ele é chamado pelo método `HttpPost Delete` em resposta a um erro de atualização de banco de dados, o parâmetro é verdadeiro, e uma mensagem de erro é passada para a exibição.

### A abordagem de primeira leitura para `HttpPost Delete`

Substitua o método de ação `HttpPost Delete` (chamado `DeleteConfirmed`) pelo código a seguir, que executa a operação de exclusão real e captura os erros de atualização de banco de dados.

```

[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var student = await _context.Students
        .AsNoTracking()
        .SingleOrDefaultAsync(m => m.ID == id);
    if (student == null)
    {
        return RedirectToAction(nameof(Index));
    }

    try
    {
        _context.Students.Remove(student);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    catch (DbUpdateException /* ex */)
    {
        //Log the error (uncomment ex variable name and write a log.)
        return RedirectToAction(nameof(Delete), new { id = id, saveChangesError = true });
    }
}

```

Esse código recupera a entidade selecionada e, em seguida, chama o método `Remove` para definir o status da entidade como `Deleted`. Quando `SaveChanges` é chamado, um comando SQL DELETE é gerado.

### A abordagem "criar e anexar" para `HttpPost Delete`

Se a melhoria do desempenho de um aplicativo de alto volume for uma prioridade, você poderá evitar uma consulta SQL desnecessária criando uma instância de uma entidade Student usando somente o valor de chave primária e, em seguida, definindo o estado da entidade como `Deleted`. Isso é tudo o que o Entity Framework precisa para excluir a entidade. (Não coloque esse código no projeto; ele está aqui apenas para ilustrar uma alternativa.)

```

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    try
    {
        Student studentToDelete = new Student() { ID = id };
        _context.Entry(studentToDelete).State = EntityState.Deleted;
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    catch (DbUpdateException /* ex */)
    {
        //Log the error (uncomment ex variable name and write a log.)
        return RedirectToAction(nameof(Delete), new { id = id, saveChangesError = true });
    }
}

```

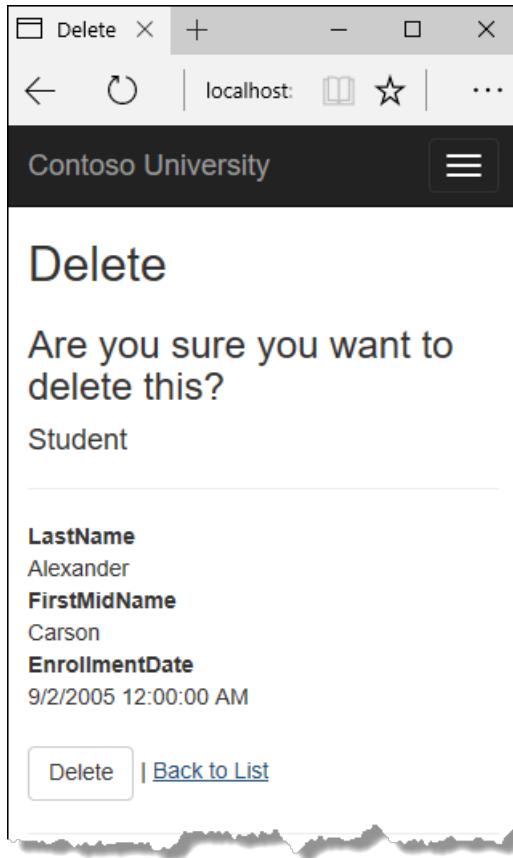
Se a entidade tiver dados relacionados, eles também deverão ser excluídos. Verifique se a exclusão em cascata está configurada no banco de dados. Com essa abordagem para a exclusão de entidade, o EF talvez não perceba que há entidades relacionadas a serem excluídas.

### Atualizar a exibição Excluir

Em `Views/Student/Delete.cshtml`, adicione uma mensagem de erro entre o cabeçalho h2 e o cabeçalho h3, conforme mostrado no seguinte exemplo:

```
<h2>Delete</h2>
<p class="text-danger">@ViewData["ErrorMessage"]</p>
<h3>Are you sure you want to delete this?</h3>
```

Execute o aplicativo, selecione a guia **Alunos** e, em seguida, clique em um hiperlink **Excluir**:



Clique em **Excluir**. A página Índice será exibida sem o aluno excluído. (Você verá um exemplo de código de tratamento de erro em ação no tutorial sobre simultaneidade.)

## Fechando conexões de banco de dados

Para liberar os recursos contidos em uma conexão de banco de dados, a instância de contexto precisa ser descartada assim que possível quando você tiver terminado. A [injeção de dependência](#) interna do ASP.NET Core cuida dessa tarefa para você.

Em *Startup.cs*, chame o [método de extensão AddDbContext](#) para provisionar a classe `DbContext` no contêiner de DI do ASP.NET Core. Esse método define o tempo de vida do serviço como `Scoped` por padrão. `Scoped` significa que o tempo de vida do objeto de contexto coincide com o tempo de vida da solicitação da Web, e o método `Dispose` será chamado automaticamente ao final da solicitação da Web.

## Manipulando transações

Por padrão, o Entity Framework implementa transações de forma implícita. Em cenários em que são feitas alterações em várias linhas ou tabelas e, em seguida, `SaveChanges` é chamado, o Entity Framework verifica automaticamente se todas as alterações tiveram êxito ou se falharam. Se algumas alterações forem feitas pela primeira vez e, em seguida, ocorrer um erro, essas alterações serão revertidas automaticamente. Para cenários em que você precisa de mais controle – por exemplo, se desejar incluir operações feitas fora do Entity Framework em uma transação –, consulte [Transações](#).

## Consultas sem controle

Quando um contexto de banco de dados recupera linhas de tabela e cria objetos de entidade que as representam, por padrão, ele controla se as entidades em memória estão em sincronia com o que está no banco de dados. Os dados em memória atuam como um cache e são usados quando uma entidade é atualizada. Esse cache costuma ser desnecessário em um aplicativo Web porque as instâncias de contexto são normalmente de curta duração (uma nova é criada e descartada para cada solicitação) e o contexto que lê uma entidade normalmente é descartado antes que essa entidade seja usada novamente.

Desabilite o controle de objetos de entidade em memória chamando o método `AsNoTracking`. Os cenários típicos em que talvez você deseje fazer isso incluem os seguintes:

- Durante o tempo de vida do contexto, não é necessário atualizar entidades nem que o EF [carregue automaticamente as propriedades de navegação com entidades recuperadas por consultas separadas](#). Com frequência, essas condições são atendidas nos métodos de ação `HttpGet` de um controlador.
- Você está executando uma consulta que recupera um volume grande de dados e apenas uma pequena parte dos dados retornados será atualizada. Pode ser mais eficiente desativar o controle para a consulta grande e executar uma consulta posteriormente para as poucas entidades que precisam ser atualizadas.
- Você deseja anexar uma entidade para atualizá-la, mas anteriormente, recuperou a mesma entidade para uma finalidade diferente. Como a entidade já está sendo controlada pelo contexto de banco de dados, não é possível anexar a entidade que você deseja alterar. Uma maneira de lidar com essa situação é chamar `AsNoTracking` na consulta anterior.

Para obter mais informações, consulte [Controle vs. Sem controle](#).

## Resumo

Agora, você tem um conjunto completo de páginas que executam operações CRUD simples para entidades Student. No próximo tutorial, você expandirá a funcionalidade da página **Índice** adicionando classificação, filtragem e paginação.

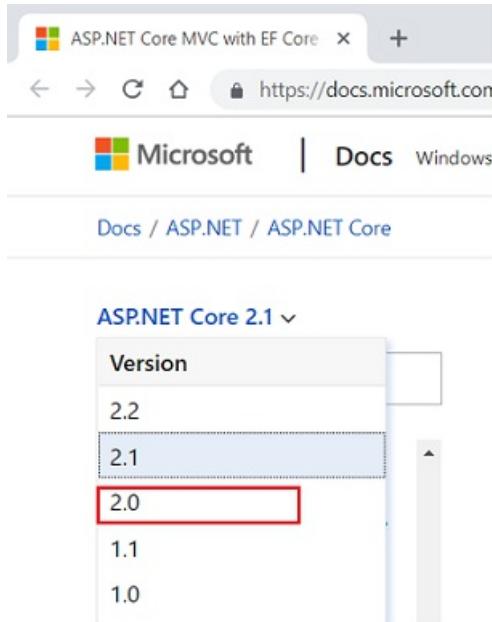
[ANTERIOR](#)

[PRÓXIMO](#)

# ASP.NET Core MVC com EF Core – classificação, filtro, paginação – 3 de 10

16/07/2018 • 27 minutes to read • [Edit Online](#)

Este tutorial não foi atualizado para o ASP.NET Core 2.1. A versão deste tutorial para o ASP.NET Core 2.0 pode ser consultada selecionando **ASP.NET Core 2.0** acima do sumário ou na parte superior da página:



The screenshot shows a browser window with the title "ASP.NET Core MVC with EF Core". The address bar shows the URL <https://docs.microsoft.com>. Below the address bar, there's a Microsoft logo and navigation links for "Docs" and "Windows". Under "Docs", the path "Docs / ASP.NET / ASP.NET Core" is visible. On the right side of the page, there's a dropdown menu titled "ASP.NET Core 2.1" with a dropdown arrow. The menu lists several versions: 2.2, 2.1, 2.0, 1.1, and 1.0. The version "2.0" is highlighted with a red border, indicating it is the selected version for this tutorial.

A versão deste tutorial do [Razor Pages](#) para o ASP.NET Core 2.1 conta com várias melhorias em relação à versão 2.0.

O tutorial do 2.0 ensina a usar o ASP.NET Core MVC e o Entity Framework Core com controladores e exibições.

O tutorial do Razor Pages foi atualizado com os seguintes aprimoramentos:

- É mais fácil de acompanhar. Por exemplo, o código de scaffolding foi bastante simplificado.
- Fornece mais práticas recomendadas do EF Core.
- Usa consultas mais eficientes.
- Usa a API do EF Core mais recente.

Por [Tom Dykstra](#) e [Rick Anderson](#)

O aplicativo web de exemplo Contoso University demonstra como criar aplicativos web do ASP.NET Core MVC usando o Entity Framework Core e o Visual Studio. Para obter informações sobre a série de tutoriais, consulte [o primeiro tutorial da série](#).

No tutorial anterior, você implementou um conjunto de páginas da Web para operações CRUD básicas para entidades Student. Neste tutorial você adicionará as funcionalidades de classificação, filtragem e paginação à página Índice de Alunos. Você também criará uma página que faz um agrupamento simples.

A ilustração a seguir mostra a aparência da página quando você terminar. Os títulos de coluna são links que o usuário pode clicar para classificar por essa coluna. Clicar em um título de coluna alterna repetidamente entre a ordem de classificação ascendente e descendente.

Last Name	First Name	Enrollment Date	
Alexander	Carson	9/1/2005 12:00:00 AM	Edit   Details   Delete
Alonso	Meredith	9/1/2002 12:00:00 AM	Edit   Details   Delete
Anand	Arturo	9/1/2003 12:00:00 AM	Edit   Details   Delete

## Adicionar links de classificação de coluna à página Índice de Alunos

Para adicionar uma classificação à página Índice de Alunos, você alterará o método `Index` do controlador Alunos e adicionará o código à exibição Índice de Alunos.

### Adicionar a funcionalidade de classificação ao método Index

Em `StudentsController.cs`, substitua o método `Index` pelo seguinte código:

```
public async Task<IActionResult> Index(string sortOrder)
{
    ViewData["NameSortParm"] = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
    ViewData["DateSortParm"] = sortOrder == "Date" ? "date_desc" : "Date";
    var students = from s in _context.Students
                  select s;
    switch (sortOrder)
    {
        case "name_desc":
            students = students.OrderByDescending(s => s.LastName);
            break;
        case "Date":
            students = students.OrderBy(s => s.EnrollmentDate);
            break;
        case "date_desc":
            students = students.OrderByDescending(s => s.EnrollmentDate);
            break;
        default:
            students = students.OrderBy(s => s.LastName);
            break;
    }
    return View(await students.AsNoTracking().ToListAsync());
}
```

Esse código recebe um parâmetro `sortOrder` da cadeia de caracteres de consulta na URL. O valor de cadeia de caracteres de consulta é fornecido pelo ASP.NET Core MVC como um parâmetro para o método de ação. O parâmetro será uma cadeia de caracteres "Name" ou "Date", opcionalmente, seguido de um sublinhado e a cadeia de caracteres "desc" para especificar a ordem descendente. A ordem de classificação crescente é padrão.

Na primeira vez que a página Índice é solicitada, não há nenhuma cadeia de caracteres de consulta. Os alunos são exibidos em ordem ascendente por sobrenome, que é o padrão, conforme estabelecido pelo caso fall-through na instrução `switch`. Quando o usuário clica em um hiperlink de título de coluna, o valor `sortOrder` apropriado é fornecido na cadeia de caracteres de consulta.

Os dois elementos `ViewData` (`NameSortParm` e `DateSortParm`) são usados pela exibição para configurar os hiperlinks de título de coluna com os valores de cadeia de caracteres de consulta apropriados.

```
public async Task<IActionResult> Index(string sortOrder)
{
    ViewData["NameSortParm"] = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
    ViewData["DateSortParm"] = sortOrder == "Date" ? "date_desc" : "Date";
    var students = from s in _context.Students
                  select s;
    switch (sortOrder)
    {
        case "name_desc":
            students = students.OrderByDescending(s => s.LastName);
            break;
        case "Date":
            students = students.OrderBy(s => s.EnrollmentDate);
            break;
        case "date_desc":
            students = students.OrderByDescending(s => s.EnrollmentDate);
            break;
        default:
            students = students.OrderBy(s => s.LastName);
            break;
    }
    return View(await students.AsNoTracking().ToListAsync());
}
```

Essas são instruções ternárias. A primeira delas especifica que o parâmetro `sortOrder` é nulo ou vazio, `NameSortParm` deve ser definido como "name\_desc"; caso contrário, ele deve ser definido como uma cadeia de caracteres vazia. Essas duas instruções permitem que a exibição defina os hiperlinks de título de coluna da seguinte maneira:

ORDEM DE CLASSIFICAÇÃO ATUAL	HIPERLINK DO SOBRENOME	HIPERLINK DE DATA
Sobrenome ascendente	descending	ascending
Sobrenome descendente	ascending	ascending
Data ascendente	ascending	descending
Data descendente	ascending	ascending

O método usa o LINQ to Entities para especificar a coluna pela qual classificar. O código cria uma variável `IQueryable` antes da instrução `switch`, modifica-a na instrução `switch` e chama o método `ToListAsync` após a instrução `switch`. Quando você cria e modifica variáveis `IQueryable`, nenhuma consulta é enviada para o banco de dados. A consulta não é executada até que você converta o objeto `IQueryable` em uma coleção chamando um método, como `ToListAsync`. Portanto, esse código resulta em uma única consulta que não é executada até a instrução `return View`.

Este código pode ficar detalhado com um grande número de colunas. [O último tutorial desta série](#) mostra como escrever um código que permite que você passe o nome da coluna `OrderBy` em uma variável de cadeia de caracteres.

## Adicionar hiperlinks de título de coluna à exibição Índice de Alunos

Substitua o código em *Views/Students/Index.cshtml* pelo código a seguir para adicionar hiperlinks de título de coluna. As linhas alteradas são realçadas.

```
@model IEnumerable<ContosoUniversity.Models.Student>

{@
    ViewData["Title"] = "Index";
}

<h2>Index</h2>

<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                <a asp-action="Index" asp-route-
sortOrder="@ViewData["NameSortParm"]">@Html.DisplayNameFor(model => model.LastName)</a>
            </th>
            <th>
                @Html.DisplayNameFor(model => model.FirstMidName)
            </th>
            <th>
                <a asp-action="Index" asp-route-
sortOrder="@ViewData["DateSortParm"]">@Html.DisplayNameFor(model => model.EnrollmentDate)</a>
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.LastName)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.FirstMidName)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.EnrollmentDate)
            </td>
            <td>
                <a asp-action="Edit" asp-route-id="@item.ID">Edit</a> |
                <a asp-action="Details" asp-route-id="@item.ID">Details</a> |
                <a asp-action="Delete" asp-route-id="@item.ID">Delete</a>
            </td>
        </tr>
}
    </tbody>
</table>
```

Esse código usa as informações nas propriedades `ViewData` para configurar hiperlinks com os valores de cadeia de caracteres de consulta apropriados.

Execute o aplicativo, selecione a guia **Alunos** e, em seguida, clique nos títulos de coluna **Sobrenome** e **Data de Registro** para verificar se a classificação funciona.

Last Name	First/Mid Name	Enrollment Date	
Alexander	Carson	9/2/2005 12:00:00 AM	Edit   Details   Delete
Alonso	Meredith	9/1/2002 12:00:00 AM	Edit   Details   Delete
Anand	Arturo	9/1/2003 12:00:00 AM	Edit   Details   Delete
Barzdukas	Gytis	9/1/2002 12:00:00 AM	Edit   Details   Delete

## Adicionar uma Caixa de Pesquisa à página Índice de Alunos

Para adicionar a filtragem à página Índice de Alunos, você adicionará uma caixa de texto e um botão Enviar à exibição e fará alterações correspondentes no método `Index`. A caixa de texto permitirá que você insira uma cadeia de caracteres a ser pesquisada nos campos de nome e sobrenome.

### Adicionar a funcionalidade de filtragem a método `Index`

Em `StudentsController.cs`, substitua o método `Index` pelo código a seguir (as alterações são realçadas).

```
public async Task<IActionResult> Index(string sortOrder, string searchString)
{
    ViewData["NameSortParm"] = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
    ViewData["DateSortParm"] = sortOrder == "Date" ? "date_desc" : "Date";
    ViewData["CurrentFilter"] = searchString;

    var students = from s in _context.Students
                  select s;
    if (!String.IsNullOrEmpty(searchString))
    {
        students = students.Where(s => s.LastName.Contains(searchString)
                           || s.FirstMidName.Contains(searchString));
    }
    switch (sortOrder)
    {
        case "name_desc":
            students = students.OrderByDescending(s => s.LastName);
            break;
        case "Date":
            students = students.OrderBy(s => s.EnrollmentDate);
            break;
        case "date_desc":
            students = students.OrderByDescending(s => s.EnrollmentDate);
            break;
        default:
            students = students.OrderBy(s => s.LastName);
            break;
    }
    return View(await students.AsNoTracking().ToListAsync());
}
```

Você adicionou um parâmetro `searchString` ao método `Index`. O valor de cadeia de caracteres de pesquisa é recebido em uma caixa de texto que você adicionará à exibição Índice. Você também adicionou à instrução LINQ uma cláusula `Where`, que seleciona somente os alunos cujo nome ou sobrenome contém a cadeia de caracteres de pesquisa. A instrução que adiciona a cláusula `Where` é executada somente se há um valor a ser pesquisado.

#### NOTE

Aqui você está chamando o método `Where` em um objeto `IQueryable`, e o filtro será processado no servidor. Em alguns cenários, você pode chamar o método `Where` como um método de extensão em uma coleção em memória. (Por exemplo, suponha que você altere a referência a `_context.Students`, de modo que em vez de um `DbSet` do EF, ela refencie um método de repositório que retorna uma coleção `IEnumerable`.) O resultado normalmente é o mesmo, mas em alguns casos pode ser diferente.

Por exemplo, a implementação do .NET Framework do método `Contains` executa uma comparação que diferencia maiúsculas de minúsculas por padrão, mas no SQL Server, isso é determinado pela configuração de agrupamento da instância do SQL Server. Por padrão, essa configuração diferencia maiúsculas de minúsculas. Você pode chamar o método `ToUpper` para fazer com que o teste diferencie maiúsculas de minúsculas de forma explícita: `Where(s => s.LastName.ToUpper().Contains(searchString.ToUpper()))`. Isso garantirá que os resultados permaneçam os mesmos se você alterar o código mais tarde para usar um repositório que retorna uma coleção `IEnumerable` em vez de um objeto `IQueryable`. (Quando você chama o método `Contains` em uma coleção `IEnumerable`, obtém a implementação do .NET Framework; quando chama-o em um objeto `IQueryable`, obtém a implementação do provedor de banco de dados.) No entanto, há uma penalidade de desempenho para essa solução. O código `ToUpper` colocará uma função na cláusula WHERE da instrução TSQL SELECT. Isso pode impedir que o otimizador use um índice. Considerando que o SQL geralmente é instalado como não diferenciando maiúsculas e minúsculas, é melhor evitar o código `ToUpper` até você migrar para um armazenamento de dados que diferencia maiúsculas de minúsculas.

### Adicionar uma Caixa de Pesquisa à exibição Índice de Alunos

Em `Views/Student/Index.cshtml`, adicione o código realçado imediatamente antes da marcação de tabela de abertura para criar uma legenda, uma caixa de texto e um botão **Pesquisar**.

```
<p>
    <a href="#" asp-action="Create">Create New</a>
</p>

<form asp-action="Index" method="get">
    <div class="form-actions no-color">
        <p>
            Find by name: <input type="text" name="SearchString" value="@ViewData["currentFilter"]" />
            <input type="submit" value="Search" class="btn btn-default" /> |
            <a href="#" asp-action="Index">Back to Full List</a>
        </p>
    </div>
</form>

<table class="table">
```

Esse código usa o auxiliar de marcação `<form>` para adicionar o botão e a caixa de texto de pesquisa. Por padrão, o auxiliar de marcação `<form>` envia dados de formulário com um POST, o que significa que os parâmetros são passados no corpo da mensagem HTTP e não na URL como cadeias de consulta. Quando você especifica HTTP GET, os dados de formulário são passados na URL como cadeias de consulta, o que permite aos usuários marcar a URL. As diretrizes do W3C recomendam o uso de GET quando a ação não resulta em uma atualização.

Execute o aplicativo, selecione a guia **Alunos**, insira uma cadeia de caracteres de pesquisa e clique em Pesquisar para verificar se a filtragem está funcionando.

Last Name	First Mid Name	Enrollment Date	
Alexander	Carson	9/2/2005 12:00:00 AM	Edit   Details   Delete
Anand	Arturo	9/1/2003 12:00:00 AM	Edit   Details   Delete
Li	Yan	9/1/2002 12:00:00 AM	Edit   Details   Delete
Norman	Laura	9/1/2003 12:00:00 AM	Edit   Details   Delete

Observe que a URL contém a cadeia de caracteres de pesquisa.

```
http://localhost:5813/Students?SearchString=an
```

Se você marcar essa página, obterá a lista filtrada quando usar o indicador. A adição de `method="get"` à marcação `form` é o que fez com que a cadeia de caracteres de consulta fosse gerada.

Neste estágio, se você clicar em um link de classificação de título de coluna perderá o valor de filtro inserido na caixa **Pesquisa**. Você corrigirá isso na próxima seção.

## Adicionar a funcionalidade de paginação à página Índice de Alunos

Para adicionar a paginação à página Índice de alunos, você criará uma classe `PaginatedList` que usa as instruções `Skip` e `Take` para filtrar os dados no servidor, em vez de recuperar sempre todas as linhas da tabela. Em seguida, você fará outras alterações no método `Index` e adicionará botões de paginação à exibição `Index`. A ilustração a seguir mostra os botões de paginação.

The screenshot shows a web browser window with the title "Index - Contoso Univers" and the URL "localhost:5813/Student". The page is titled "Contoso University" and features a heading "Index". A "Create New" link is visible. Below it is a search bar with a "Search" button and a link to "Back to Full List". The main content is a table with three columns: "Last Name", "First Name", and "Enrollment Date". The data rows are:

Last Name	First Name	Enrollment Date
Alexander	Carson	9/1/2005 12:00:00 AM
Alonso	Meredith	9/1/2002 12:00:00 AM
Anand	Arturo	9/1/2003 12:00:00 AM

Each row has "Edit | Details | Delete" links. At the bottom are "Previous" and "Next" navigation buttons.

Na pasta do projeto, crie `PaginatedList.cs` e, em seguida, substitua o código de modelo pelo código a seguir.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;

namespace ContosoUniversity
{
    public class PaginatedList<T> : List<T>
    {
        public int PageIndex { get; private set; }
        public int TotalPages { get; private set; }

        public PaginatedList(List<T> items, int count, int pageIndex, int pageSize)
        {
            PageIndex = pageIndex;
            TotalPages = (int)Math.Ceiling(count / (double)pageSize);

            this.AddRange(items);
        }

        public bool HasPreviousPage
        {
            get
            {
                return (PageIndex > 1);
            }
        }

        public bool HasNextPage
        {
            get
            {
                return (PageIndex < TotalPages);
            }
        }

        public static async Task<PaginatedList<T>> CreateAsync(IQueryable<T> source, int pageIndex, int pageSize)
        {
            var count = await source.CountAsync();
            var items = await source.Skip((pageIndex - 1) * pageSize).Take(pageSize).ToListAsync();
            return new PaginatedList<T>(items, count, pageIndex, pageSize);
        }
    }
}

```

O método `CreateAsync` nesse código usa o tamanho da página e o número da página e aplica as instruções `Skip` e `Take` ao `IQueryable`. Quando `ToListAsync` for chamado no `IQueryable`, ele retornará uma Lista que contém somente a página solicitada. As propriedades `HasPreviousPage` e `HasNextPage` podem ser usadas para habilitar ou desabilitar os botões de paginação **Anterior** e **Próximo**.

Um método `CreateAsync` é usado em vez de um construtor para criar o objeto `PaginatedList<T>`, porque os construtores não podem executar um código assíncrono.

## Adicionar a funcionalidade de paginação ao método Index

Em `StudentsController.cs`, substitua o método `Index` pelo código a seguir.

```

public async Task<IActionResult> Index(
    string sortOrder,
    string currentFilter,
    string searchString,
    int? page)
{
    ViewData["CurrentSort"] = sortOrder;
    ViewData["NameSortParm"] = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
    ViewData["DateSortParm"] = sortOrder == "Date" ? "date_desc" : "Date";

    if (searchString != null)
    {
        page = 1;
    }
    else
    {
        searchString = currentFilter;
    }

    ViewData["CurrentFilter"] = searchString;

    var students = from s in _context.Students
                  select s;
    if (!String.IsNullOrEmpty(searchString))
    {
        students = students.Where(s => s.LastName.Contains(searchString)
                           || s.FirstMidName.Contains(searchString));
    }
    switch (sortOrder)
    {
        case "name_desc":
            students = students.OrderByDescending(s => s.LastName);
            break;
        case "Date":
            students = students.OrderBy(s => s.EnrollmentDate);
            break;
        case "date_desc":
            students = students.OrderByDescending(s => s.EnrollmentDate);
            break;
        default:
            students = students.OrderBy(s => s.LastName);
            break;
    }

    int pageSize = 3;
    return View(await PaginatedList<Student>.CreateAsync(students.AsNoTracking(), page ?? 1, pageSize));
}

```

Esse código adiciona um parâmetro de número de página, um parâmetro de ordem de classificação atual e um parâmetro de filtro atual à assinatura do método.

```

public async Task<IActionResult> Index(
    string sortOrder,
    string currentFilter,
    string searchString,
    int? page)

```

Na primeira vez que a página for exibida, ou se o usuário ainda não tiver clicado em um link de paginação ou classificação, todos os parâmetros serão nulos. Se um link de paginação receber um clique, a variável de página conterá o número da página a ser exibido.

O elemento `ViewData` chamado `CurrentSort` fornece à exibição a ordem de classificação atual, pois isso precisa ser incluído nos links de paginação para manter a ordem de classificação igual durante a paginação.

O elemento `ViewData` chamado `CurrentFilter` fornece à exibição a cadeia de caracteres de filtro atual. Esse valor precisa ser incluído nos links de paginação para manter as configurações de filtro durante a paginação e precisa ser restaurado para a caixa de texto quando a página é exibida novamente.

Se a cadeia de caracteres de pesquisa for alterada durante a paginação, a página precisará ser redefinida como 1, porque o novo filtro pode resultar na exibição de dados diferentes. A cadeia de caracteres de pesquisa é alterada quando um valor é inserido na caixa de texto e o botão Enviar é pressionado. Nesse caso, o parâmetro `searchString` não é nulo.

```
if (searchString != null)
{
    page = 1;
}
else
{
    searchString = currentFilter;
}
```

Ao final do método `Index`, o método `PaginatedList.CreateAsync` converte a consulta de alunos em uma única página de alunos de um tipo de coleção compatível com paginação. A única página de alunos é então passada para a exibição.

```
return View(await PaginatedList<Student>.CreateAsync(students.AsNoTracking(), page ?? 1, pageSize));
```

O método `PaginatedList.CreateAsync` usa um número de página. Os dois pontos de interrogação representam o operador de união de nulo. O operador de união de nulo define um valor padrão para um tipo que permite valor nulo; a expressão `(page ?? 1)` significa retornar o valor de `page` se ele tiver um valor ou retornar 1 se `page` for nulo.

## Adicionar links de paginação à exibição Índice de Alunos

Em `Views/Students/Index.cshtml`, substitua o código existente pelo código a seguir. As alterações são realçadas.

```
@model PaginatedList<ContosoUniversity.Models.Student>

 @{
     ViewData["Title"] = "Index";
 }

<h2>Index</h2>

<p>
    <a asp-action="Create">Create New</a>
</p>

<form asp-action="Index" method="get">
    <div class="form-actions no-color">
        <p>
            Find by name: <input type="text" name="SearchString" value="@ViewData["currentFilter"]" />
            <input type="submit" value="Search" class="btn btn-default" /> |
            <a asp-action="Index">Back to Full List</a>
        </p>
    </div>
</form>

<table class="table">
    <thead>
        <tr>
            <th>
                <a asp-action="Index" asp-route-sortOrder="@ViewData["NameSortParm"]" asp-route-
```

```

        currentFilter="@ ViewData["CurrentFilter"]">Last Name</a>
    </th>
    <th>
        First Name
    </th>
    <th>
        <a asp-action="Index" asp-route-sortOrder="@ ViewData["DateSortParm]" asp-route-
currentFilter="@ ViewData["CurrentFilter"]">Enrollment Date</a>
    </th>
    <th></th>
</tr>
</thead>
<tbody>
    @foreach (var item in Model)
    {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.LastName)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.FirstMidName)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.EnrollmentDate)
            </td>
            <td>
                <a asp-action="Edit" asp-route-id="@item.ID">Edit</a> | 
                <a asp-action="Details" asp-route-id="@item.ID">Details</a> | 
                <a asp-action="Delete" asp-route-id="@item.ID">Delete</a>
            </td>
        </tr>
    }
</tbody>
</table>

@{
    var prevDisabled = !Model.HasPreviousPage ? "disabled" : "";
    var nextDisabled = !Model.HasNextPage ? "disabled" : "";
}

<a asp-action="Index"
    asp-route-sortOrder="@ ViewData["CurrentSort"]"
    asp-route-page="@ (ModelPageIndex - 1)"
    asp-route-currentFilter="@ ViewData["CurrentFilter"]"
    class="btn btn-default @prevDisabled">
    Previous
</a>
<a asp-action="Index"
    asp-route-sortOrder="@ ViewData["CurrentSort"]"
    asp-route-page="@ (ModelPageIndex + 1)"
    asp-route-currentFilter="@ ViewData["CurrentFilter"]"
    class="btn btn-default @nextDisabled">
    Next
</a>

```

A instrução `@model` na parte superior da página especifica que a exibição agora obtém um objeto `PaginatedList<T>`, em vez de um objeto `List<T>`.

Os links de cabeçalho de coluna usam a cadeia de caracteres de consulta para passar a cadeia de caracteres de pesquisa atual para o controlador, de modo que o usuário possa classificar nos resultados do filtro:

```

<a asp-action="Index" asp-route-sortOrder="@ ViewData["DateSortParm]" asp-route-currentFilter
    ="@ ViewData["CurrentFilter"]">Enrollment Date</a>

```

Os botões de paginação são exibidos por auxiliares de marcação:

```

<a asp-action="Index"
    asp-route-sortOrder="@ViewData["CurrentSort"]"
    asp-route-page="@{ModelPageIndex - 1}"
    asp-route-currentFilter="@ViewData["CurrentFilter"]"
    class="btn btn-default @prevDisabled">
    Previous
</a>

```

Execute o aplicativo e acesse a página Alunos.

Last Name	First Name	Enrollment Date	
Alexander	Carson	9/1/2005 12:00:00 AM	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Alonso	Meredith	9/1/2002 12:00:00 AM	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Anand	Arturo	9/1/2003 12:00:00 AM	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

Clique nos links de paginação em ordens de classificação diferentes para verificar se a paginação funciona. Em seguida, insira uma cadeia de caracteres de pesquisa e tente fazer a paginação novamente para verificar se ela também funciona corretamente com a classificação e filtragem.

## Criar uma página Sobre que mostra as estatísticas de Alunos

Para a página **Sobre** do site da Contoso University, você exibirá quantos alunos se registraram para cada data de registro. Isso exige agrupamento e cálculos simples nos grupos. Para fazer isso, você fará o seguinte:

- Criar uma classe de modelo de exibição para os dados que você precisa passar para a exibição.
- Modificar o método About no controlador Home.
- Modificar a exibição Sobre.

### Criar o modelo de exibição

Crie uma pasta *SchoolViewModels* na pasta *Models*.

Na nova pasta, adicione um arquivo de classe *EnrollmentDateGroup.cs* e substitua o código de modelo pelo seguinte código:

```
using System;
using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models.SchoolViewModels
{
    public class EnrollmentDateGroup
    {
        [DataType(DataType.Date)]
        public DateTime? EnrollmentDate { get; set; }

        public int StudentCount { get; set; }
    }
}
```

## Modificar o controlador Home

Em `HomeController.cs`, adicione o seguinte usando as instruções na parte superior do arquivo:

```
using Microsoft.EntityFrameworkCore;
using ContosoUniversity.Data;
using ContosoUniversity.Models.SchoolViewModels;
```

Adicione uma variável de classe ao contexto de banco de dados imediatamente após a chave de abertura da classe e obtenha uma instância do contexto da DI do ASP.NET Core:

```
public class HomeController : Controller
{
    private readonly SchoolContext _context;

    public HomeController(SchoolContext context)
    {
        _context = context;
    }
}
```

Substitua o método `About` pelo seguinte código:

```
public async Task<ActionResult> About()
{
    IQueryable<EnrollmentDateGroup> data =
        from student in _context.Students
        group student by student.EnrollmentDate into dateGroup
        select new EnrollmentDateGroup()
    {
        EnrollmentDate = dateGroup.Key,
        StudentCount = dateGroup.Count()
    };
    return View(await data.AsNoTracking().ToListAsync());
}
```

A instrução LINQ agrupa as entidades de alunos por data de registro, calcula o número de entidades em cada grupo e armazena os resultados em uma coleção de objetos de modelo de exibição `EnrollmentDateGroup`.

### NOTE

Na versão 1.0 do Entity Framework Core, todo o conjunto de resultados é retornado para o cliente e o agrupamento é feito no cliente. Em alguns cenários, isso pode criar problemas de desempenho. Teste o desempenho com volumes de dados de produção e, se necessário, use o SQL bruto para fazer o agrupamento no servidor. Para obter informações sobre como usar o SQL bruto, veja [o último tutorial desta série](#).

## Modificar a exibição Sobre

Substitua o código no arquivo *Views/Home/About.cshtml* pelo seguinte código:

```
@model IEnumerable<ContosoUniversity.Models.SchoolViewModels.EnrollmentDateGroup>

@{
    ViewData["Title"] = "Student Body Statistics";
}



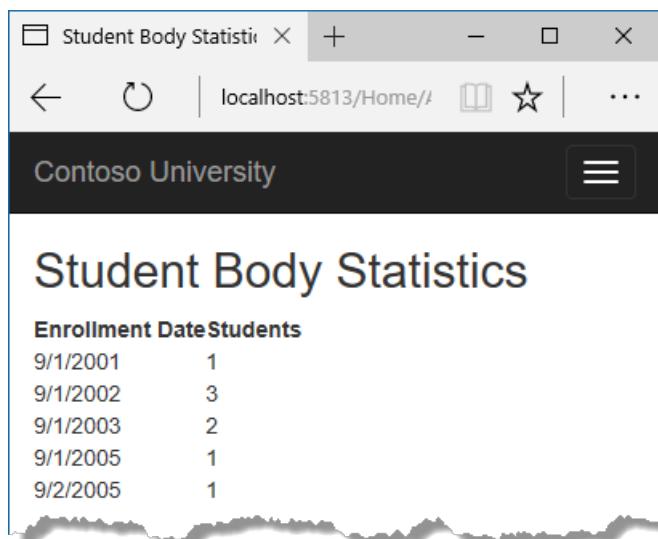
## Student Body Statistics



| Enrollment Date | Students |
|-----------------|----------|
|-----------------|----------|


```

Execute o aplicativo e acesse a página Sobre. A contagem de alunos para cada data de registro é exibida em uma tabela.



## Resumo

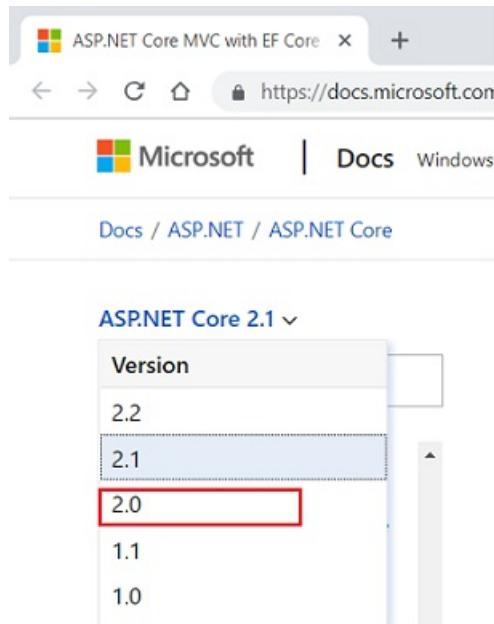
Neste tutorial, você viu como realizar classificação, filtragem, paginação e agrupamento. No próximo tutorial, você aprenderá a manipular as alterações do modelo de dados usando migrações.



# ASP.NET Core MVC com EF Core – migrações – 4 de 10

01/11/2018 • 15 minutes to read • [Edit Online](#)

Este tutorial não foi atualizado para o ASP.NET Core 2.1. A versão deste tutorial para o ASP.NET Core 2.0 pode ser consultada selecionando **ASP.NET Core 2.0** acima do sumário ou na parte superior da página:



The screenshot shows a browser window with the URL <https://docs.microsoft.com>. The page title is "ASP.NET Core MVC with EF Core". Below the title, there's a Microsoft logo and navigation links for "Docs" and "Windows". Under "Docs", the path "Docs / ASP.NET / ASP.NET Core" is shown. A dropdown menu titled "ASP.NET Core 2.1" is open, listing versions: 2.2, 2.1, 2.0, 1.1, and 1.0. The version "2.0" is highlighted with a red border.

A versão deste tutorial do [Razor Pages](#) para o ASP.NET Core 2.1 conta com várias melhorias em relação à versão 2.0.

O tutorial do 2.0 ensina a usar o ASP.NET Core MVC e o Entity Framework Core com controladores e exibições. O tutorial do Razor Pages foi atualizado com os seguintes aprimoramentos:

- É mais fácil de acompanhar. Por exemplo, o código de scaffolding foi bastante simplificado.
- Fornece mais práticas recomendadas do EF Core.
- Usa consultas mais eficientes.
- Usa a API do EF Core mais recente.

Por [Tom Dykstra](#) e [Rick Anderson](#)

O aplicativo web de exemplo Contoso University demonstra como criar aplicativos web do ASP.NET Core MVC usando o Entity Framework Core e o Visual Studio. Para obter informações sobre a série de tutoriais, consulte [o primeiro tutorial da série](#).

Neste tutorial, você começa usando o recurso de migrações do EF Core para o gerenciamento de alterações do modelo de dados. Em tutoriais seguintes, você adicionará mais migrações conforme você alterar o modelo de dados.

## Introdução às migrações

Quando você desenvolve um novo aplicativo, o modelo de dados é alterado com frequência e, sempre que o modelo é alterado, ele fica fora de sincronia com o banco de dados. Você começou estes tutoriais configurando o Entity Framework para criar o banco de dados, caso ele não exista. Em seguida, sempre que você alterar o modelo de dados – adicionar, remover, alterar classes de entidade ou alterar a classe `DbContext` –, poderá excluir

o banco de dados e o EF criará um novo que corresponde ao modelo e o propagará com os dados de teste.

Esse método de manter o banco de dados em sincronia com o modelo de dados funciona bem até que você implante o aplicativo em produção. Quando o aplicativo é executado em produção, ele normalmente armazena os dados que você deseja manter, e você não quer perder tudo sempre que fizer uma alteração, como a adição de uma nova coluna. O recurso Migrações do EF Core resolve esse problema, permitindo que o EF atualize o esquema de banco de dados em vez de criar um novo banco de dados.

## Pacotes NuGet do Entity Framework Core para migrações

Para trabalhar com migrações, use o **PMC** (Console do Gerenciador de Pacotes) ou a CLI (interface de linha de comando). Esses tutoriais mostram como usar comandos da CLI. Encontre informações sobre o PMC no [final deste tutorial](#).

As ferramentas do EF para a CLI (interface de linha de comando) são fornecidas em [Microsoft.EntityFrameworkCore.Tools.DotNet](#). Para instalar esse pacote, adicione-o à coleção `DotNetCliToolReference` no arquivo `.csproj`, conforme mostrado. **Observação:** é necessário instalar este pacote editando o arquivo `.csproj`; não é possível usar o comando `install-package` ou a GUI do Gerenciador de Pacotes. Edite o arquivo `.csproj` clicando com o botão direito do mouse no nome do projeto no **Gerenciador de Soluções** e selecionando **Editar ContosoUniversity.csproj**.

```
<ItemGroup>
  <DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet" Version="2.0.0" />
  <DotNetCliToolReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Tools" Version="2.0.0" />
</ItemGroup>
```

(Neste exemplo, os números de versão eram atuais no momento em que o tutorial foi escrito.)

## Alterar a cadeia de conexão

No arquivo `appsettings.json`, altere o nome do banco de dados na cadeia de conexão para `ContosoUniversity2` ou outro nome que você ainda não usou no computador que está sendo usado.

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=
(localdb)\.\mssqllocaldb;Database=ContosoUniversity2;Trusted_Connection=True;MultipleActiveResultSets=true"
  },
}
```

Essa alteração configura o projeto, de modo que a primeira migração crie um novo banco de dados. Isso não é necessário para começar as migrações, mas você verá mais tarde o motivo pelo qual essa é uma boa ideia.

### NOTE

Como alternativa à alteração do nome do banco de dados, você pode excluir o banco de dados. Use o **SSOX** (Pesquisador de Objetos do SQL Server) ou o comando `database drop` da CLI:

```
dotnet ef database drop
```

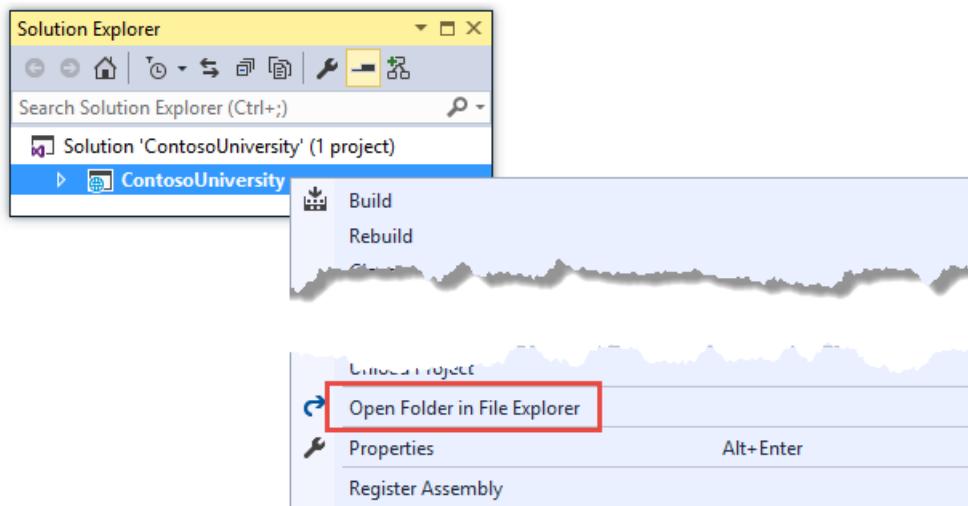
A seção a seguir explica como executar comandos da CLI.

## Criar uma migração inicial

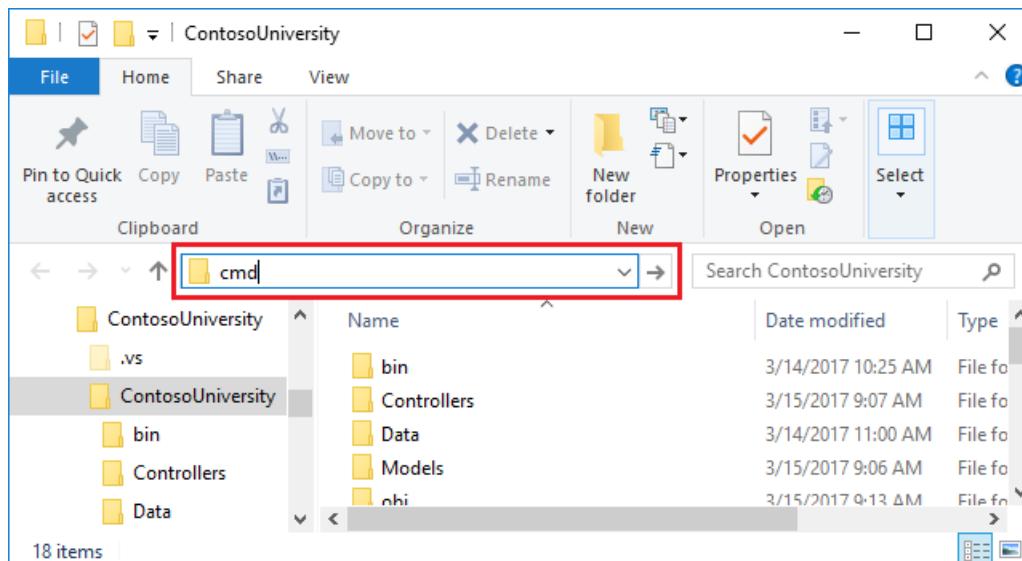
Salve as alterações e compile o projeto. Em seguida, abra uma janela Comando e navegue para a pasta do

projeto. Esta é uma maneira rápida de fazer isso:

- No **Gerenciador de Soluções**, clique com o botão direito do mouse no projeto e escolha **Abrir no Explorador de Arquivos** no menu de contexto.



- Insira "cmd" na barra de endereços e pressione Enter.



Insira o seguinte comando na janela de comando:

```
dotnet ef migrations add InitialCreate
```

Você verá uma saída semelhante à seguinte na janela Comando:

```
info: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[0]
      User profile is available. Using 'C:\Users\username\AppData\Local\ASP.NET\DataProtection-Keys' as key
      repository and Windows DPAPI to encrypt keys at rest.
info: Microsoft.EntityFrameworkCore.Infrastructure[100403]
      Entity Framework Core 2.0.0-rtm-26452 initialized 'SchoolContext' using provider
      'Microsoft.EntityFrameworkCore.SqlServer' with options: None
      Done. To undo this action, use 'ef migrations remove'
```

#### NOTE

Se você receber uma mensagem de erro *Nenhum comando "dotnet-ef" executável correspondente encontrado*, consulte [esta postagem no blog](#) para ajudar a solucionar o problema.

Se você receber uma mensagem de erro "*Não é possível acessar o arquivo... ContosoUniversity.dll porque ele está sendo usado por outro processo*", localize o ícone do IIS Express na Bandeja do Sistema do Windows, clique com o botão direito do mouse nele e, em seguida, clique em **ContosoUniversity > Parar Site**.

## Examinar os métodos Up e Down

Quando você executou o comando `migrations add`, o EF gerou o código que criará o banco de dados do zero. Esse código está localizado na pasta *Migrations*, no arquivo chamado `<timestamp>_InitialCreate.cs`. O método `Up` da classe `InitialCreate` cria as tabelas de banco de dados que correspondem aos conjuntos de entidades do modelo de dados, e o método `Down` exclui-as, conforme mostrado no exemplo a seguir.

```
public partial class InitialCreate : Migration
{
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "Course",
            columns: table => new
            {
                CourseID = table.Column<int>(nullable: false),
                Credits = table.Column<int>(nullable: false),
                Title = table.Column<string>(nullable: true)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_Course", x => x.CourseID);
            });
        // Additional code not shown
    }

    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropTable(
            name: "Enrollment");
        // Additional code not shown
    }
}
```

As migrações chamam o método `Up` para implementar as alterações do modelo de dados para uma migração. Quando você insere um comando para reverter a atualização, as Migrações chamam o método `Down`.

Esse código destina-se à migração inicial que foi criada quando você inseriu o comando `migrations add InitialCreate`. O parâmetro de nome da migração ("InitialCreate" no exemplo) é usado para o nome do arquivo e pode ser o que você desejar. É melhor escolher uma palavra ou frase que resume o que está sendo feito na migração. Por exemplo, você pode nomear uma migração posterior "AddDepartmentTable".

Se você criou a migração inicial quando o banco de dados já existia, o código de criação de banco de dados é gerado, mas ele não precisa ser executado porque o banco de dados já corresponde ao modelo de dados. Quando você implantar o aplicativo em outro ambiente no qual o banco de dados ainda não existe, esse código será executado para criar o banco de dados; portanto, é uma boa ideia testá-lo primeiro. É por isso que você alterou o nome do banco de dados na cadeia de conexão anteriormente – para que as migrações possam criar um novo do zero.

# O instantâneo do modelo de dados

As migrações criam um *instantâneo* do esquema de banco de dados atual em `Migrations/SchoolContextModelSnapshot.cs`. Quando você adiciona uma migração, o EF determina o que foi alterado, comparando o modelo de dados com o arquivo de instantâneo.

Ao excluir uma migração, use o comando `dotnet ef migrations remove`. `dotnet ef migrations remove` exclui a migração e garante que o instantâneo seja redefinido corretamente.

Confira [Migrações do EF Core em ambientes de equipe](#) para obter mais informações de como o arquivo de instantâneo é usado.

## Aplicar a migração ao banco de dados

Na janela Comando, insira o comando a seguir para criar o banco de dados e tabelas nele.

```
dotnet ef database update
```

A saída do comando é semelhante ao comando `migrations add`, exceto que os logs para os comandos SQL que configuraram o banco de dados são exibidos. A maioria dos logs é omitida na seguinte saída de exemplo. Se você preferir não ver esse nível de detalhe em mensagens de log, altere o nível de log no arquivo `appsettings.Development.json`. Para obter mais informações, consulte [Registro em log no ASP.NET Core](#).

```
info: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[0]
      User profile is available. Using 'C:\Users\username\AppData\Local\ASP.NET\DataProtection-Keys' as key
      repository and Windows DPAPI to encrypt keys at rest.
info: Microsoft.EntityFrameworkCore.Infrastructure[100403]
      Entity Framework Core 2.0.0-rtm-26452 initialized 'SchoolContext' using provider
'Microsoft.EntityFrameworkCore.SqlServer' with options: None
info: Microsoft.EntityFrameworkCore.Database.Command[200101]
      Executed DbCommand (467ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
      CREATE DATABASE [ContosoUniversity2];
info: Microsoft.EntityFrameworkCore.Database.Command[200101]
      Executed DbCommand (20ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE TABLE [__EFMigrationsHistory] (
          [MigrationId] nvarchar(150) NOT NULL,
          [ProductVersion] nvarchar(32) NOT NULL,
          CONSTRAINT [PK__EFMigrationsHistory] PRIMARY KEY ([MigrationId])
      );
<logs omitted for brevity>
info: Microsoft.EntityFrameworkCore.Database.Command[200101]
      Executed DbCommand (3ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
      VALUES (N'20170816151242_InitialCreate', N'2.0.0-rtm-26452');
Done.
```

Use o [Pesquisador de Objetos do SQL Server](#) para inspecionar o banco de dados como você fez no primeiro tutorial. Você observará a adição de uma tabela `__EFMigrationsHistory` que controla quais migrações foram aplicadas ao banco de dados. Exiba os dados dessa tabela e você verá uma linha para a primeira migração. (O último log no exemplo de saída da CLI anterior mostra a instrução `INSERT` que cria essa linha.)

Execute o aplicativo para verificar se tudo ainda funciona como antes.

Last Name	First Name	Enrollment Date	
Alexander	Carson	9/1/2005 12:00:00 AM	Edit   Details   Delete
Alonso	Meredith	9/1/2002 12:00:00 AM	Edit   Details   Delete
Anand	Arturo	9/1/2003 12:00:00 AM	Edit   Details   Delete

## CLI (interface de linha de comando) vs. PMC (Console do Gerenciador de Pacotes)

As ferramentas do EF para gerenciamento de migrações estão disponíveis por meio dos comandos da CLI do .NET Core ou de cmdlets do PowerShell na janela **PMC** (Console do Gerenciador de Pacotes) do Visual Studio. Este tutorial mostra como usar a CLI, mas você poderá usar o PMC se preferir.

Os comandos do EF para os comandos do PMC estão no pacote [Microsoft.EntityFrameworkCore.Tools](#). Esse pacote está incluído no [metapacote Microsoft.AspNetCore.App](#), portanto você não precisa adicionar uma referência de pacote se o aplicativo tem uma referência de pacote ao [Microsoft.AspNetCore.App](#).

**Importante:** esse não é o mesmo pacote que é instalado para a CLI com a edição do arquivo `.csproj`. O nome deste termina com `Tools`, ao contrário do nome do pacote da CLI que termina com `Tools.DotNet`.

Para obter mais informações sobre os comandos da CLI, consulte [CLI do .NET Core](#).

Para obter mais informações sobre os comandos do PMC, consulte [Console do Gerenciador de Pacotes \(Visual Studio\)](#).

## Resumo

Neste tutorial, você viu como criar e aplicar sua primeira migração. No próximo tutorial, você começará examinando tópicos mais avançados com a expansão do modelo de dados. Ao longo do processo, você criará e aplicará migrações adicionais.

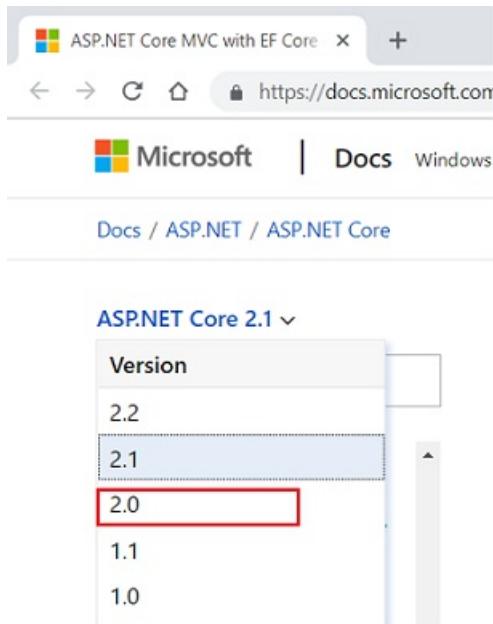
[ANTERIOR](#)

[PRÓXIMO](#)

# ASP.NET Core MVC com EF Core – modelo de dados – 5 de 10

30/10/2018 • 53 minutes to read • [Edit Online](#)

Este tutorial não foi atualizado para o ASP.NET Core 2.1. A versão deste tutorial para o ASP.NET Core 2.0 pode ser consultada selecionando **ASP.NET Core 2.0** acima do sumário ou na parte superior da página:



The screenshot shows a browser window with the URL <https://docs.microsoft.com>. The page title is "ASP.NET Core MVC with EF Core". Below the title, there's a Microsoft logo and navigation links for "Docs" and "Windows". Under "Docs", the path "Docs / ASP.NET / ASP.NET Core" is shown. On the right side, there's a dropdown menu titled "ASP.NET Core 2.1" with a list of versions: 2.2, 2.1, 2.0, 1.1, and 1.0. The version "2.0" is highlighted with a red border.

A versão deste tutorial do [Razor Pages](#) para o ASP.NET Core 2.1 conta com várias melhorias em relação à versão 2.0.

O tutorial do 2.0 ensina a usar o ASP.NET Core MVC e o Entity Framework Core com controladores e exibições. O tutorial do Razor Pages foi atualizado com os seguintes aprimoramentos:

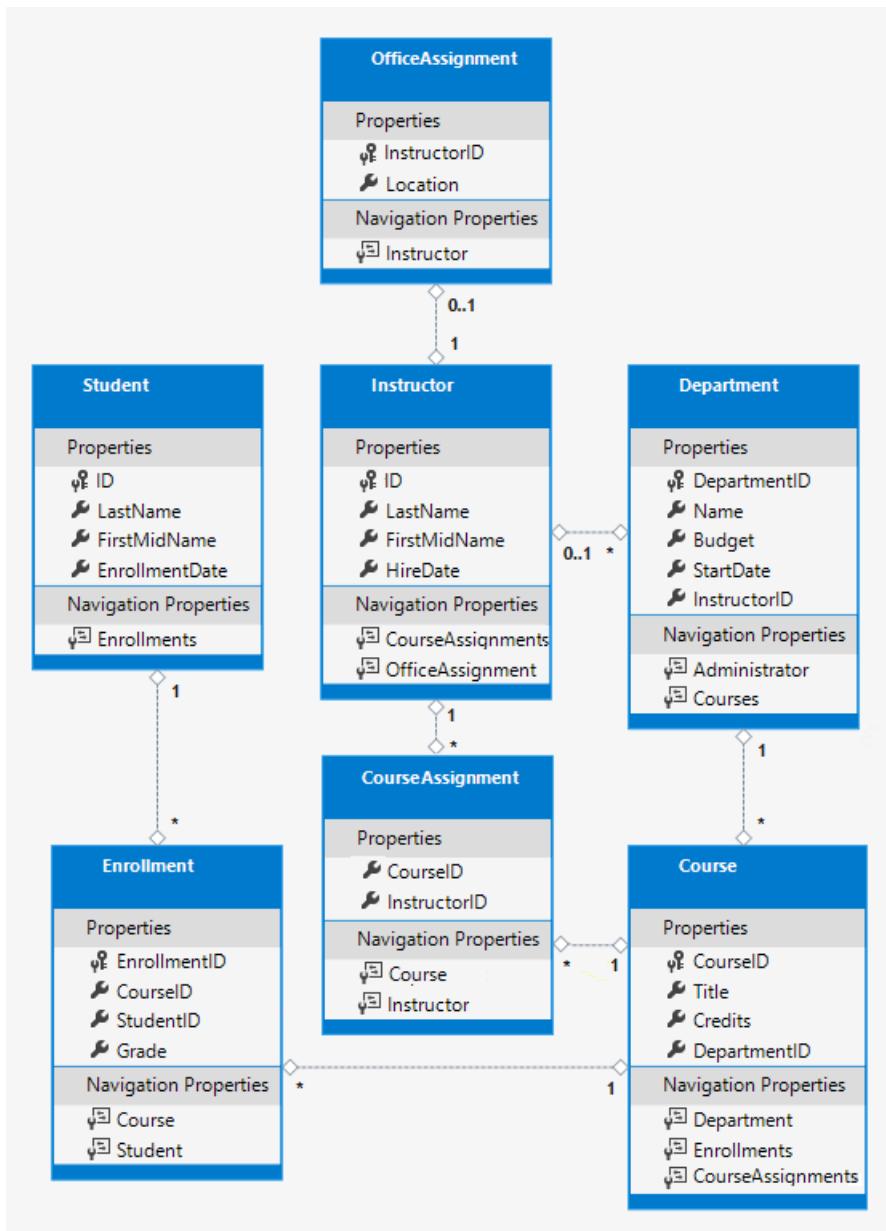
- É mais fácil de acompanhar. Por exemplo, o código de scaffolding foi bastante simplificado.
- Fornece mais práticas recomendadas do EF Core.
- Usa consultas mais eficientes.
- Usa a API do EF Core mais recente.

Por [Tom Dykstra](#) e [Rick Anderson](#)

O aplicativo web de exemplo Contoso University demonstra como criar aplicativos web do ASP.NET Core MVC usando o Entity Framework Core e o Visual Studio. Para obter informações sobre a série de tutoriais, consulte [o primeiro tutorial da série](#).

Nos tutoriais anteriores, você trabalhou com um modelo de dados simples composto por três entidades. Neste tutorial, você adicionará mais entidades e relações e personalizará o modelo de dados especificando formatação, validação e regras de mapeamento de banco de dados.

Quando terminar, as classes de entidade formarão o modelo de dados concluído mostrado na seguinte ilustração:



## Personalizar o modelo de dados usando atributos

Nesta seção, você verá como personalizar o modelo de dados usando atributos que especificam formatação, validação e regras de mapeamento de banco de dados. Em seguida, em várias seções a seguir, você criará o modelo de dados Escola completo com a adição de atributos às classes já criadas e criação de novas classes para os demais tipos de entidade no modelo.

### O atributo `DataType`

Para datas de registro de alunos, todas as páginas da Web atualmente exibem a hora junto com a data, embora tudo o que você deseje exibir nesse campo seja a data. Usando atributos de anotação de dados, você pode fazer uma alteração de código que corrigirá o formato de exibição em cada exibição que mostra os dados. Para ver um exemplo de como fazer isso, você adicionará um atributo à propriedade `EnrollmentDate` na classe `Student`.

Em `Models/Student.cs`, adicione uma instrução `using` ao namespace `System.ComponentModel.DataAnnotations` e adicione os atributos `DataType` e `DisplayFormat` à `EnrollmentDate` propriedade, conforme mostrado no seguinte exemplo:

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models
{
    public class Student
    {
        public int ID { get; set; }
        public string LastName { get; set; }
        public string FirstMidName { get; set; }
        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        public DateTime EnrollmentDate { get; set; }

        public ICollection<Enrollment> Enrollments { get; set; }
    }
}

```

O atributo `DataType` é usado para especificar um tipo de dados mais específico do que o tipo intrínseco de banco de dados. Nesse caso, apenas desejamos acompanhar a data, não a data e a hora. A Enumeração `DataType` fornece muitos tipos de dados, como Date, Time, PhoneNumber, Currency, EmailAddress e muito mais. O atributo `DataType` também pode permitir que o aplicativo forneça automaticamente recursos específicos a um tipo. Por exemplo, um link `mailto:` pode ser criado para `DataType.EmailAddress` e um seletor de data pode ser fornecido para `DataType.Date` em navegadores que dão suporte a HTML5. O atributo `DataType` emite atributos `data-` HTML 5 (pronunciados "data dash") que são reconhecidos pelos navegadores HTML 5. Os atributos `DataType` não fornecem nenhuma validação.

`DataType.Date` não especifica o formato da data exibida. Por padrão, o campo de dados é exibido de acordo com os formatos padrão com base nas `CultureInfo` do servidor.

O atributo `DisplayFormat` é usado para especificar explicitamente o formato de data:

```
[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
```

A configuração `ApplyFormatInEditMode` especifica que a formatação também deve ser aplicada quando o valor é exibido em uma caixa de texto para edição. (Talvez você não deseje ter isso em alguns campos – por exemplo, para valores de moeda, provavelmente, você não deseja exibir o símbolo de moeda na caixa de texto para edição.)

Use Você pode usar o atributo `DisplayFormat` por si só, mas geralmente é uma boa ideia usar o atributo `DataType` também. O atributo `DataType` transmite a semântica dos dados, ao invés de apresentar como renderizá-lo em uma tela, e oferece os seguintes benefícios que você não obtém com `DisplayFormat`:

- O navegador pode habilitar os recursos do HTML5 (por exemplo, mostrar um controle de calendário, o símbolo de moeda apropriado à localidade, links de email, uma validação de entrada do lado do cliente, etc.).
- Por padrão, o navegador renderizará os dados usando o formato correto de acordo com a localidade.

Para obter mais informações, consulte a [documentação do auxiliar de marcação <input>](#).

Execute o aplicativo, acesse a página Índice de Alunos e observe que as horas não são mais exibidas nas datas de registro. O mesmo será verdadeiro para qualquer exibição que usa o modelo Aluno.

Last Name	First Name	Enrollment Date	
Alexander	Carson	2005-09-01	Edit   Details   Delete
Alonso	Meredith	2002-09-01	Edit   Details   Delete
Anand	Arturo	2003-09-01	Edit   Details   Delete

## O atributo StringLength

Você também pode especificar regras de validação de dados e mensagens de erro de validação usando atributos. O atributo `StringLength` define o tamanho máximo do banco de dados e fornece validação do lado do cliente e do lado do servidor para o ASP.NET Core MVC. Você também pode especificar o tamanho mínimo da cadeia de caracteres nesse atributo, mas o valor mínimo não tem nenhum impacto sobre o esquema de banco de dados.

Suponha que você deseja garantir que os usuários não insiram mais de 50 caracteres em um nome. Para adicionar essa limitação, adicione atributos `StringLength` às propriedades `LastName` e `FirstMidName`, conforme mostrado no seguinte exemplo:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models
{
    public class Student
    {
        public int ID { get; set; }
        [StringLength(50)]
        public string LastName { get; set; }
        [StringLength(50, ErrorMessage = "First name cannot be longer than 50 characters.")]  
public string FirstMidName { get; set; }
        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        public DateTime EnrollmentDate { get; set; }

        public ICollection<Enrollment> Enrollments { get; set; }
    }
}
```

O atributo `StringLength` não impedirá que um usuário insira um espaço em branco em um nome. Use o atributo `RegularExpression` para aplicar restrições à entrada. Por exemplo, o seguinte código exige que o primeiro caractere esteja em maiúscula e os caracteres restantes estejam em ordem alfabética:

```
[RegularExpression(@"^([A-Z][a-zA-Z\s-]*$)")]
```

O atributo `MaxLength` fornece uma funcionalidade semelhante ao atributo `StringLength`, mas não fornece a validação do lado do cliente.

Agora, o modelo de banco de dados foi alterado de uma forma que exige uma alteração no esquema de banco de dados. Você usará migrações para atualizar o esquema sem perda dos dados que podem ter sido adicionados ao banco de dados usando a interface do usuário do aplicativo.

Salve as alterações e compile o projeto. Em seguida, abra a janela Comando na pasta do projeto e insira os seguintes comandos:

```
dotnet ef migrations add maxLengthOnNames
```

```
dotnet ef database update
```

O comando `migrations add` alerta que pode ocorrer perda de dados, pois a alteração torna o tamanho máximo mais curto para duas colunas. As migrações criam um arquivo chamado `<timeStamp>_MaxLengthOnNames.cs`. Esse arquivo contém o código no método `Up` que atualizará o banco de dados para que ele corresponda ao modelo de dados atual. O comando `database update` executou esse código.

O carimbo de data/hora prefixado ao nome do arquivo de migrações é usado pelo Entity Framework para ordenar as migrações. Crie várias migrações antes de executar o comando de atualização de banco de dados e, em seguida, todas as migrações são aplicadas na ordem em que foram criadas.

Execute o aplicativo, selecione a guia **Alunos**, clique em **Criar Novo** e insira um nome com mais de 50 caracteres. Quando você clica em **Criar**, a validação do lado do cliente mostra uma mensagem de erro.

Create - Contoso University

localhost:5813/Student

## Create

### Student

**LastName**

Davolio very long last name longer than !

The field LastName must be a string with a maximum length of 50.

**FirstMidName**

Nancy very long first name longer than 5

First name cannot be longer than 50 characters.

**EnrollmentDate**

2/15/2017

Create

## O atributo Column

Você também pode usar atributos para controlar como as classes e propriedades são mapeadas para o banco de dados. Suponha que você tenha usado o nome `FirstMidName` para o campo de nome porque o campo também pode conter um sobrenome. Mas você deseja que a coluna do banco de dados seja nomeada `FirstName`, pois os usuários que escreverão consultas ad hoc no banco de dados estão acostumados com esse nome. Para fazer esse mapeamento, use o atributo `Column`.

O atributo `Column` especifica que quando o banco de dados for criado, a coluna da tabela `Student` que é mapeada para a propriedade `FirstMidName` será nomeada `FirstName`. Em outras palavras, quando o código se referir a `Student.FirstMidName`, os dados serão obtidos ou atualizados na coluna `FirstName` da tabela `Student`. Se você não especificar nomes de coluna, elas receberão o mesmo nome da propriedade.

No arquivo `Student.cs`, adicione uma instrução `using` a `System.ComponentModel.DataAnnotations.Schema` e adicione o atributo de nome de coluna à propriedade `FirstMidName`, conforme mostrado no seguinte código realçado:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Student
    {
        public int ID { get; set; }
        [StringLength(50)]
        public string LastName { get; set; }
        [StringLength(50, ErrorMessage = "First name cannot be longer than 50 characters.")]
        [Column("FirstName")]
        public string FirstMidName { get; set; }
        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        public DateTime EnrollmentDate { get; set; }

        public ICollection<Enrollment> Enrollments { get; set; }
    }
}
```

A adição do atributo `Column` altera o modelo que dá suporte ao `SchoolContext` e, portanto, ele não corresponde ao banco de dados.

Salve as alterações e compile o projeto. Em seguida, abra a janela Comando na pasta do projeto e insira os seguintes comandos para criar outra migração:

```
dotnet ef migrations add ColumnFirstName
```

```
dotnet ef database update
```

No **Pesquisador de Objetos do SQL Server**, abra o designer de tabela Aluno clicando duas vezes na tabela **Aluno**.

The screenshot shows the SQL Server Management Studio (SSMS) interface. In the top left, it says "ContosoUniversity". Below that, "dbo.Students [Design]" is selected. At the top, there's a toolbar with "Update" and "Script File: dbo.Students.sql". The main area shows a table structure with four columns: "Name" (ID), "Data Type" (int, datetime2(7), nvarchar(50), nvarchar(50)), and "Allow Nulls" (checkboxes). To the right, there are sections for "Keys (1)" (PK\_Students), "Check Constraints (0)", "Indexes (0)", "Foreign Keys (0)", and "Triggers (0)". Below the table, there are two tabs: "Design" (selected) and "T-SQL". The "T-SQL" tab contains the following code:

```
1 CREATE TABLE [dbo].[Students] (
2     [ID] INT IDENTITY (1, 1) NOT NULL,
3     [EnrollmentDate] DATETIME2 (7) NOT NULL,
4     [FirstName] NVARCHAR (50) NULL,
5     [LastName] NVARCHAR (50) NULL,
6     CONSTRAINT [PK_Students] PRIMARY KEY CLUSTERED ([ID] ASC)
7 );
```

Antes de você aplicar as duas primeiras migrações, as colunas de nome eram do tipo nvarchar(MAX). Agora elas são nvarchar(50) e o nome da coluna foi alterado de FirstMidName para FirstName.

#### NOTE

Se você tentar compilar antes de concluir a criação de todas as classes de entidade nas seções a seguir, poderá receber erros do compilador.

## Alterações finais na entidade Student

The screenshot shows the EntityDataSource Designer for the "Student" entity. On the left, there's a tree view with "Properties" expanded, showing "ID", "LastName", "FirstMidName", and "EnrollmentDate". Below that is "Navigation Properties", which shows "Enrollments".

Em *Models/Student.cs*, substitua o código que você adicionou anteriormente pelo código a seguir. As alterações são realçadas.

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Student
    {
        public int ID { get; set; }

        [Required]
        [StringLength(50)]
        [Display(Name = "Last Name")]
        public string LastName { get; set; }

        [Required]
        [StringLength(50, ErrorMessage = "First name cannot be longer than 50 characters.")]
        [Column("FirstName")]
        [Display(Name = "First Name")]
        public string FirstMidName { get; set; }

        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        [Display(Name = "Enrollment Date")]
        public DateTime EnrollmentDate { get; set; }

        [Display(Name = "Full Name")]
        public string FullName
        {
            get
            {
                return LastName + ", " + FirstMidName;
            }
        }

        public ICollection<Enrollment> Enrollments { get; set; }
    }
}

```

## O atributo Required

O atributo `Required` torna as propriedades de nome campos obrigatórios. O atributo `Required` não é necessário para tipos que permitem valor nulo como tipos de valor (`DateTime`, `int`, `double`, `float`, etc.). Tipos que não podem ser nulos são tratados automaticamente como campos obrigatórios.

Remova o atributo `Required` e substitua-o por um parâmetro de tamanho mínimo para o atributo `StringLength`:

```

[Display(Name = "Last Name")]
[StringLength(50, MinimumLength=1)]
public string LastName { get; set; }

```

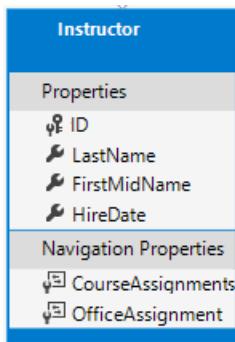
## O atributo Display

O atributo `Display` especifica que a legenda para as caixas de texto deve ser "Nome", "Sobrenome", "Nome Completo" e "Data de Registro", em vez do nome da propriedade em cada instância (que não tem nenhum espaço entre as palavras).

## A propriedade calculada FullName

`FullName` é uma propriedade calculada que retorna um valor criado pela concatenação de duas outras propriedades. Portanto, ela tem apenas um acessador `get` e nenhuma coluna `FullName` será gerada no banco de dados.

## Criar a entidade Instructor



Crie `Models/Instructor.cs`, substituindo o código de modelo com o seguinte código:

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Instructor
    {
        public int ID { get; set; }

        [Required]
        [Display(Name = "Last Name")]
        [StringLength(50)]
        public string LastName { get; set; }

        [Required]
        [Column("FirstName")]
        [Display(Name = "First Name")]
        [StringLength(50)]
        public string FirstMidName { get; set; }

        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        [Display(Name = "Hire Date")]
        public DateTime HireDate { get; set; }

        [Display(Name = "Full Name")]
        public string FullName
        {
            get { return LastName + ", " + FirstMidName; }
        }

        public ICollection<CourseAssignment> CourseAssignments { get; set; }
        public OfficeAssignment OfficeAssignment { get; set; }
    }
}

```

Observe que várias propriedades são as mesmas nas entidades Student e Instructor. No tutorial [Implementando a herança](#) mais adiante nesta série, você refatorará esse código para eliminar a redundância.

Coloque vários atributos em uma linha, de modo que você também possa escrever os atributos `HireDate` da seguinte maneira:

```
[DataType(DataType.Date),Display(Name = "Hire Date"),DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
```

### As propriedades de navegação `CourseAssignments` e `OfficeAssignment`

As propriedades `CourseAssignments` e `OfficeAssignment` são propriedades de navegação.

Um instrutor pode ministrar qualquer quantidade de cursos e, portanto, `CourseAssignments` é definido como uma coleção.

```
public ICollection<CourseAssignment> CourseAssignments { get; set; }
```

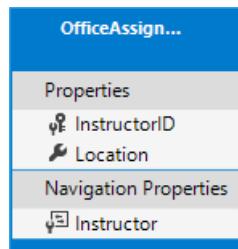
Se uma propriedade de navegação pode conter várias entidades, o tipo precisa ser uma lista na qual as entradas podem ser adicionadas, excluídas e atualizadas. Especifique `ICollection<T>` ou um tipo, como `List<T>` ou `HashSet<T>`. Se você especificar `Icollection<T>`, o EF criará uma coleção `HashSet<T>` por padrão.

O motivo pelo qual elas são entidades `CourseAssignment` é explicado abaixo na seção sobre relações muitos para muitos.

As regras de negócio do Contoso Universidade indicam que um instrutor pode ter apenas, no máximo, um escritório; portanto, a propriedade `OfficeAssignment` contém uma única entidade `OfficeAssignment` (que pode ser nulo se o escritório não está atribuído).

```
public OfficeAssignment OfficeAssignment { get; set; }
```

## Criar a entidade `OfficeAssignment`



Crie `Models/OfficeAssignment.cs` com o seguinte código:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class OfficeAssignment
    {
        [Key]
        public int InstructorID { get; set; }
        [StringLength(50)]
        [Display(Name = "Office Location")]
        public string Location { get; set; }

        public Instructor Instructor { get; set; }
    }
}
```

### O atributo Key

Há uma relação um para zero ou um entre as entidades `Instructor` e `OfficeAssignment`. Uma atribuição de escritório existe apenas em relação ao instrutor ao qual ela é atribuída e, portanto, sua chave primária também é a chave estrangeira da entidade `Instructor`. Mas o Entity Framework não pode reconhecer `InstructorID` automaticamente como a chave primária dessa entidade porque o nome não segue a convenção de nomenclatura ID ou `classnameID`. Portanto, o atributo `Key` é usado para identificá-la como a chave:

```
[Key]  
public int InstructorID { get; set; }
```

Você também pode usar o atributo `Key` se a entidade tem sua própria chave primária, mas você deseja atribuir um nome de propriedade que não seja `classnameID` ou `ID`.

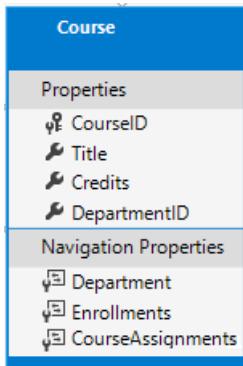
Por padrão, o EF trata a chave como não gerada pelo banco de dados porque a coluna destina-se a uma relação de identificação.

### A propriedade de navegação Instructor

A entidade `Instructor` tem uma propriedade de navegação `OfficeAssignment` que permite valor nulo (porque um instrutor pode não ter uma atribuição de escritório), e a entidade `OfficeAssignment` tem uma propriedade de navegação `Instructor` que não permite valor nulo (porque uma atribuição de escritório não pode existir sem um instrutor – `InstructorID` não permite valor nulo). Quando uma entidade `Instructor` tiver uma entidade `OfficeAssignment` relacionada, cada entidade terá uma referência à outra em sua propriedade de navegação.

Você pode colocar um atributo `[Required]` na propriedade de navegação `Instructor` para especificar que deve haver um instrutor relacionado, mas não precisa fazer isso porque a chave estrangeira `InstructorID` (que também é a chave para esta tabela) não permite valor nulo.

## Modificar a entidade Course



Em `Models/Course.cs`, substitua o código que você adicionou anteriormente pelo código a seguir. As alterações são realçadas.

```

using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Course
    {
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        [Display(Name = "Number")]
        public int CourseID { get; set; }

        [StringLength(50, MinimumLength = 3)]
        public string Title { get; set; }

        [Range(0, 5)]
        public int Credits { get; set; }

        public int DepartmentID { get; set; }

        public Department Department { get; set; }
        public ICollection<Enrollment> Enrollments { get; set; }
        public ICollection<CourseAssignment> CourseAssignments { get; set; }
    }
}

```

A entidade de curso tem uma propriedade de chave estrangeira `DepartmentID` que aponta para a entidade `Department` relacionada e ela tem uma propriedade de navegação `Department`.

O Entity Framework não exige que você adicione uma propriedade de chave estrangeira ao modelo de dados quando você tem uma propriedade de navegação para uma entidade relacionada. O EF cria chaves estrangeiras no banco de dados sempre que elas são necessárias e cria automaticamente [propriedades de sombra](#) para elas. No entanto, ter a chave estrangeira no modelo de dados pode tornar as atualizações mais simples e mais eficientes. Por exemplo, quando você busca uma entidade de curso a ser editada, a entidade `Department` é nula se você não carregá-la; portanto, quando você atualiza a entidade de curso, você precisa primeiro buscar a entidade `Department`. Quando a propriedade de chave estrangeira `DepartmentID` está incluída no modelo de dados, você não precisa buscar a entidade `Department` antes da atualização.

### O atributo `DatabaseGenerated`

O atributo `DatabaseGenerated` com o parâmetro `None` na propriedade `CourseID` especifica que os valores de chave primária são fornecidos pelo usuário, em vez de serem gerados pelo banco de dados.

```

[DatabaseGenerated(DatabaseGeneratedOption.None)]
[Display(Name = "Number")]
public int CourseID { get; set; }

```

Por padrão, o Entity Framework pressupõe que os valores de chave primária sejam gerados pelo banco de dados. É isso que você quer na maioria dos cenários. No entanto, para entidades `Course`, você usará um número de curso especificado pelo usuário como uma série 1000 de um departamento, uma série 2000 para outro departamento e assim por diante.

O atributo `DatabaseGenerated` também pode ser usado para gerar valores padrão, como no caso de colunas de banco de dados usadas para registrar a data em que uma linha foi criada ou atualizada. Para obter mais informações, consulte [Propriedades geradas](#).

### Propriedades de navegação e de chave estrangeira

As propriedades de navegação e de chave estrangeira na entidade `Course` refletem as seguintes relações:

Um curso é atribuído a um departamento e, portanto, há uma propriedade de chave estrangeira `DepartmentID` e de navegação `Department` pelas razões mencionadas acima.

```
public int DepartmentID { get; set; }
public Department Department { get; set; }
```

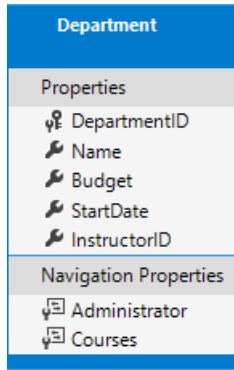
Um curso pode ter qualquer quantidade de estudantes inscritos; portanto, a propriedade de navegação `Enrollments` é uma coleção:

```
public ICollection<Enrollment> Enrollments { get; set; }
```

Um curso pode ser ministrado por vários instrutores; portanto, a propriedade de navegação `CourseAssignments` é uma coleção (o tipo `CourseAssignment` é explicado [posteriormente](#)):

```
public ICollection<CourseAssignment> CourseAssignments { get; set; }
```

## Criar a entidade Department



Crie `Models/Department.cs` com o seguinte código:

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Department
    {
        public int DepartmentID { get; set; }

        [StringLength(50, MinimumLength = 3)]
        public string Name { get; set; }

        [DataType(DataType.Currency)]
        [Column(TypeName = "money")]
        public decimal Budget { get; set; }

        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        [Display(Name = "Start Date")]
        public DateTime StartDate { get; set; }

        public int? InstructorID { get; set; }

        public Instructor Administrator { get; set; }
        public ICollection<Course> Courses { get; set; }
    }
}

```

## O atributo Column

Anteriormente, você usou o atributo `Column` para alterar o mapeamento de nome de coluna. No código da entidade `Department`, o atributo `Column` está sendo usado para alterar o mapeamento de tipo de dados SQL, do modo que a coluna seja definida usando o tipo de dinheiro do SQL Server no banco de dados:

```

[Column(TypeName="money")]
public decimal Budget { get; set; }

```

O mapeamento de coluna geralmente não é necessário, pois o Entity Framework escolhe o tipo de dados do SQL Server apropriado com base no tipo CLR definido para a propriedade. O tipo `decimal` CLR é mapeado para um tipo `decimal` SQL Server. Mas, nesse caso, você sabe que a coluna armazenará os valores de moeda e o tipo de dados dinheiro é mais apropriado para isso.

## Propriedades de navegação e de chave estrangeira

As propriedades de navegação e de chave estrangeira refletem as seguintes relações:

Um departamento pode ou não ter um administrador, e um administrador é sempre um instrutor. Portanto, a propriedade `InstructorID` é incluída como a chave estrangeira na entidade `Instructor` e um ponto de interrogação é adicionado após a designação de tipo `int` para marcar a propriedade como uma propriedade que permite valor nulo. A propriedade de navegação é chamada `Administrator`, mas contém uma entidade `Instructor`:

```

public int? InstructorID { get; set; }
public Instructor Administrator { get; set; }

```

Um departamento pode ter vários cursos e, portanto, há uma propriedade de navegação `Courses`:

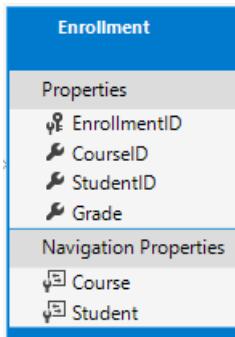
```
public ICollection<Course> Courses { get; set; }
```

#### NOTE

Por convenção, o Entity Framework habilita a exclusão em cascata para chaves estrangeiras que não permitem valor nulo e em relações muitos para muitos. Isso pode resultar em regras de exclusão em cascata circular, que causará uma exceção quando você tentar adicionar uma migração. Por exemplo, se você não definiu a propriedade Department.InstructorID como uma propriedade que permite valor nulo, o EF configura uma regra de exclusão em cascata para excluir o instrutor quando você exclui o departamento, que não é o que você deseja que aconteça. Se as regras de negócios exigissem que a propriedade `InstructorID` não permitisse valor nulo, você precisaria usar a seguinte instrução de API fluente para desabilitar a exclusão em cascata na relação:

```
modelBuilder.Entity<Department>()
    .HasOne(d => d.Administrator)
    .WithMany()
    .OnDelete(DeleteBehavior.Restrict)
```

## Modificar a entidade Enrollment



Em `Models/Enrollment.cs`, substitua o código que você adicionou anteriormente pelo seguinte código:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public enum Grade
    {
        A, B, C, D, F
    }

    public class Enrollment
    {
        public int EnrollmentID { get; set; }
        public int CourseID { get; set; }
        public int StudentID { get; set; }
        [DisplayFormat(NullDisplayText = "No grade")]
        public Grade? Grade { get; set; }

        public Course Course { get; set; }
        public Student Student { get; set; }
    }
}
```

### Propriedades de navegação e de chave estrangeira

As propriedades de navegação e de chave estrangeira refletem as seguintes relações:

Um registro destina-se a um único curso e, portanto, há uma propriedade de chave estrangeira `CourseID` e uma propriedade de navegação `Course`:

```
public int CourseID { get; set; }
public Course Course { get; set; }
```

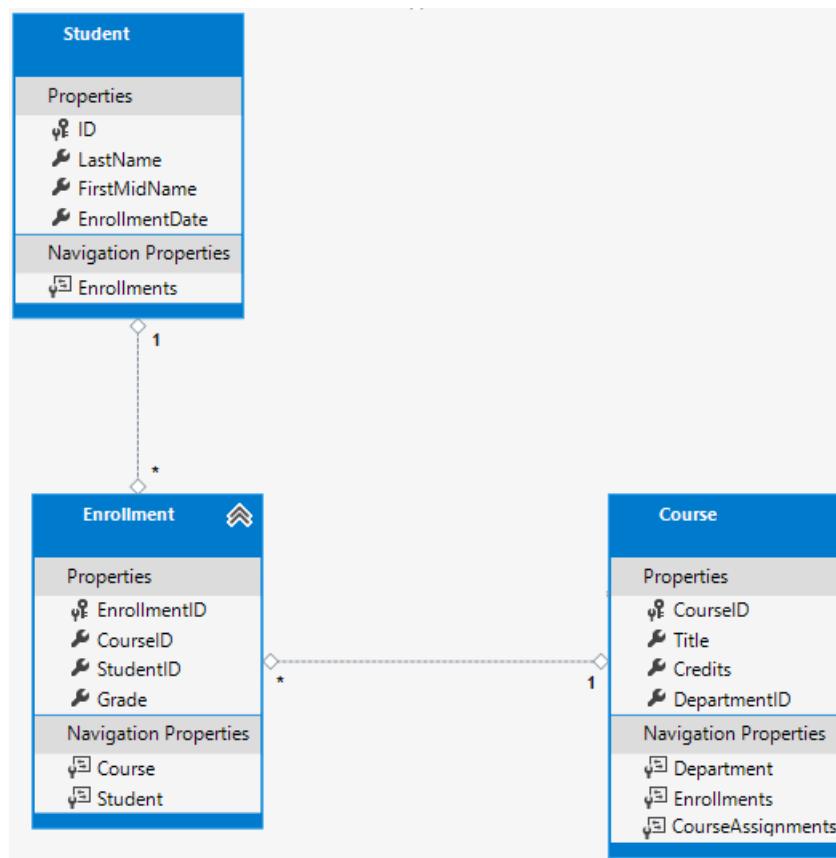
Um registro destina-se a um único aluno e, portanto, há uma propriedade de chave estrangeira `StudentID` e uma propriedade de navegação `Student`:

```
public int StudentID { get; set; }
public Student Student { get; set; }
```

## Relações muitos para muitos

Há uma relação muitos para muitos entre as entidades `Student` e `Course` e a entidade `Enrollment` funciona como uma tabela de junção muitos para muitos *com conteúdo* no banco de dados. "Com conteúdo" significa que a tabela Registro contém dados adicionais além das chaves estrangeiras para as tabelas unidas (nesse caso, uma chave primária e uma propriedade `Grade`).

A ilustração a seguir mostra a aparência dessas relações em um diagrama de entidades. (Esse diagrama foi gerado usando o Entity Framework Power Tools para o EF 6.x; a criação do diagrama não faz parte do tutorial, mas está apenas sendo usada aqui como uma ilustração.)



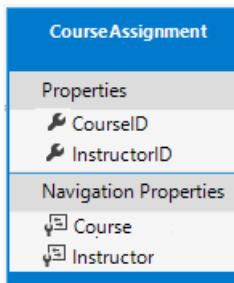
Cada linha de relação tem um 1 em uma extremidade e um asterisco (\*) na outra, indicando uma relação um para muitos.

Se a tabela Registro não incluir informações de nota, ela apenas precisará conter as duas chaves estrangeiras `CourseID` e `StudentID`. Nesse caso, ela será uma tabela de junção de muitos para muitos sem conteúdo (ou uma tabela de junção pura) no banco de dados. As entidades `Instructor` e `Course` têm esse tipo de relação muitos para muitos e a próxima etapa é criar uma classe de entidade para funcionar como uma tabela de junção sem

conteúdo.

(O EF 6.x é compatível com tabelas de junção implícita para relações muitos para muitos, ao contrário do EF Core. Para obter mais informações, confira a [discussão no repositório GitHub do EF Core](#).)

## A entidade CourseAssignment



Crie `Models/CourseAssignment.cs` com o seguinte código:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class CourseAssignment
    {
        public int InstructorID { get; set; }
        public int CourseID { get; set; }
        public Instructor Instructor { get; set; }
        public Course Course { get; set; }
    }
}
```

### Unir nomes de entidade

Uma tabela de junção é necessária no banco de dados para a relação muitos para muitos de Instrutor para Cursos, e ela precisa ser representada por um conjunto de entidades. É comum nomear uma entidade de junção `EntityName1EntityName2`, que, nesse caso, será `CourseInstructor`. No entanto, recomendamos que você escolha um nome que descreve a relação. Modelos de dados começam simples e aumentam, com junções não referentes a conteúdo obtendo com frequência o conteúdo mais tarde. Se você começar com um nome descritivo de entidade, não precisará alterar o nome posteriormente. O ideal é que a entidade de junção tenha seu próprio nome natural (possivelmente, uma única palavra) no domínio de negócios. Por exemplo, Manuais e Clientes podem ser vinculados por meio de Classificações. Para essa relação, `CourseAssignment` é uma escolha melhor que `CourseInstructor`.

### Chave composta

Como as chaves estrangeiras não permitem valor nulo e, juntas, identificam exclusivamente cada linha da tabela, não é necessário ter uma chave primária. As propriedades `InstructorID` e `CourseID` devem funcionar como uma chave primária composta. A única maneira de identificar chaves primárias compostas no EF é usando a *API fluente* (isso não pode ser feito por meio de atributos). Você verá como configurar a chave primária composta na próxima seção.

A chave composta garante que, embora você possa ter várias linhas para um curso e várias linhas para um instrutor, não poderá ter várias linhas para o mesmo instrutor e curso. A entidade de junção `Enrollment` define sua própria chave primária e, portanto, duplicatas desse tipo são possíveis. Para evitar essas duplicatas, você pode adicionar um índice exclusivo nos campos de chave estrangeira ou configurar `Enrollment` com uma chave primária composta semelhante a `courseAssignment`. Para obter mais informações, consulte [Índices](#).

## Atualizar o contexto de banco de dados

Adicione o seguinte código realçado ao arquivo *Data/SchoolContext.cs*:

```
using ContosoUniversity.Models;
using Microsoft.EntityFrameworkCore;

namespace ContosoUniversity.Data
{
    public class SchoolContext : DbContext
    {
        public SchoolContext(DbContextOptions<SchoolContext> options) : base(options)
        {
        }

        public DbSet<Course> Courses { get; set; }
        public DbSet<Enrollment> Enrollments { get; set; }
        public DbSet<Student> Students { get; set; }
        public DbSet<Department> Departments { get; set; }
        public DbSet<Instructor> Instructors { get; set; }
        public DbSet<OfficeAssignment> OfficeAssignments { get; set; }
        public DbSet<CourseAssignment> CourseAssignments { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Course>().ToTable("Course");
            modelBuilder.Entity<Enrollment>().ToTable("Enrollment");
            modelBuilder.Entity<Student>().ToTable("Student");
            modelBuilder.Entity<Department>().ToTable("Department");
            modelBuilder.Entity<Instructor>().ToTable("Instructor");
            modelBuilder.Entity<OfficeAssignment>().ToTable("OfficeAssignment");
            modelBuilder.Entity<CourseAssignment>().ToTable("CourseAssignment");

            modelBuilder.Entity<CourseAssignment>()
                .HasKey(c => new { c.CourseID, c.InstructorID });
        }
    }
}
```

Esse código adiciona novas entidades e configura a chave primária composta da entidade *CourseAssignment*.

## Alternativa de API fluente para atributos

O código no método `OnModelCreating` da classe `DbContext` usa a *API fluente* para configurar o comportamento do EF. A API é chamada "fluente" porque costuma ser usada pelo encadeamento de uma série de chamadas de método em uma única instrução, como neste exemplo da [documentação do EF Core](#):

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .Property(b => b.Url)
        .IsRequired();
}
```

Neste tutorial, você usa a API fluente somente para o mapeamento de banco de dados que não é possível fazer com atributos. No entanto, você também pode usar a API fluente para especificar a maioria das regras de formatação, validação e mapeamento que pode ser feita por meio de atributos. Alguns atributos como `MinimunLength` não podem ser aplicados com a API fluente. Conforme mencionado anteriormente, `MinimunLength` não altera o esquema; apenas aplica uma regra de validação do lado do cliente e do servidor.

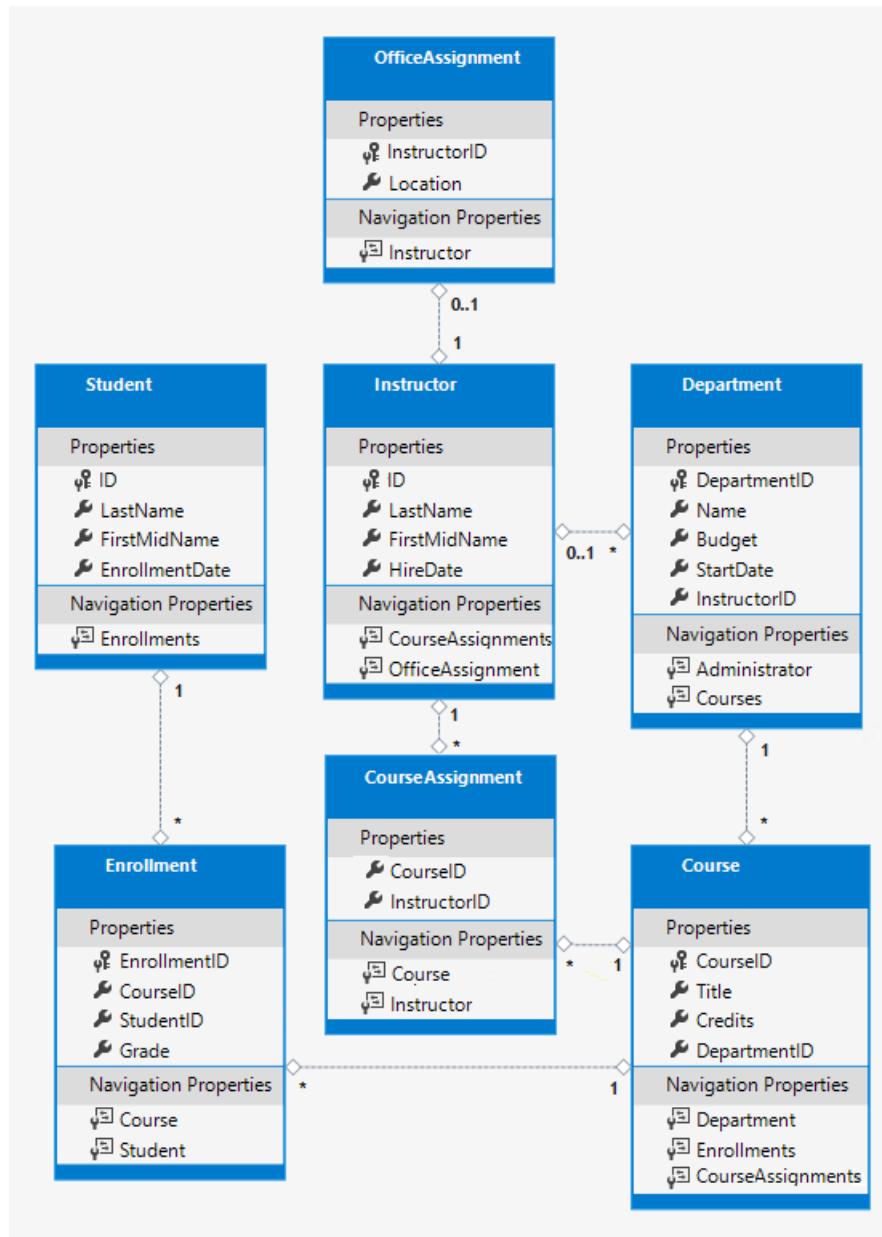
Alguns desenvolvedores preferem usar a API fluente exclusivamente para que possam manter suas classes de

entidade "limpas". Combine atributos e a API fluente se desejar. Além disso, há algumas personalizações que podem ser feitas apenas com a API fluente, mas em geral, a prática recomendada é escolher uma dessas duas abordagens e usar isso com o máximo de consistência possível. Se usar as duas, observe que sempre que houver um conflito, a API fluente substituirá atributos.

Para obter mais informações sobre atributos vs. API fluente, consulte [Métodos de configuração](#).

## Diagrama de entidade mostrando relações

A ilustração a seguir mostra o diagrama criado pelo Entity Framework Power Tools para o modelo Escola concluído.



Além das linhas de relação um-para-muitos (1 para \*), você pode ver a linha de relação um para zero ou um (1 para 0..1) entre as entidades `Instructor` e `OfficeAssignment` e a linha de relação zero-ou-um-para-muitos (0..1 para \*) entre as entidades `Instructor` e `Department`.

## Propagar o banco de dados com os dados de teste

Substitua o código no arquivo `Data/DbInitializer.cs` pelo código a seguir para fornecer dados de semente para as novas entidades criadas.

```
using System;
```

```

using System.Linq;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using ContosoUniversity.Models;

namespace ContosoUniversity.Data
{
    public static class DbInitializer
    {
        public static void Initialize(SchoolContext context)
        {
            //context.Database.EnsureCreated();

            // Look for any students.
            if (context.Students.Any())
            {
                return; // DB has been seeded
            }

            var students = new Student[]
            {
                new Student { FirstMidName = "Carson", LastName = "Alexander",
                    EnrollmentDate = DateTime.Parse("2010-09-01") },
                new Student { FirstMidName = "Meredith", LastName = "Alonso",
                    EnrollmentDate = DateTime.Parse("2012-09-01") },
                new Student { FirstMidName = "Arturo", LastName = "Anand",
                    EnrollmentDate = DateTime.Parse("2013-09-01") },
                new Student { FirstMidName = "Gytis", LastName = "Barzdukas",
                    EnrollmentDate = DateTime.Parse("2012-09-01") },
                new Student { FirstMidName = "Yan", LastName = "Li",
                    EnrollmentDate = DateTime.Parse("2012-09-01") },
                new Student { FirstMidName = "Peggy", LastName = "Justice",
                    EnrollmentDate = DateTime.Parse("2011-09-01") },
                new Student { FirstMidName = "Laura", LastName = "Norman",
                    EnrollmentDate = DateTime.Parse("2013-09-01") },
                new Student { FirstMidName = "Nino", LastName = "Olivetto",
                    EnrollmentDate = DateTime.Parse("2005-09-01") }
            };

            foreach (Student s in students)
            {
                context.Students.Add(s);
            }
            context.SaveChanges();

            var instructors = new Instructor[]
            {
                new Instructor { FirstMidName = "Kim", LastName = "Abercrombie",
                    HireDate = DateTime.Parse("1995-03-11") },
                new Instructor { FirstMidName = "Fadi", LastName = "Fakhouri",
                    HireDate = DateTime.Parse("2002-07-06") },
                new Instructor { FirstMidName = "Roger", LastName = "Harui",
                    HireDate = DateTime.Parse("1998-07-01") },
                new Instructor { FirstMidName = "Candace", LastName = "Kapoor",
                    HireDate = DateTime.Parse("2001-01-15") },
                new Instructor { FirstMidName = "Roger", LastName = "Zheng",
                    HireDate = DateTime.Parse("2004-02-12") }
            };

            foreach (Instructor i in instructors)
            {
                context.Instructors.Add(i);
            }
            context.SaveChanges();

            var departments = new Department[]
            {
                new Department { Name = "English", Budget = 350000,
                    StartDate = DateTime.Parse("2007-09-01"),

```

```

        InstructorID = instructors.Single( i => i.LastName == "Abercrombie").ID },
        new Department { Name = "Mathematics", Budget = 100000,
            StartDate = DateTime.Parse("2007-09-01"),
            InstructorID = instructors.Single( i => i.LastName == "Fakhouri").ID },
        new Department { Name = "Engineering", Budget = 350000,
            StartDate = DateTime.Parse("2007-09-01"),
            InstructorID = instructors.Single( i => i.LastName == "Harui").ID },
        new Department { Name = "Economics", Budget = 100000,
            StartDate = DateTime.Parse("2007-09-01"),
            InstructorID = instructors.Single( i => i.LastName == "Kapoor").ID }
    };

    foreach (Department d in departments)
    {
        context.Departments.Add(d);
    }
    context.SaveChanges();

    var courses = new Course[]
    {
        new Course {CourseID = 1050, Title = "Chemistry", Credits = 3,
            DepartmentID = departments.Single( s => s.Name == "Engineering").DepartmentID },
        new Course {CourseID = 4022, Title = "Microeconomics", Credits = 3,
            DepartmentID = departments.Single( s => s.Name == "Economics").DepartmentID },
        new Course {CourseID = 4041, Title = "Macroeconomics", Credits = 3,
            DepartmentID = departments.Single( s => s.Name == "Economics").DepartmentID },
        new Course {CourseID = 1045, Title = "Calculus", Credits = 4,
            DepartmentID = departments.Single( s => s.Name == "Mathematics").DepartmentID },
        new Course {CourseID = 3141, Title = "Trigonometry", Credits = 4,
            DepartmentID = departments.Single( s => s.Name == "Mathematics").DepartmentID },
        new Course {CourseID = 2021, Title = "Composition", Credits = 3,
            DepartmentID = departments.Single( s => s.Name == "English").DepartmentID },
        new Course {CourseID = 2042, Title = "Literature", Credits = 4,
            DepartmentID = departments.Single( s => s.Name == "English").DepartmentID },
    };

    foreach (Course c in courses)
    {
        context.Courses.Add(c);
    }
    context.SaveChanges();

    var officeAssignments = new OfficeAssignment[]
    {
        new OfficeAssignment {
            InstructorID = instructors.Single( i => i.LastName == "Fakhouri").ID,
            Location = "Smith 17" },
        new OfficeAssignment {
            InstructorID = instructors.Single( i => i.LastName == "Harui").ID,
            Location = "Gowan 27" },
        new OfficeAssignment {
            InstructorID = instructors.Single( i => i.LastName == "Kapoor").ID,
            Location = "Thompson 304" },
    };

    foreach (OfficeAssignment o in officeAssignments)
    {
        context.OfficeAssignments.Add(o);
    }
    context.SaveChanges();
}

```

```

var courseInstructors = new CourseAssignment[]
{
    new CourseAssignment {
        CourseID = courses.Single(c => c.Title == "Chemistry" ).CourseID,
        InstructorID = instructors.Single(i => i.LastName == "Kapoor").ID
    },
    new CourseAssignment {
        CourseID = courses.Single(c => c.Title == "Chemistry" ).CourseID,
        InstructorID = instructors.Single(i => i.LastName == "Harui").ID
    },
    new CourseAssignment {
        CourseID = courses.Single(c => c.Title == "Microeconomics" ).CourseID,
        InstructorID = instructors.Single(i => i.LastName == "Zheng").ID
    },
    new CourseAssignment {
        CourseID = courses.Single(c => c.Title == "Macroeconomics" ).CourseID,
        InstructorID = instructors.Single(i => i.LastName == "Zheng").ID
    },
    new CourseAssignment {
        CourseID = courses.Single(c => c.Title == "Calculus" ).CourseID,
        InstructorID = instructors.Single(i => i.LastName == "Fakhouri").ID
    },
    new CourseAssignment {
        CourseID = courses.Single(c => c.Title == "Trigonometry" ).CourseID,
        InstructorID = instructors.Single(i => i.LastName == "Harui").ID
    },
    new CourseAssignment {
        CourseID = courses.Single(c => c.Title == "Composition" ).CourseID,
        InstructorID = instructors.Single(i => i.LastName == "Abercrombie").ID
    },
    new CourseAssignment {
        CourseID = courses.Single(c => c.Title == "Literature" ).CourseID,
        InstructorID = instructors.Single(i => i.LastName == "Abercrombie").ID
    },
},
};

foreach (CourseAssignment ci in courseInstructors)
{
    context.CourseAssignments.Add(ci);
}
context.SaveChanges();

var enrollments = new Enrollment[]
{
    new Enrollment {
        StudentID = students.Single(s => s.LastName == "Alexander").ID,
        CourseID = courses.Single(c => c.Title == "Chemistry" ).CourseID,
        Grade = Grade.A
    },
    new Enrollment {
        StudentID = students.Single(s => s.LastName == "Alexander").ID,
        CourseID = courses.Single(c => c.Title == "Microeconomics" ).CourseID,
        Grade = Grade.C
    },
    new Enrollment {
        StudentID = students.Single(s => s.LastName == "Alexander").ID,
        CourseID = courses.Single(c => c.Title == "Macroeconomics" ).CourseID,
        Grade = Grade.B
    },
    new Enrollment {
        StudentID = students.Single(s => s.LastName == "Alonso").ID,
        CourseID = courses.Single(c => c.Title == "Calculus" ).CourseID,
        Grade = Grade.B
    },
    new Enrollment {
        StudentID = students.Single(s => s.LastName == "Alonso").ID,
        CourseID = courses.Single(c => c.Title == "Trigonometry" ).CourseID,
        Grade = Grade.B
    },
}

```

```

        new Enrollment {
            StudentID = students.Single(s => s.LastName == "Alonso").ID,
            CourseID = courses.Single(c => c.Title == "Composition" ).CourseID,
            Grade = Grade.B
        },
        new Enrollment {
            StudentID = students.Single(s => s.LastName == "Anand").ID,
            CourseID = courses.Single(c => c.Title == "Chemistry" ).CourseID
        },
        new Enrollment {
            StudentID = students.Single(s => s.LastName == "Anand").ID,
            CourseID = courses.Single(c => c.Title == "Microeconomics").CourseID,
            Grade = Grade.B
        },
        new Enrollment {
            StudentID = students.Single(s => s.LastName == "Barzdukas").ID,
            CourseID = courses.Single(c => c.Title == "Chemistry").CourseID,
            Grade = Grade.B
        },
        new Enrollment {
            StudentID = students.Single(s => s.LastName == "Li").ID,
            CourseID = courses.Single(c => c.Title == "Composition").CourseID,
            Grade = Grade.B
        },
        new Enrollment {
            StudentID = students.Single(s => s.LastName == "Justice").ID,
            CourseID = courses.Single(c => c.Title == "Literature").CourseID,
            Grade = Grade.B
        }
    }

    foreach (Enrollment e in enrollments)
    {
        var enrollmentInDataBase = context.Enrollments.Where(
            s =>
                s.Student.ID == e.StudentID &&
                s.Course.CourseID == e.CourseID).SingleOrDefault();
        if (enrollmentInDataBase == null)
        {
            context.Enrollments.Add(e);
        }
    }
    context.SaveChanges();
}
}

```

Como você viu no primeiro tutorial, a maioria do código apenas cria novos objetos de entidade e carrega dados de exemplo em propriedades, conforme necessário, para teste. Observe como as relações muitas para muitos são tratadas: o código cria relações com a criação de entidades nos conjuntos de entidades de junção

Enrollments e CourseAssignment .

## Adicionar uma migração

Salve as alterações e compile o projeto. Em seguida, abra a janela Comando na pasta do projeto e insira o comando `migrations add` (não execute ainda o comando de atualização de banco de dados):

```
dotnet ef migrations add ComplexDataModel
```

Você receberá um aviso sobre a possível perda de dados.

```
An operation was scaffolded that may result in the loss of data. Please review the migration for accuracy.  
Done. To undo this action, use 'ef migrations remove'
```

Se você tiver tentado executar o comando `database update` neste ponto (não faça isso ainda), receberá o seguinte erro:

```
A instrução ALTER TABLE entrou em conflito com a restrição FOREIGN KEY  
"FK_dbo.Course_dbo.Department_DepartmentID". O conflito ocorreu no banco de dados  
"ContosoUniversity", tabela "dbo.Departamento", coluna 'DepartmentID'.
```

Às vezes, quando você executa migrações com os dados existentes, precisa inserir dados de stub no banco de dados para atender às restrições de chave estrangeira. O código gerado no método `up` adiciona uma chave estrangeira `DepartmentID` que não permite valor nulo para a tabela `Curso`. Se já houver linhas na tabela `Curso` quando o código for executado, a operação `AddColumn` falhará, porque o SQL Server não saberá qual valor deve ser colocado na coluna que não pode ser nulo. Para este tutorial, você executará a migração em um novo banco de dados, mas em um aplicativo de produção, você precisará fazer com que a migração manipule os dados existentes. Portanto, as instruções a seguir mostram um exemplo de como fazer isso.

Para fazer a migração funcionar com os dados existentes, você precisa alterar o código para fornecer à nova coluna um valor padrão e criar um departamento de stub chamado "Temp" para atuar como o departamento padrão. Como resultado, as linhas `Curso` existentes serão todas relacionadas ao departamento "Temp" após a execução do método `Up`.

- Abra o arquivo `{timestamp}_ComplexDataModel.cs`.
- Comente a linha de código que adiciona a coluna `DepartmentID` à tabela `Curso`.

```
migrationBuilder.AlterColumn<string>(  
    name: "Title",  
    table: "Course",  
    maxLength: 50,  
    nullable: true,  
    oldClrType: typeof(string),  
    oldNullable: true);  
  
//migrationBuilder.AddColumn<int>(  
//    name: "DepartmentID",  
//    table: "Course",  
//    nullable: false,  
//    defaultValue: 0);
```

- Adicione o seguinte código realçado após o código que cria a tabela `Departamento`:

```

migrationBuilder.CreateTable(
    name: "Department",
    columns: table => new
    {
        DepartmentID = table.Column<int>(nullable: false)
            .Annotation("SqlServer:ValueGenerationStrategy",
            SqlServerValueGenerationStrategy.IdentityColumn),
        Budget = table.Column<decimal>(type: "money", nullable: false),
        InstructorID = table.Column<int>(nullable: true),
        Name = table.Column<string>(maxLength: 50, nullable: true),
        StartDate = table.Column<DateTime>(nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Department", x => x.DepartmentID);
        table.ForeignKey(
            name: "FK_Department_Instructor_InstructorID",
            column: x => x.InstructorID,
            principalTable: "Instructor",
            principalColumn: "ID",
            onDelete: ReferentialAction.Restrict);
    });
}

migrationBuilder.Sql("INSERT INTO dbo.Department (Name, Budget, StartDate) VALUES ('Temp', 0.00,
GETDATE())");
// Default value for FK points to department created above, with
// defaultValue changed to 1 in following AddColumn statement.

migrationBuilder.AddColumn<int>(
    name: "DepartmentID",
    table: "Course",
    nullable: false,
    defaultValue: 1);

```

Em um aplicativo de produção, você escreverá código ou scripts para adicionar linhas Departamento e relacionar linhas Curso às novas linhas Departamento. Em seguida, você não precisará mais do departamento "Temp" ou do valor padrão na coluna Course.DepartmentID.

Salve as alterações e compile o projeto.

## Alterar a cadeia de conexão e atualizar o banco de dados

Agora, você tem novo código na classe `DbInitializer` que adiciona dados de semente para as novas entidades a um banco de dados vazio. Para fazer com que o EF crie um novo banco de dados vazio, altere o nome do banco de dados na cadeia de conexão em `appsettings.json` para ContosoUniversity3 ou para outro nome que você ainda não usou no computador que está sendo usado.

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=
(localdb)\\mssqllocaldb;Database=ContosoUniversity3;Trusted_Connection=True;MultipleActiveResultSets=true"
  },
}
```

Salve as alterações em `appsettings.json`.

#### NOTE

Como alternativa à alteração do nome do banco de dados, você pode excluir o banco de dados. Use o **SSOX** (Pesquisador de Objetos do SQL Server) ou o comando `database drop` da CLI:

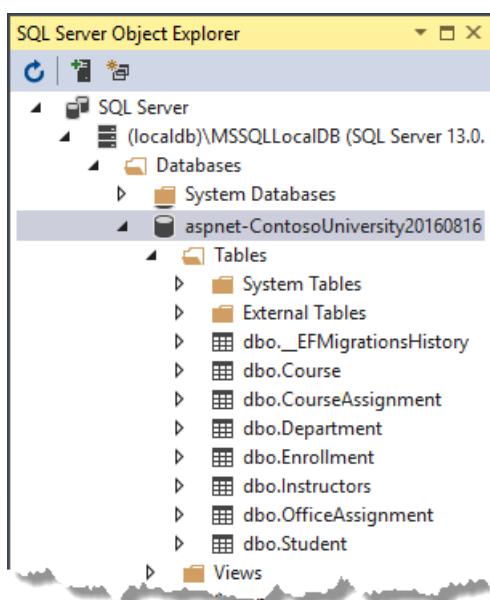
```
dotnet ef database drop
```

Depois que você tiver alterado o nome do banco de dados ou excluído o banco de dados, execute o comando `database update` na janela Comando para executar as migrações.

```
dotnet ef database update
```

Execute o aplicativo para fazer com que o método `DbInitializer.Initialize` execute e popule o novo banco de dados.

Abra o banco de dados no SSOX, como você fez anteriormente, e expanda o nó **Tabelas** para ver se todas as tabelas foram criadas. (Se você ainda tem o SSOX aberto do momento anterior, clique no botão **Atualizar**.)



Execute o aplicativo para disparar o código inicializador que propaga o banco de dados.

Clique com o botão direito do mouse na tabela **CourseAssignment** e selecione **Exibir Dados** para verificar se existem dados nela.

	CourseID	InstructorID
▶	2021	1
	2042	1
	1045	2
	1050	3
	3141	3
	1050	4
	4022	5
	4041	5
*	NULL	NULL

## Resumo

Agora você tem um modelo de dados mais complexo e um banco de dados correspondente. No tutorial a seguir, você aprenderá mais sobre como acessar dados relacionados.

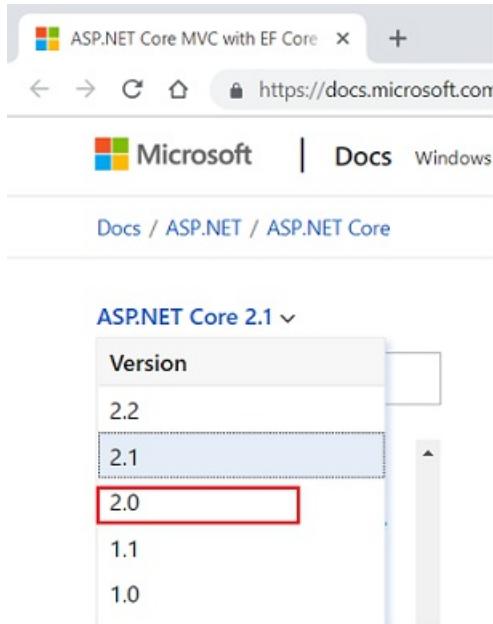
[ANTERIOR](#)

[PRÓXIMO](#)

# ASP.NET Core MVC com o EF Core – ler dados relacionados – 6 de 10

16/07/2018 • 26 minutes to read • [Edit Online](#)

Este tutorial não foi atualizado para o ASP.NET Core 2.1. A versão deste tutorial para o ASP.NET Core 2.0 pode ser consultada selecionando **ASP.NET Core 2.0** acima do sumário ou na parte superior da página:



A versão deste tutorial do [Razor Pages](#) para o ASP.NET Core 2.1 conta com várias melhorias em relação à versão 2.0.

O tutorial do 2.0 ensina a usar o ASP.NET Core MVC e o Entity Framework Core com controladores e exibições. O tutorial do Razor Pages foi atualizado com os seguintes aprimoramentos:

- É mais fácil de acompanhar. Por exemplo, o código de scaffolding foi bastante simplificado.
- Fornece mais práticas recomendadas do EF Core.
- Usa consultas mais eficientes.
- Usa a API do EF Core mais recente.

Por [Tom Dykstra](#) e [Rick Anderson](#)

O aplicativo web de exemplo Contoso University demonstra como criar aplicativos web do ASP.NET Core MVC usando o Entity Framework Core e o Visual Studio. Para obter informações sobre a série de tutoriais, consulte [o primeiro tutorial da série](#).

No tutorial anterior, você concluiu o modelo de dados Escola. Neste tutorial, você lerá e exibirá dados relacionados – ou seja, os dados que o Entity Framework carrega nas propriedades de navegação.

As ilustrações a seguir mostram as páginas com as quais você trabalhará.

Courses - Contoso Univ																								
<a href="#">←</a> <a href="#">→</a> <a href="#">⟳</a>   localhost:5813/Courses <a href="#">≡</a> <a href="#">★</a>   <a href="#">≡</a> <a href="#">✎</a> <a href="#">...</a>																								
Contoso University																								
<h2>Courses</h2>																								
<a href="#">Create New</a> <table border="1"> <thead> <tr> <th>Number</th> <th>Title</th> <th>Credits</th> <th>Department</th> <th></th> </tr> </thead> <tbody> <tr> <td>1045</td> <td>Calculus</td> <td>4</td> <td>Mathematics</td> <td><a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a></td> </tr> <tr> <td>1050</td> <td>Chemistry</td> <td>3</td> <td>Engineering</td> <td><a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a></td> </tr> <tr> <td>2021</td> <td>Composition</td> <td>3</td> <td>English</td> <td><a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a></td> </tr> </tbody> </table>					Number	Title	Credits	Department		1045	Calculus	4	Mathematics	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	1050	Chemistry	3	Engineering	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	2021	Composition	3	English	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Number	Title	Credits	Department																					
1045	Calculus	4	Mathematics	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>																				
1050	Chemistry	3	Engineering	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>																				
2021	Composition	3	English	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>																				

Instructors - Contoso U																												
<a href="#">←</a> <a href="#">→</a> <a href="#">⟳</a>   localhost:5813/Instructors/Index/1? <a href="#">≡</a> <a href="#">★</a>   <a href="#">≡</a> <a href="#">✎</a> <a href="#">...</a>																												
Contoso University																												
<h2>Instructors</h2>																												
<a href="#">Create New</a> <table border="1"> <thead> <tr> <th>Last Name</th> <th>First Name</th> <th>Hire Date</th> <th>Office</th> <th>Courses</th> <th></th> </tr> </thead> <tbody> <tr> <td>Abercrombie</td> <td>Kim</td> <td>1995-03-11</td> <td></td> <td>2021 Composition 2042 Literature</td> <td><a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a></td> </tr> <tr> <td>Fakhouri</td> <td>Fadi</td> <td>2002-07-06</td> <td>Smith 17</td> <td>1045 Calculus</td> <td><a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a></td> </tr> <tr> <td>Harui</td> <td>Roger</td> <td>1998-07-01</td> <td>Gowan 27</td> <td>1050 Chemistry 3141 Trigonometry</td> <td><a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a></td> </tr> </tbody> </table>					Last Name	First Name	Hire Date	Office	Courses		Abercrombie	Kim	1995-03-11		2021 Composition 2042 Literature	<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Fakhouri	Fadi	2002-07-06	Smith 17	1045 Calculus	<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Harui	Roger	1998-07-01	Gowan 27	1050 Chemistry 3141 Trigonometry	<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Last Name	First Name	Hire Date	Office	Courses																								
Abercrombie	Kim	1995-03-11		2021 Composition 2042 Literature	<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>																							
Fakhouri	Fadi	2002-07-06	Smith 17	1045 Calculus	<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>																							
Harui	Roger	1998-07-01	Gowan 27	1050 Chemistry 3141 Trigonometry	<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>																							

Courses Taught by Selected Instructor			
	Number	Title	Department
Select	2021	Composition	English
Select	2042	Literature	English
<h2>Students Enrolled in Selected Course</h2>			
Name	Grade		
Alonso, Meredith	B		
Li, Yan	B		

## Carregamento adiantado, explícito e lento de dados relacionados

Há várias maneiras pelas quais um software ORM (Object-Relational Mapping), como o Entity Framework, pode carregar dados relacionados nas propriedades de navegação de uma entidade:

- Carregamento adiantado. Quando a entidade é lida, os dados relacionados são recuperados com ela. Normalmente, isso resulta em uma única consulta de junção que recupera todos os dados necessários. Especifique o carregamento adiantado no Entity Framework Core usando os métodos `Include` e `ThenInclude`.

```
var departments = _context.Departments.Include(d => d.Courses);
foreach (Department d in departments)
{
    foreach(Course c in d.Courses)
    {
        courseList.Add(d.Name + c.Title);
    }
}
```

Query: all Department entities and related Course entities

Recupere alguns dos dados em consultas separadas e o EF "corrigirá" as propriedades de navegação. Ou seja, o EF adiciona de forma automática as entidades recuperadas separadamente no local em que pertencem nas propriedades de navegação de entidades recuperadas anteriormente. Para a consulta que recupera dados relacionados, você pode usar o método `Load` em vez de um método que retorna uma lista ou um objeto, como `ToList` ou `Single`.

```
var departments = _context.Departments;
foreach (Department d in departments)
{
    _context.Courses.Where(c => c.DepartmentID == d.DepartmentID).Load();
    foreach (Course c in d.Courses)
    {
        courseList.Add(d.Name + c.Title);
    }
}
```

Query: all Department rows

Query: Course rows related to Department d

- Carregamento explícito. Quando a entidade é lida pela primeira vez, os dados relacionados não são recuperados. Você escreve o código que recupera os dados relacionados se eles são necessários. Como no caso do carregamento adiantado com consultas separadas, o carregamento explícito resulta no envio de várias consultas ao banco de dados. A diferença é que, com o carregamento explícito, o código especifica as propriedades de navegação a serem carregadas. No Entity Framework Core 1.1, você pode usar o método `Load` para fazer o carregamento explícito. Por exemplo:

```
var departments = _context.Departments;
foreach (Department d in departments)
{
    _context.Entry(d).Collection(p => p.Courses).Load();
    foreach (Course c in d.Courses)
    {
        courseList.Add(d.Name + c.Title);
    }
}
```

Query: all Department rows

Query: Course rows related to Department d

- Carregamento lento. Quando a entidade é lida pela primeira vez, os dados relacionados não são recuperados. No entanto, na primeira vez que você tenta acessar uma propriedade de navegação, os dados necessários para essa propriedade de navegação são recuperados automaticamente. Uma consulta é enviada ao banco de dados sempre que você tenta obter dados de uma propriedade de navegação pela primeira vez. O Entity Framework Core 1.0 não dá suporte ao carregamento lento.

### Considerações sobre desempenho

Se você sabe que precisa de dados relacionados para cada entidade recuperada, o carregamento adiantado costuma oferecer o melhor desempenho, porque uma única consulta enviada para o banco de dados é geralmente mais eficiente do que consultas separadas para cada entidade recuperada. Por exemplo, suponha que cada departamento tenha dez cursos relacionados. O carregamento adiantado de todos os dados relacionados resultará em apenas uma única consulta (junção) e uma única viagem de ida e volta para o banco

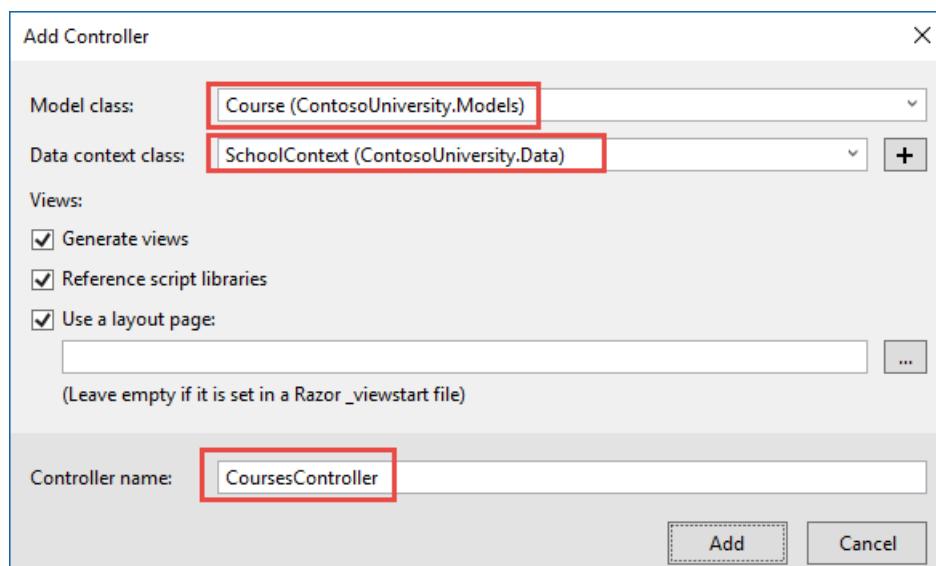
de dados. Uma consulta separada para cursos de cada departamento resultará em onze viagens de ida e volta para o banco de dados. As viagens de ida e volta extras para o banco de dados são especialmente prejudiciais ao desempenho quando a latência é alta.

Por outro lado, em alguns cenários, consultas separadas são mais eficientes. O carregamento adiantado de todos os dados relacionados em uma consulta pode fazer com que uma junção muito complexa seja gerada, que o SQL Server não consegue processar com eficiência. Ou se precisar acessar as propriedades de navegação de uma entidade somente para um subconjunto de um conjunto de entidades que está sendo processado, consultas separadas poderão ter um melhor desempenho, pois o carregamento adiantado de tudo desde o início recupera mais dados do que você precisa. Se o desempenho for crítico, será melhor testar o desempenho das duas maneiras para fazer a melhor escolha.

## Criar uma página Courses que exibe o nome do Departamento

A entidade Course inclui uma propriedade de navegação que contém a entidade Department do departamento ao qual o curso é atribuído. Para exibir o nome do departamento atribuído em uma lista de cursos, você precisa obter a propriedade Name da entidade Department que está na propriedade de navegação `Course.Department`.

Crie um controlador chamado CoursesController para o tipo de entidade Course, usando as mesmas opções para o scaffolder **Controlador MVC com exibições, usando o Entity Framework** que você usou anteriormente para o controlador Alunos, conforme mostrado na seguinte ilustração:



Abra `CoursesController.cs` e examine o método `Index`. O scaffolding automático especificou o carregamento adiantado para a propriedade de navegação `Department` usando o método `Include`.

Substitua o método `Index` pelo seguinte código, que usa um nome mais apropriado para o `IQueryable` que retorna as entidades Course (`courses` em vez de `schoolContext`):

```
public async Task<IActionResult> Index()
{
    var courses = _context.Courses
        .Include(c => c.Department)
        .AsNoTracking();
    return View(await courses.ToListAsync());
}
```

Abra `Views/Courses/Index.cshtml` e substitua o código de modelo pelo código a seguir. As alterações são realçadas:

```

@model IEnumerable<ContosoUniversity.Models.Course>

 @{
     ViewData["Title"] = "Courses";
 }

<h2>Courses</h2>

<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.CourseID)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Title)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Credits)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Department)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.CourseID)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Title)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Credits)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Department.Name)
                </td>
                <td>
                    <a asp-action="Edit" asp-route-id="@item.CourseID">Edit</a> |
                    <a asp-action="Details" asp-route-id="@item.CourseID">Details</a> |
                    <a asp-action="Delete" asp-route-id="@item.CourseID">Delete</a>
                </td>
            </tr>
        }
    </tbody>
</table>

```

Você fez as seguintes alterações no código gerado por scaffolding:

- Alterou o cabeçalho de Índice para Cursos.
- Adicionou uma coluna **Número** que mostra o valor da propriedade `CourseID`. Por padrão, as chaves primárias não são geradas por scaffolding porque normalmente não têm sentido para os usuários finais. No entanto, nesse caso, a chave primária é significativa e você deseja mostrá-la.
- Alterou a coluna **Departamento** para que ela exiba o nome de departamento. O código exibe a propriedade `Name` da entidade `Department` que é carregada na propriedade de navegação `Department`:

```
@Html.DisplayFor(modelItem => item.Department.Name)
```

Execute o aplicativo e selecione a guia **Cursos** para ver a lista com nomes de departamentos.

Number	Title	Credits	Department	
1045	Calculus	4	Mathematics	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
1050	Chemistry	3	Engineering	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
2021	Composition	3	English	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

## Criar uma página Instrutores que mostra Cursos e Registros

Nesta seção, você criará um controlador e uma exibição para a entidade Instructor para exibir a página Instrutores:

**Instructors**

Create New

Last Name	First Name	Hire Date	Office	Courses	
Abercrombie	Kim	1995-03-11		2021 Composition 2042 Literature	Select   Edit   Details   Delete
Fakhouri	Fadi	2002-07-06	Smith 17	1045 Calculus	Select   Edit   Details   Delete
Harui	Roger	1998-07-01	Gowan 27	1050 Chemistry 3141 Trigonometry	Select   Edit   Details   Delete

**Courses Taught by Selected Instructor**

Number	Title	Department
Select	2021	Composition English
Select	2042	Literature English

**Students Enrolled in Selected Course**

Name	Grade
Alonso, Meredith	B
Li, Yan	B

Essa página lê e exibe dados relacionados das seguintes maneiras:

- A lista de instrutores exibe dados relacionados da entidade `OfficeAssignment`. As entidades `Instructor` e `OfficeAssignment` estão em uma relação um para zero ou um. Você usará o carregamento adiantado para as entidades `OfficeAssignment`. Conforme explicado anteriormente, o carregamento adiantado é geralmente mais eficiente quando você precisa dos dados relacionados para todas as linhas recuperadas da tabela primária. Nesse caso, você deseja exibir atribuições de escritório para todos os instrutores exibidos.
- Quando o usuário seleciona um instrutor, as entidades `Course` relacionadas são exibidas. As entidades `Instructor` e `Course` estão em uma relação muitos para muitos. Você usará o carregamento adiantado para as entidades `Course` e suas entidades `Department` relacionadas. Nesse caso, consultas separadas podem ser mais eficientes porque você precisa de cursos somente para o instrutor selecionado. No entanto, este exemplo mostra como usar o carregamento adiantado para propriedades de navegação em entidades que estão nas propriedades de navegação.
- Quando o usuário seleciona um curso, dados relacionados do conjunto de entidades `Enrollments` são exibidos. As entidades `Course` e `Enrollment` estão em uma relação um para muitos. Você usará consultas

separadas para entidades Enrollment e suas entidades Student relacionadas.

### Criar um modelo de exibição para a exibição Índice de Instrutor

A página Instrutores mostra dados de três tabelas diferentes. Portanto, você criará um modelo de exibição que inclui três propriedades, cada uma contendo os dados de uma das tabelas.

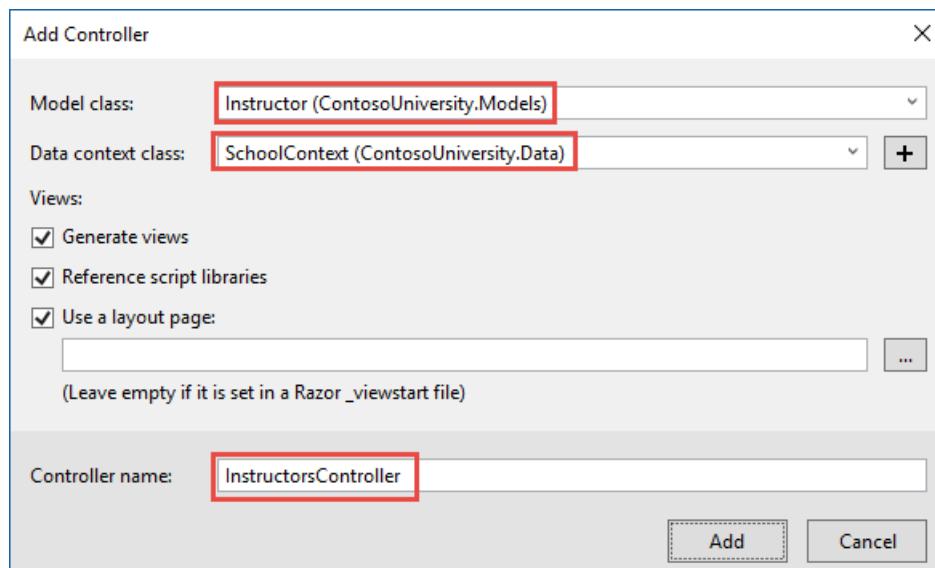
Na pasta *SchoolViewModels*, crie *InstructorIndexData.cs* e substitua o código existente pelo seguinte código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ContosoUniversity.Models.SchoolViewModels
{
    public class InstructorIndexData
    {
        public IEnumerable<Instructor> Instructors { get; set; }
        public IEnumerable<Course> Courses { get; set; }
        public IEnumerable<Enrollment> Enrollments { get; set; }
    }
}
```

### Criar exibições e o controlador Instrutor

Crie um controlador Instrutores com ações de leitura/gravação do EF, conforme mostrado na seguinte ilustração:



Abra *InstructorsController.cs* e adicione um usando a instrução para o namespace ViewModels:

```
using ContosoUniversity.Models.SchoolViewModels;
```

Substitua o método Index pelo código a seguir para fazer o carregamento adiantado de dados relacionados e colocá-los no modelo de exibição.

```

public async Task<IActionResult> Index(int? id, int? courseID)
{
    var viewModel = new InstructorIndexData();
    viewModel.Instructors = await _context.Instructors
        .Include(i => i.OfficeAssignment)
        .Include(i => i.CourseAssignments)
        .ThenInclude(i => i.Course)
        .ThenInclude(i => i.Enrollments)
        .ThenInclude(i => i.Student)
        .Include(i => i.CourseAssignments)
        .ThenInclude(i => i.Course)
        .ThenInclude(i => i.Department)
    .AsNoTracking()
    .OrderBy(i => i.LastName)
    .ToListAsync();

    if (id != null)
    {
        ViewData["InstructorID"] = id.Value;
        Instructor instructor = viewModel.Instructors.Where(
            i => i.ID == id.Value).Single();
        viewModel.Courses = instructor.CourseAssignments.Select(s => s.Course);
    }

    if (courseID != null)
    {
        ViewData["CourseID"] = courseID.Value;
        viewModel.Enrollments = viewModel.Courses.Where(
            x => x.CourseID == courseID).Single().Enrollments;
    }

    return View(viewModel);
}

```

O método aceita dados de rota opcionais (`id`) e um parâmetro de cadeia de caracteres de consulta (`courseID`) que fornece os valores de ID do curso e do instrutor selecionados. Os parâmetros são fornecidos pelos hiperlinks **Selecionar** na página.

O código começa com a criação de uma instância do modelo de exibição e colocando-a na lista de instrutores. O código especifica o carregamento adiantado para as propriedades de navegação `Instructor.OfficeAssignment` e `Instructor.CourseAssignments`. Dentro da propriedade `CourseAssignments`, a propriedade `Course` é carregada e, dentro dela, as propriedades `Enrollments` e `Department` são carregadas e, dentro de cada entidade `Enrollment`, a propriedade `Student` é carregada.

```

viewModel.Instructors = await _context.Instructors
    .Include(i => i.OfficeAssignment)
    .Include(i => i.CourseAssignments)
    .ThenInclude(i => i.Course)
    .ThenInclude(i => i.Enrollments)
    .ThenInclude(i => i.Student)
    .Include(i => i.CourseAssignments)
    .ThenInclude(i => i.Course)
    .ThenInclude(i => i.Department)
    .AsNoTracking()
    .OrderBy(i => i.LastName)
    .ToListAsync();

```

Como a exibição sempre exige a entidade `OfficeAssignment`, é mais eficiente buscar isso na mesma consulta. As entidades `Course` são necessárias quando um instrutor é selecionado na página da Web; portanto, uma única consulta é melhor do que várias consultas apenas se a página é exibida com mais frequência com um curso selecionado do que sem ele.

O código repete `CourseAssignments` e `Course` porque você precisa de duas propriedades de `Course`. A primeira cadeia de caracteres de chamadas `ThenInclude` obtém `CourseAssignment.Course`, `Course.Enrollments` e `Enrollment.Student`.

```
viewModel.Instructors = await _context.Instructors
    .Include(i => i.OfficeAssignment)
    .Include(i => i.CourseAssignments)
    .ThenInclude(i => i.Course)
        .ThenInclude(i => i.Enrollments)
            .ThenInclude(i => i.Student)
    .Include(i => i.CourseAssignments)
    .ThenInclude(i => i.Course)
        .ThenInclude(i => i.Department)
    .AsNoTracking()
    .OrderBy(i => i.LastName)
    .ToListAsync();
```

Nesse ponto do código, outro `ThenInclude` se refere às propriedades de navegação de `Student`, que não é necessário. Mas a chamada a `Include` é reiniciada com propriedades `Instructor` e, portanto, você precisa passar pela cadeia novamente, dessa vez, especificando `Course.Department` em vez de `Course.Enrollments`.

```
viewModel.Instructors = await _context.Instructors
    .Include(i => i.OfficeAssignment)
    .Include(i => i.CourseAssignments)
    .ThenInclude(i => i.Course)
        .ThenInclude(i => i.Enrollments)
            .ThenInclude(i => i.Student)
    .Include(i => i.CourseAssignments)
    .ThenInclude(i => i.Course)
        .ThenInclude(i => i.Department)
    .AsNoTracking()
    .OrderBy(i => i.LastName)
    .ToListAsync();
```

O código a seguir é executado quando o instrutor é selecionado. O instrutor selecionado é recuperado da lista de instrutores no modelo de exibição. Em seguida, a propriedade `Courses` do modelo de exibição é carregada com as entidades `Course` da propriedade de navegação `CourseAssignments` desse instrutor.

```
if (id != null)
{
    ViewData["InstructorID"] = id.Value;
    Instructor instructor = viewModel.Instructors.Where(
        i => i.ID == id.Value).Single();
    viewModel.Courses = instructor.CourseAssignments.Select(s => s.Course);
}
```

O método `Where` retorna uma coleção, mas nesse caso, os critérios passado para esse método resultam no retorno de apenas uma única entidade `Instructor`. O método `Single` converte a coleção em uma única entidade `Instructor`, que fornece acesso à propriedade `CourseAssignments` dessa entidade. A propriedade `CourseAssignments` contém entidades `CourseAssignment`, das quais você deseja apenas entidades `Course` relacionadas.

Use o método `Single` em uma coleção quando souber que a coleção terá apenas um item. O método `Single` gera uma exceção se a coleção passada para ele está vazia ou se há mais de um item. Uma alternativa é `SingleOrDefault`, que retorna um valor padrão (nulo, nesse caso) se a coleção está vazia. No entanto, nesse caso, isso ainda resultará em uma exceção (da tentativa de encontrar uma propriedade `Courses` em uma referência nula), e a mensagem de exceção menos claramente indicará a causa do problema. Quando você chama o

método `Single`, também pode passar a condição `Where`, em vez de chamar o método `Where` separadamente:

```
.Single(i => i.ID == id.Value)
```

Em vez de:

```
.Where(i => i.ID == id.Value).Single()
```

Em seguida, se um curso foi selecionado, o curso selecionado é recuperado na lista de cursos no modelo de exibição. Em seguida, a propriedade `Enrollments` do modelo de exibição é carregada com as entidades `Enrollment` da propriedade de navegação `Enrollments` desse curso.

```
if (courseID != null)
{
    ViewData["CourseID"] = courseID.Value;
    viewModel.Enrollments = viewModel.Courses.Where(
        x => x.CourseID == courseID).Single().Enrollments;
}
```

### Modificar a exibição Índice de Instrutor

Em `Views/Instructors/Index.cshtml`, substitua o código de modelo pelo código a seguir. As alterações são realçadas.

```

@model ContosoUniversity.Models.SchoolViewModels.InstructorIndexData

 @{
     ViewData["Title"] = "Instructors";
 }

<h2>Instructors</h2>

<p>
    <a asp-action="Create">Create New</a>
</p>


| Last Name | First Name | Hire Date | Office | Courses |  |
|-----------|------------|-----------|--------|---------|--|
|-----------|------------|-----------|--------|---------|--|


```

Você fez as seguintes alterações no código existente:

- Alterou a classe de modelo para `InstructorIndexData`.
- Alterou o título de página de **Índice** para **Instrutores**.
- Adicionou uma coluna **Office** que exibe `item.OfficeAssignment.Location` somente se `item.OfficeAssignment` não é nulo. (Como essa é uma relação um para zero ou um, pode não haver uma entidade OfficeAssignment relacionada.)

```
@if (item.OfficeAssignment != null)
{
    @item.OfficeAssignment.Location
}
```

- Adicionou uma coluna **Courses** que exibe os cursos ministrados por cada instrutor. Consulte [Transição de linha explícita com `@:`](#) para obter mais informações sobre essa sintaxe Razor.
- Adicionou um código que adiciona `class="success"` dinamicamente ao elemento `tr` do instrutor selecionado. Isso define uma cor da tela de fundo para a linha selecionada usando uma classe Bootstrap.

```
string selectedRow = "";
if (item.ID == (int?)ViewData["InstructorID"])
{
    selectedRow = "success";
}
<tr class="@selectedRow">
```

- Adicionou um novo hiperlink rotulado **Selecionar** imediatamente antes dos outros links em cada linha, o que faz com que a ID do instrutor selecionado seja enviada para o método `Index`.

```
<a asp-action="Index" asp-route-id="@item.ID">Select</a> |
```

Execute o aplicativo e selecione a guia **Instrutores**. A página exibe a propriedade Location das entidades OfficeAssignment relacionadas e uma célula de tabela vazia quando não há nenhuma entidade OfficeAssignment relacionada.

Last Name	First Name	Hire Date	Office	Courses
Abercrombie	Kim	1995-03-11	Smith 17	2021 Composition 2042 Literature <a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Fakhouri	Fadi	2002-07-06	Smith 17	1045 Calculus <a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

No arquivo `Views/Instructors/Index.cshtml`, após o elemento de tabela de fechamento (ao final do arquivo),

adicone o código a seguir. Esse código exibe uma lista de cursos relacionados a um instrutor quando um instrutor é selecionado.

```
@if (Model.Courses != null)
{
    <h3>Courses Taught by Selected Instructor</h3>
    <table class="table">
        <tr>
            <th></th>
            <th>Number</th>
            <th>Title</th>
            <th>Department</th>
        </tr>

        @foreach (var item in Model.Courses)
        {
            string selectedRow = "";
            if (item.CourseID == (int?)ViewData["CourseID"])
            {
                selectedRow = "success";
            }
            <tr class="@selectedRow">
                <td>
                    @Html.ActionLink("Select", "Index", new { courseID = item.CourseID })
                </td>
                <td>
                    @item.CourseID
                </td>
                <td>
                    @item.Title
                </td>
                <td>
                    @item.Department.Name
                </td>
            </tr>
        }
    </table>
}
```

Esse código lê a propriedade `Courses` do modelo de exibição para exibir uma lista de cursos. Também fornece um hiperlink **Selecionar** que envia a ID do curso selecionado para o método de ação `Index`.

Atualize a página e selecione um instrutor. Agora, você verá uma grade que exibe os cursos atribuídos ao instrutor selecionado, e para cada curso, verá o nome do departamento atribuído.

## Courses Taught by Selected Instructor

	Number	Title	Department
Select	2021	Composition	English
Select	2042	Literature	English

Após o bloco de código que você acabou de adicionar, adicione o código a seguir. Isso exibe uma lista dos alunos que estão registrados em um curso quando esse curso é selecionado.

```

@if (Model.Enrollments != null)
{
    <h3>
        Students Enrolled in Selected Course
    </h3>
    <table class="table">
        <tr>
            <th>Name</th>
            <th>Grade</th>
        </tr>
        @foreach (var item in Model.Enrollments)
        {
            <tr>
                <td>
                    @item.Student.FullName
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Grade)
                </td>
            </tr>
        }
    </table>
}

```

Esse código lê a propriedade Enrollments do modelo de exibição para exibir uma lista dos alunos registrados no curso.

Atualize a página novamente e selecione um instrutor. Em seguida, selecione um curso para ver a lista de alunos

registrados e suas notas.

Instructors - Contoso U X + ← → ⏪ | localhost:5813/Instructors/Index/1? ⏴ ⏵ ⏷ ⏸ ⏹ ⏺ ⏻ ⏻

Contoso University

## Instructors

Create New

Last Name	First Name	Hire Date	Office	Courses	
Abercrombie	Kim	1995-03-11		2021 Composition 2042 Literature	Select   Edit   Details   Delete
Fakhouri	Fadi	2002-07-06	Smith 17	1045 Calculus	Select   Edit   Details   Delete
Harui	Roger	1998-07-01	Gowan 27	1050 Chemistry 3141 Trigonometry	Select   Edit   Details   Delete

### Courses Taught by Selected Instructor

Number	Title	Department
Select 2021	Composition	English
Select 2042	Literature	English

### Students Enrolled in Selected Course

Name	Grade
Alonso, Meredith	B
Li, Yan	B

## Carregamento explícito

Quando você recuperou a lista de instrutores em `InstructorsController.cs`, você especificou o carregamento adiantado para a propriedade de navegação `CourseAssignments`.

Suponha que os usuários esperados raramente desejem ver registros em um curso e um instrutor selecionados. Nesse caso, talvez você deseje carregar os dados de registro somente se eles forem solicitados. Para ver um exemplo de como fazer carregamento explícito, substitua o método `Index` pelo código a seguir, que remove o carregamento adiantado para `Enrollments` e carrega essa propriedade de forma explícita. As alterações de código são realçadas.

```

public async Task<IActionResult> Index(int? id, int? courseID)
{
    var viewModel = new InstructorIndexData();
    viewModel.Instructors = await _context.Instructors
        .Include(i => i.OfficeAssignment)
        .Include(i => i.CourseAssignments)
        .ThenInclude(i => i.Course)
        .ThenInclude(i => i.Department)
        .OrderBy(i => i.LastName)
        .ToListAsync();

    if (id != null)
    {
        ViewData["InstructorID"] = id.Value;
        Instructor instructor = viewModel.Instructors.Where(
            i => i.ID == id.Value).Single();
        viewModel.Courses = instructor.CourseAssignments.Select(s => s.Course);
    }

    if (courseID != null)
    {
        ViewData["CourseID"] = courseID.Value;
        var selectedCourse = viewModel.Courses.Where(x => x.CourseID == courseID).Single();
        await _context.Entry(selectedCourse).Collection(x => x.Enrollments).LoadAsync();
        foreach (Enrollment enrollment in selectedCourse.Enrollments)
        {
            await _context.Entry(enrollment).Reference(x => x.Student).LoadAsync();
        }
        viewModel.Enrollments = selectedCourse.Enrollments;
    }

    return View(viewModel);
}

```

O novo código remove as chamadas do método `ThenInclude` para dados de registro do código que recupera as entidades do instrutor. Se um curso e um instrutor são selecionados, o código realçado recupera entidades `Enrollment` para o curso selecionado e as entidades `Student` para cada `Enrollment`.

Execute que o aplicativo, acesse a página Índice de Instrutores agora e você não verá nenhuma diferença no que é exibido na página, embora você tenha alterado a maneira como os dados são recuperados.

## Resumo

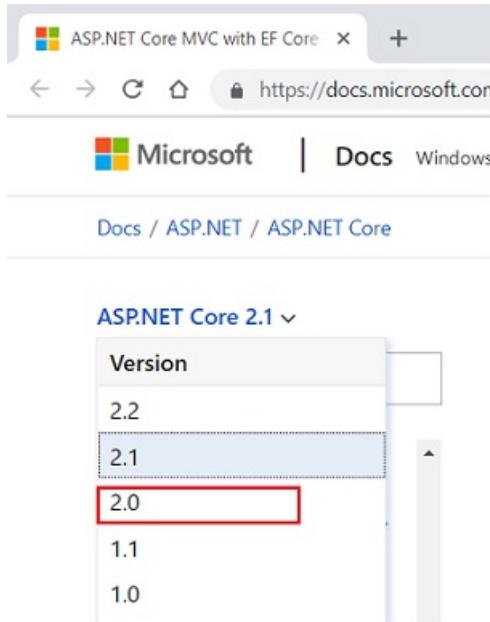
Agora, você usou o carregamento adiantado com uma consulta e com várias consultas para ler dados relacionados nas propriedades de navegação. No próximo tutorial, você aprenderá a atualizar dados relacionados.

[ANTERIOR](#)
[PRÓXIMO](#)

# ASP.NET Core MVC com o EF Core – atualizar dados relacionados – 7 de 10

30/10/2018 • 32 minutes to read • [Edit Online](#)

Este tutorial não foi atualizado para o ASP.NET Core 2.1. A versão deste tutorial para o ASP.NET Core 2.0 pode ser consultada selecionando **ASP.NET Core 2.0** acima do sumário ou na parte superior da página:



A versão deste tutorial do [Razor Pages](#) para o ASP.NET Core 2.1 conta com várias melhorias em relação à versão 2.0.

O tutorial do 2.0 ensina a usar o ASP.NET Core MVC e o Entity Framework Core com controladores e exibições. O tutorial do Razor Pages foi atualizado com os seguintes aprimoramentos:

- É mais fácil de acompanhar. Por exemplo, o código de scaffolding foi bastante simplificado.
- Fornece mais práticas recomendadas do EF Core.
- Usa consultas mais eficientes.
- Usa a API do EF Core mais recente.

Por [Tom Dykstra](#) e [Rick Anderson](#)

O aplicativo web de exemplo Contoso University demonstra como criar aplicativos web do ASP.NET Core MVC usando o Entity Framework Core e o Visual Studio. Para obter informações sobre a série de tutoriais, consulte [o primeiro tutorial da série](#).

No tutorial anterior, você exibiu dados relacionados; neste tutorial, você atualizará dados relacionados pela atualização dos campos de chave estrangeira e das propriedades de navegação.

As ilustrações a seguir mostram algumas das páginas com as quais você trabalhará.

Edit - Col X + - □ ×

← ⌂ localhost:58 | ⚑ ⚒ ⚓ ⚔

Contoso University ⚓

# Edit

Course

---

**Number**  
1000

**Title**  
 ×

**Credits**

**Department**  
 ▼



Last Name  
Abercrombie

First Name  
Kim

Hire Date  
3/11/1995

Office Location  
44/3P

1000 Algebra 2    1045 Calculus    1050 Chemistry  
 2021 Composition    2042 Literature    3141 Trigonometry  
 4022 Microeconomics    4041 Macroeconomics

Save

## Personalizar as páginas Criar e Editar dos cursos

Quando uma nova entidade de curso é criada, ela precisa ter uma relação com um departamento existente. Para facilitar isso, o código gerado por scaffolding inclui métodos do controlador e exibições Criar e Editar que incluem uma lista suspensa para seleção do departamento. A lista suspensa define a propriedade de chave estrangeira `Course.DepartmentID`, e isso é tudo o que o Entity Framework precisa para carregar a propriedade de navegação `Department` com a entidade Department apropriada. Você usará o código gerado por scaffolding, mas o alterará ligeiramente para adicionar tratamento de erro e classificação à lista suspensa.

Em `CoursesController.cs`, exclua os quatro métodos Create e Edit e substitua-os pelo seguinte código:

```
public IActionResult Create()
{
    PopulateDepartmentsDropDownList();
    return View();
}
```

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("CourseID,Credits,DepartmentID,Title")] Course course)
{
    if (ModelState.IsValid)
    {
        _context.Add(course);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    PopulateDepartmentsDropDownList(course.DepartmentID);
    return View(course);
}
```

```
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var course = await _context.Courses
        .AsNoTracking()
        .SingleOrDefaultAsync(m => m.CourseID == id);
    if (course == null)
    {
        return NotFound();
    }
    PopulateDepartmentsDropDownList(course.DepartmentID);
    return View(course);
}
```

```

[HttpPost, ActionName("Edit")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> EditPost(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var courseToUpdate = await _context.Courses
        .SingleOrDefaultAsync(c => c.CourseID == id);

    if (await TryUpdateModelAsync<Course>(courseToUpdate,
        "",
        c => c.Credits, c => c.DepartmentID, c => c.Title))
    {
        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateException /* ex */)
        {
            //Log the error (uncomment ex variable name and write a log.)
            ModelState.AddModelError("", "Unable to save changes. " +
                "Try again, and if the problem persists, " +
                "see your system administrator.");
        }
        return RedirectToAction(nameof(Index));
    }
    PopulateDepartmentsDropDownList(courseToUpdate.DepartmentID);
    return View(courseToUpdate);
}

```

Após o método `HttpPost` `Edit`, crie um novo método que carrega informações de departamento para a lista suspensa.

```

private void PopulateDepartmentsDropDownList(object selectedDepartment = null)
{
    var departmentsQuery = from d in _context.Departments
                           orderby d.Name
                           select d;
    ViewBag.DepartmentID = new SelectList(departmentsQuery.AsNoTracking(), "DepartmentID", "Name",
        selectedDepartment);
}

```

O método `PopulateDepartmentsDropDownList` obtém uma lista de todos os departamentos classificados por nome, cria uma coleção `SelectList` para uma lista suspensa e passa a coleção para a exibição em `ViewBag`. O método aceita o parâmetro `selectedDepartment` opcional que permite que o código de chamada especifique o item que será selecionado quando a lista suspensa for renderizada. A exibição passará o nome "DepartmentID" para o auxiliar de marcação `<select>` e o auxiliar então saberá que deve examinar o objeto `ViewBag` em busca de uma `SelectList` chamada "DepartmentID".

O método `HttpGet` `Create` chama o método `PopulateDepartmentsDropDownList` sem definir o item selecionado, porque um novo curso do departamento ainda não foi estabelecido:

```

public IActionResult Create()
{
    PopulateDepartmentsDropDownList();
    return View();
}

```

O método `HttpGet` `Edit` define o item selecionado, com base na ID do departamento já atribuído ao curso que está sendo editado:

```
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var course = await _context.Courses
        .AsNoTracking()
        .SingleOrDefaultAsync(m => m.CourseID == id);
    if (course == null)
    {
        return NotFound();
    }
    PopulateDepartmentsDropDownList(course.DepartmentID);
    return View(course);
}
```

Os métodos `HttpPost` para `Create` e `Edit` também incluem o código que define o item selecionado quando eles exibem novamente a página após um erro. Isso garante que quando a página for exibida novamente para mostrar a mensagem de erro, qualquer que tenha sido o departamento selecionado permaneça selecionado.

### Adicionar `.AsNoTracking` aos métodos `Details` e `Delete`

Para otimizar o desempenho das páginas `Detalhes do Curso` e `Excluir`, adicione chamadas `AsNoTracking` aos métodos `HttpGet` `Details` e `Delete`.

```
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var course = await _context.Courses
        .Include(c => c.Department)
        .AsNoTracking()
        .SingleOrDefaultAsync(m => m.CourseID == id);
    if (course == null)
    {
        return NotFound();
    }

    return View(course);
}
```

```

public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var course = await _context.Courses
        .Include(c => c.Department)
        .AsNoTracking()
        .SingleOrDefaultAsync(m => m.CourseID == id);
    if (course == null)
    {
        return NotFound();
    }

    return View(course);
}

```

## Modificar as exibições Curso

Em *Views/Courses/Create.cshtml*, adicione uma opção "Selecionar Departamento" à lista suspensa **Departamento**, altere a legenda de **DepartmentID** para **Departamento** e adicione uma mensagem de validação.

```

<div class="form-group">
    <label asp-for="Department" class="control-label"></label>
    <select asp-for="DepartmentID" class="form-control" asp-items="ViewBag.DepartmentID">
        <option value="">-- Select Department --</option>
    </select>
    <span asp-validation-for="DepartmentID" class="text-danger" />

```

Em *Views/Courses/Edit.cshtml*, faça a mesma alteração no campo Departamento que você acabou de fazer em *Create.cshtml*.

Também em *Views/Courses/Edit.cshtml*, adicione um campo de número de curso antes do campo **Título**. Como o número de curso é a chave primária, ele é exibido, mas não pode ser alterado.

```

<div class="form-group">
    <label asp-for="CourseID" class="control-label"></label>
    <div>@Html.DisplayFor(model => model.CourseID)</div>
</div>

```

Já existe um campo oculto (`<input type="hidden">`) para o número de curso na exibição Editar. A adição de um auxiliar de marcação `<label>` não elimina a necessidade do campo oculto, porque ele não faz com que o número de curso seja incluído nos dados postados quando o usuário clica em **Salvar** na página **Editar**.

Em *Views/Courses/Delete.cshtml*, adicione um campo de número de curso na parte superior e altere a ID do departamento para o nome do departamento.

```

@model ContosoUniversity.Models.Course

 @{
     ViewData["Title"] = "Delete";
 }

<h2>Delete</h2>

<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>Course</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.CourseID)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.CourseID)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Title)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Title)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Credits)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Credits)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Department)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Department.Name)
        </dd>
    </dl>

    <form asp-action="Delete">
        <div class="form-actions no-color">
            <input type="submit" value="Delete" class="btn btn-default" /> |
            <a asp-action="Index">Back to List</a>
        </div>
    </form>
</div>

```

Em *Views/Courses/Details.cshtml*, faça a mesma alteração que você acabou de fazer para *Delete.cshtml*.

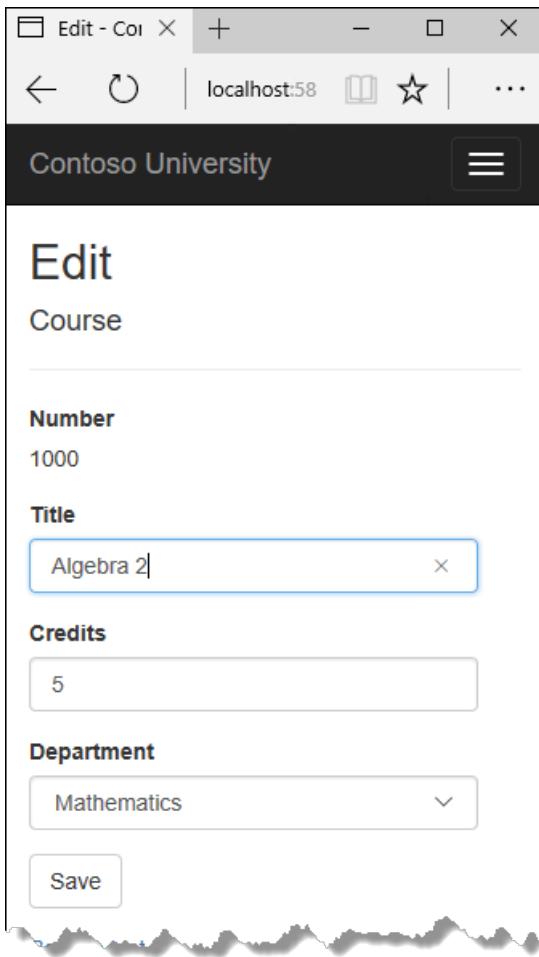
## Testar as páginas Curso

Execute o aplicativo, selecione a guia **Cursos**, clique em **Criar Novo** e insira dados para um novo curso:

The screenshot shows a web browser window with the address bar displaying 'localhost:581'. The title bar says 'Create - C...' and has standard window controls. The main content area is titled 'Create' and has a sub-section 'Course'. It contains four input fields: 'Number' with value '1000', 'Title' with value 'Algebra', 'Credits' with value '5', and a dropdown 'Department' set to 'Mathematics'. At the bottom is a large blue 'Create' button.

Clique em **Criar**. A página Índice de Cursos é exibida com o novo curso adicionado à lista. O nome do departamento na lista de páginas de Índice é obtido da propriedade de navegação, mostrando que a relação foi estabelecida corretamente.

Clique em **Editar** em um curso na página Índice de Cursos.



Altere dados na página e clique em **Salvar**. A página Índice de Cursos é exibida com os dados de cursos atualizados.

## Adicionar uma página Editar para instrutores

Quando você edita um registro de instrutor, deseja poder atualizar a atribuição de escritório do instrutor. A entidade Instructor tem uma relação um para zero ou um com a entidade OfficeAssignment, o que significa que o código deve manipular as seguintes situações:

- Se o usuário apagar a atribuição de escritório e ela originalmente tinha um valor, exclua a entidade OfficeAssignment.
- Se o usuário inserir um valor de atribuição de escritório e ele originalmente estava vazio, crie uma nova entidade OfficeAssignment.
- Se o usuário alterar o valor de uma atribuição de escritório, altere o valor em uma entidade OfficeAssignment existente.

### Atualizar o controlador Instrutores

Em *InstructorsController.cs*, altere o código no método `HttpGet` `Edit` para que ele carregue a propriedade de navegação `OfficeAssignment` da entidade Instructor e chame `AsNoTracking`:

```

public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var instructor = await _context.Instructors
        .Include(i => i.OfficeAssignment)
        .AsNoTracking()
        .SingleOrDefaultAsync(m => m.ID == id);
    if (instructor == null)
    {
        return NotFound();
    }
    return View(instructor);
}

```

Substitua o método `HttpPost` `Edit` pelo seguinte código para manipular atualizações de atribuição de escritório:

```

[HttpPost, ActionName("Edit")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> EditPost(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var instructorToUpdate = await _context.Instructors
        .Include(i => i.OfficeAssignment)
        .SingleOrDefaultAsync(s => s.ID == id);

    if (await TryUpdateModelAsync<Instructor>(
        instructorToUpdate,
        "",
        i => i.FirstMidName, i => i.LastName, i => i.HireDate, i => i.OfficeAssignment))
    {
        if (String.IsNullOrWhiteSpace(instructorToUpdate.OfficeAssignment?.Location))
        {
            instructorToUpdate.OfficeAssignment = null;
        }
        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateException /* ex */)
        {
            //Log the error (uncomment ex variable name and write a log.)
            ModelState.AddModelError("", "Unable to save changes. " +
                "Try again, and if the problem persists, " +
                "see your system administrator.");
        }
        return RedirectToAction(nameof(Index));
    }
    return View(instructorToUpdate);
}

```

O código faz o seguinte:

- Altera o nome do método para `EditPost` porque a assinatura agora é a mesma do método `HttpGet` `Edit` (o atributo `ActionName` especifica que a URL `/Edit/` ainda é usada).
- Obtém a entidade `Instructor` atual do banco de dados usando o carregamento adiantado para a

propriedade de navegação `OfficeAssignment`. Isso é o mesmo que você fez no método `HttpGet Edit`.

- Atualiza a entidade `Instructor` recuperada com valores do associador de modelos. A sobrecarga `TryUpdateModel` permite que você adicione à lista de permissões as propriedades que você deseja incluir. Isso impede o excesso de postagem, conforme explicado no [segundo tutorial](#).

```
if (await TryUpdateModelAsync<Instructor>(
    instructorToUpdate,
    "",
    i => i.FirstMidName, i => i.LastName, i => i.HireDate, i => i.OfficeAssignment))
```

- Se o local do escritório estiver em branco, a propriedade `Instructor.OfficeAssignment` será definida como nula para que a linha relacionada na tabela `OfficeAssignment` seja excluída.

```
if (String.IsNullOrWhiteSpace(instructorToUpdate.OfficeAssignment?.Location))
{
    instructorToUpdate.OfficeAssignment = null;
}
```

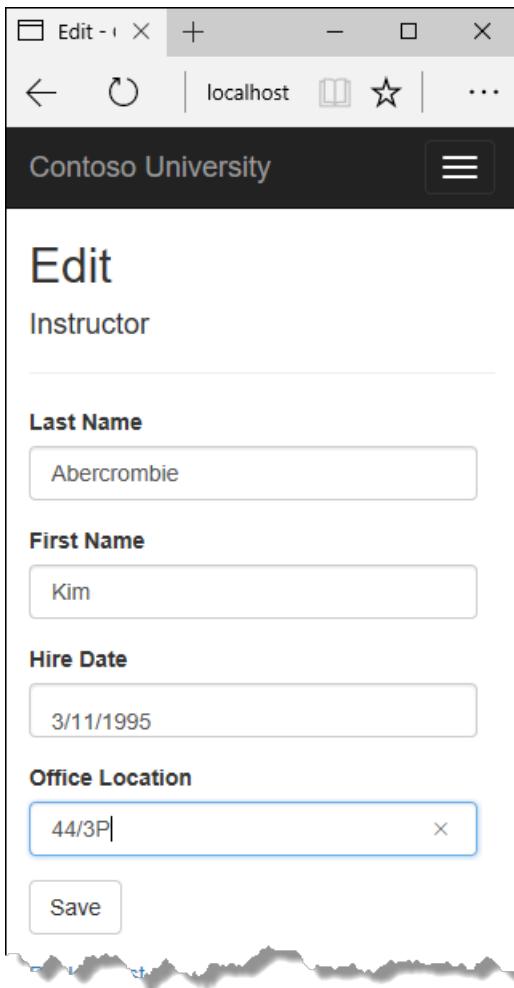
- Salva as alterações no banco de dados.

### Atualizar a exibição Editar Instrutor

Em `Views/Instructors/Edit.cshtml`, adicione um novo campo para editar o local do escritório, ao final, antes do botão **Salvar**:

```
<div class="form-group">
    <label asp-for="OfficeAssignment.Location" class="control-label"></label>
    <input asp-for="OfficeAssignment.Location" class="form-control" />
    <span asp-validation-for="OfficeAssignment.Location" class="text-danger" />
</div>
```

Execute o aplicativo, selecione a guia **Instrutores** e, em seguida, clique em **Editar** em um instrutor. Altere o **Local do Escritório** e clique em **Salvar**.



## Adicionar atribuições de Curso à página Editar Instrutor

Os instrutores podem ministrar a quantidade de cursos que desejarem. Agora, você aprimorará a página Editar Instrutor adicionando a capacidade de alterar as atribuições de curso usando um grupo de caixas de seleção, conforme mostrado na seguinte captura de tela:

Last Name  
Abercrombie

First Name  
Kim

Hire Date  
3/11/1995

Office Location  
44/3P

1000 Algebra 2    1045 Calculus    1050 Chemistry  
 2021 Composition    2042 Literature    3141 Trigonometry  
 4022 Microeconomics    4041 Macroeconomics

Save

A relação entre as entidades Course e Instructor é muitos para muitos. Para adicionar e remover relações, adicione e remova entidades bidirecionalmente no conjunto de entidades de junção CourseAssignments.

A interface do usuário que permite alterar a quais cursos um instrutor é atribuído é um grupo de caixas de seleção. Uma caixa de seleção é exibida para cada curso no banco de dados, e aqueles aos quais o instrutor está atribuído no momento são marcados. O usuário pode marcar ou desmarcar as caixas de seleção para alterar as atribuições de curso. Se a quantidade de cursos for muito maior, provavelmente, você desejará usar outro método de apresentação dos dados na exibição, mas usará o mesmo método de manipulação de uma entidade de junção para criar ou excluir relações.

#### Atualizar o controlador Instrutores

Para fornecer dados à exibição para a lista de caixas de seleção, você usará uma classe de modelo de exibição.

Crie *AssignedCourseData.cs* na pasta *SchoolViewModels* e substitua o código existente pelo seguinte código:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ContosoUniversity.Models.SchoolViewModels
{
    public class AssignedCourseData
    {
        public int CourseID { get; set; }
        public string Title { get; set; }
        public bool Assigned { get; set; }
    }
}

```

Em *InstructorsController.cs*, substitua o método `HttpGet` `Edit` pelo código a seguir. As alterações são realçadas.

```

public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var instructor = await _context.Instructors
        .Include(i => i.OfficeAssignment)
        .Include(i => i.CourseAssignments).ThenInclude(i => i.Course)
        .AsNoTracking()
        .SingleOrDefaultAsync(m => m.ID == id);
    if (instructor == null)
    {
        return NotFound();
    }
    PopulateAssignedCourseData(instructor);
    return View(instructor);
}

private void PopulateAssignedCourseData(Instructor instructor)
{
    var allCourses = _context.Courses;
    var instructorCourses = new HashSet<int>(instructor.CourseAssignments.Select(c => c.CourseID));
    var viewModel = new List<AssignedCourseData>();
    foreach (var course in allCourses)
    {
        viewModel.Add(new AssignedCourseData
        {
            CourseID = course.CourseID,
            Title = course.Title,
            Assigned = instructorCourses.Contains(course.CourseID)
        });
    }
    ViewData["Courses"] = viewModel;
}

```

O código adiciona o carregamento adianteado à propriedade de navegação `Courses` e chama o novo método `PopulateAssignedCourseData` para fornecer informações para a matriz de caixa de seleção usando a classe de modelo de exibição `AssignedCourseData`.

O código no método `PopulateAssignedCourseData` lê todas as entidades `Course` para carregar uma lista de cursos usando a classe de modelo de exibição. Para cada curso, o código verifica se o curso existe na propriedade de navegação `Courses` do instrutor. Para criar uma pesquisa eficiente ao verificar se um curso é atribuído ao instrutor, os cursos atribuídos ao instrutor são colocados em uma coleção `HashSet`. A propriedade `Assigned` está

definida como verdadeiro para os cursos aos quais instrutor é atribuído. A exibição usará essa propriedade para determinar quais caixas de seleção precisam ser exibidas como selecionadas. Por fim, a lista é passada para a exibição em `ViewData`.

Em seguida, adicione o código que é executado quando o usuário clica em **Salvar**. Substitua o método `EditPost` pelo código a seguir e adicione um novo método que atualiza a propriedade de navegação `Courses` da entidade `Instructor`.

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int? id, string[] selectedCourses)
{
    if (id == null)
    {
        return NotFound();
    }

    var instructorToUpdate = await _context.Instructors
        .Include(i => i.OfficeAssignment)
        .Include(i => i.CourseAssignments)
        .ThenInclude(i => i.Course)
        .SingleOrDefaultAsync(m => m.ID == id);

    if (await TryUpdateModelAsync<Instructor>(
        instructorToUpdate,
        "",
        i => i.FirstMidName, i => i.LastName, i => i.HireDate, i => i.OfficeAssignment))
    {
        if (String.IsNullOrWhiteSpace(instructorToUpdate.OfficeAssignment?.Location))
        {
            instructorToUpdate.OfficeAssignment = null;
        }
        UpdateInstructorCourses(selectedCourses, instructorToUpdate);
        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateException /* ex */)
        {
            //Log the error (uncomment ex variable name and write a log.)
            ModelState.AddModelError("", "Unable to save changes. " +
                "Try again, and if the problem persists, " +
                "see your system administrator.");
        }
        return RedirectToAction(nameof(Index));
    }
    UpdateInstructorCourses(selectedCourses, instructorToUpdate);
    PopulateAssignedCourseData(instructorToUpdate);
    return View(instructorToUpdate);
}
```

```

private void UpdateInstructorCourses(string[] selectedCourses, Instructor instructorToUpdate)
{
    if (selectedCourses == null)
    {
        instructorToUpdate.CourseAssignments = new List<CourseAssignment>();
        return;
    }

    var selectedCoursesHS = new HashSet<string>(selectedCourses);
    var instructorCourses = new HashSet<int>
        (instructorToUpdate.CourseAssignments.Select(c => c.Course.CourseID));
    foreach (var course in _context.Courses)
    {
        if (selectedCoursesHS.Contains(course.CourseID.ToString()))
        {
            if (!instructorCourses.Contains(course.CourseID))
            {
                instructorToUpdate.CourseAssignments.Add(new CourseAssignment { InstructorID =
instructorToUpdate.ID, CourseID = course.CourseID });
            }
        }
        else
        {

            if (instructorCourses.Contains(course.CourseID))
            {
                CourseAssignment courseToRemove = instructorToUpdate.CourseAssignments.SingleOrDefault(i =>
i.CourseID == course.CourseID);
                _context.Remove(courseToRemove);
            }
        }
    }
}

```

A assinatura do método agora é diferente do método `HttpGet Edit` e, portanto, o nome do método é alterado de `EditPost` para `Edit` novamente.

Como a exibição não tem uma coleção de entidades `Course`, o associador de modelos não pode atualizar automaticamente a propriedade de navegação `CourseAssignments`. Em vez de usar o associador de modelos para atualizar a propriedade de navegação `CourseAssignments`, faça isso no novo método `UpdateInstructorCourses`. Portanto, você precisa excluir a propriedade `CourseAssignments` do model binding. Isso não exige nenhuma alteração no código que chama `TryUpdateModel`, porque você está usando a sobrecarga da lista de permissões e `CourseAssignments` não está na lista de inclusões.

Se nenhuma caixa de seleção foi marcada, o código em `UpdateInstructorCourses` inicializa a propriedade de navegação `CourseAssignments` com uma coleção vazia e retorna:

```

private void UpdateInstructorCourses(string[] selectedCourses, Instructor instructorToUpdate)
{
    if (selectedCourses == null)
    {
        instructorToUpdate.CourseAssignments = new List<CourseAssignment>();
        return;
    }

    var selectedCoursesHS = new HashSet<string>(selectedCourses);
    var instructorCourses = new HashSet<int>
        (instructorToUpdate.CourseAssignments.Select(c => c.Course.CourseID));
    foreach (var course in _context.Courses)
    {
        if (selectedCoursesHS.Contains(course.CourseID.ToString()))
        {
            if (!instructorCourses.Contains(course.CourseID))
            {
                instructorToUpdate.CourseAssignments.Add(new CourseAssignment { InstructorID =
instructorToUpdate.ID, CourseID = course.CourseID });
            }
        }
        else
        {

            if (instructorCourses.Contains(course.CourseID))
            {
                CourseAssignment courseToRemove = instructorToUpdate.CourseAssignments.SingleOrDefault(i =>
i.CourseID == course.CourseID);
                _context.Remove(courseToRemove);
            }
        }
    }
}

```

Em seguida, o código executa um loop em todos os cursos no banco de dados e verifica cada curso em relação àqueles atribuídos no momento ao instrutor e em relação àqueles que foram selecionados na exibição. Para facilitar pesquisas eficientes, as últimas duas coleções são armazenadas em objetos `HashSet`.

Se a caixa de seleção para um curso foi marcada, mas o curso não está na propriedade de navegação `Instructor.CourseAssignments`, o curso é adicionado à coleção na propriedade de navegação.

```
private void UpdateInstructorCourses(string[] selectedCourses, Instructor instructorToUpdate)
{
    if (selectedCourses == null)
    {
        instructorToUpdate.CourseAssignments = new List<CourseAssignment>();
        return;
    }

    var selectedCoursesHS = new HashSet<string>(selectedCourses);
    var instructorCourses = new HashSet<int>
        (instructorToUpdate.CourseAssignments.Select(c => c.Course.CourseID));
    foreach (var course in _context.Courses)
    {
        if (selectedCoursesHS.Contains(course.CourseID.ToString()))
        {
            if (!instructorCourses.Contains(course.CourseID))
            {
                instructorToUpdate.CourseAssignments.Add(new CourseAssignment { InstructorID =
instructorToUpdate.ID, CourseID = course.CourseID });
            }
        }
        else
        {

            if (instructorCourses.Contains(course.CourseID))
            {
                CourseAssignment courseToRemove = instructorToUpdate.CourseAssignments.SingleOrDefault(i =>
i.CourseID == course.CourseID);
                _context.Remove(courseToRemove);
            }
        }
    }
}
```

Se a caixa de seleção para um curso não foi marcada, mas o curso está na propriedade de navegação `Instructor.CourseAssignments`, o curso é removido da propriedade de navegação.

```

private void UpdateInstructorCourses(string[] selectedCourses, Instructor instructorToUpdate)
{
    if (selectedCourses == null)
    {
        instructorToUpdate.CourseAssignments = new List<CourseAssignment>();
        return;
    }

    var selectedCoursesHS = new HashSet<string>(selectedCourses);
    var instructorCourses = new HashSet<int>
        (instructorToUpdate.CourseAssignments.Select(c => c.Course.CourseID));
    foreach (var course in _context.Courses)
    {
        if (selectedCoursesHS.Contains(course.CourseID.ToString()))
        {
            if (!instructorCourses.Contains(course.CourseID))
            {
                instructorToUpdate.CourseAssignments.Add(new CourseAssignment { InstructorID =
instructorToUpdate.ID, CourseID = course.CourseID });
            }
        }
        else
        {

            if (instructorCourses.Contains(course.CourseID))
            {
                CourseAssignment courseToRemove = instructorToUpdate.CourseAssignments.SingleOrDefault(i =>
i.CourseID == course.CourseID);
                _context.Remove(courseToRemove);
            }
        }
    }
}

```

## Atualizar as exibições Instrutor

Em *Views/Instructors/Edit.cshtml*, adicione um campo **Cursos** com uma matriz de caixas de seleção, adicionando o código a seguir imediatamente após os elementos `div` para o campo **Escritório** e antes do elemento `div` para o botão **Salvar**.

### NOTE

Quando você colar o código no Visual Studio, as quebras de linha serão alteradas de uma forma que divide o código. Pressione Ctrl+Z uma vez para desfazer a formatação automática. Isso corrigirá as quebras de linha para que elas se pareçam com o que você vê aqui. O recuo não precisa ser perfeito, mas cada uma das linhas `@</tr><tr>`, `@:<td>`, `@:</td>` e `@:</tr>` precisa estar em uma única linha, conforme mostrado, ou você receberá um erro de tempo de execução. Com o bloco de novo código selecionado, pressione Tab três vezes para alinhar o novo código com o código existente. Verifique o status deste problema [aqui](#).

```

<div class="form-group">
    <div class="col-md-offset-2 col-md-10">
        <table>
            <tr>
                @{
                    int cnt = 0;
                    List<ContosoUniversity.Models.SchoolViewModels.AssignedCourseData> courses =
                    ViewBag.Courses;

                    foreach (var course in courses)
                    {
                        if (cnt++ % 3 == 0)
                        {
                            @:</tr><tr>
                        }
                        @:<td>
                            <input type="checkbox"
                                name="selectedCourses"
                                value="@course.CourseID"
                                @(Html.Raw(course.Assigned ? "checked=\"checked\"" : ""))
                                @course.CourseID @: @course.Title
                            @:</td>
                        }
                        @:</tr>
                    }
                </table>
            </div>
        </div>

```

Esse código cria uma tabela HTML que contém três colunas. Em cada coluna há uma caixa de seleção, seguida de uma legenda que consiste no número e título do curso. Todas as caixas de seleção têm o mesmo nome ("selectedCourses"), o que informa ao associador de modelos de que elas devem ser tratadas como um grupo. O atributo de valor de cada caixa de seleção é definido com o valor de `CourseID`. Quando a página é postada, o associador de modelos passa uma matriz para o controlador que consiste nos valores `CourseID` para apenas as caixas de seleção marcadas.

Quando as caixas de seleção são inicialmente renderizadas, aquelas que se destinam aos cursos atribuídos ao instrutor têm atributos marcados, que os seleciona (exibe-os como marcados).

Execute o aplicativo, selecione a guia **Instrutores** e clique em **Editar** em um instrutor para ver a página **Editar**.

The screenshot shows a web browser window titled "Edit - Contoso Universit" with the URL "localhost:5813/Instruct". The main content is titled "Edit Instructor". It contains the following form fields:

- Last Name:** Abercrombie
- First Name:** Kim
- Hire Date:** 3/11/1995
- Office Location:** 44/3P
- Courses:** A list of checkboxes for courses:
  - 1000 Algebra 2
  - 1045 Calculus
  - 1050 Chemistry
  - 2021 Composition
  - 2042 Literature
  - 3141 Trigonometry
  - 4022 Microeconomics
  - 4041 Macroeconomics
- Save:** A button to save the changes.

Altere algumas atribuições de curso e clique em Salvar. As alterações feitas são refletidas na página Índice.

**NOTE**

A abordagem usada aqui para editar os dados de curso do instrutor funciona bem quando há uma quantidade limitada de cursos. Para coleções muito maiores, uma interface do usuário e um método de atualização diferentes são necessários.

## Atualizar a página Excluir

Em *InstructorsController.cs*, exclua o método `DeleteConfirmed` e insira o código a seguir em seu lugar.

```

[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    Instructor instructor = await _context.Instructors
        .Include(i => i.CourseAssignments)
        .SingleAsync(i => i.ID == id);

    var departments = await _context.Departments
        .Where(d => d.InstructorID == id)
        .ToListAsync();
    departments.ForEach(d => d.InstructorID = null);

    _context.Instructors.Remove(instructor);

    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

```

Este código faz as seguintes alterações:

- Executa o carregamento adiantado para a propriedade de navegação `CourseAssignments`. Você precisa incluir isso ou o EF não reconhecerá as entidades `CourseAssignment` relacionadas e não as excluirá. Para evitar a necessidade de lê-las aqui, você pode configurar a exclusão em cascata no banco de dados.
- Se o instrutor a ser excluído é atribuído como administrador de qualquer departamento, remove a atribuição de instrutor desse departamento.

## Adicionar local de escritório e cursos para a página Criar

Em `InstructorsController.cs`, exclua os métodos `HttpGet` e `HttpPost` `Create` e, em seguida, adicione o seguinte código em seu lugar:

```

public IActionResult Create()
{
    var instructor = new Instructor();
    instructor.CourseAssignments = new List<CourseAssignment>();
    PopulateAssignedCourseData(instructor);
    return View();
}

// POST: Instructors/Create
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("FirstMidName,HireDate,LastName,OfficeAssignment")] Instructor
instructor, string[] selectedCourses)
{
    if (selectedCourses != null)
    {
        instructor.CourseAssignments = new List<CourseAssignment>();
        foreach (var course in selectedCourses)
        {
            var courseToAdd = new CourseAssignment { InstructorID = instructor.ID, CourseID =
int.Parse(course) };
            instructor.CourseAssignments.Add(courseToAdd);
        }
    }
    if (ModelState.IsValid)
    {
        _context.Add(instructor);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    PopulateAssignedCourseData(instructor);
    return View(instructor);
}

```

Esse código é semelhante ao que você viu nos métodos `Edit`, exceto que inicialmente nenhum curso está selecionado. O método `HttpGet Create` chama o método `PopulateAssignedCourseData`, não porque pode haver cursos selecionados, mas para fornecer uma coleção vazia para o loop `foreach` na exibição (caso contrário, o código de exibição gera uma exceção de referência nula).

O método `HttpPost Create` adiciona cada curso selecionado à propriedade de navegação `CourseAssignments` antes que ele verifica se há erros de validação e adiciona o novo instrutor ao banco de dados. Os cursos são adicionados, mesmo se há erros de modelo, de modo que quando houver erros de modelo (por exemplo, o usuário inseriu uma data inválida) e a página for exibida novamente com uma mensagem de erro, as seleções de cursos que foram feitas sejam restauradas automaticamente.

Observe que para poder adicionar cursos à propriedade de navegação `CourseAssignments`, é necessário inicializar a propriedade como uma coleção vazia:

```
instructor.CourseAssignments = new List<CourseAssignment>();
```

Como alternativa a fazer isso no código do controlador, faça isso no modelo `Instructor` alterando o getter de propriedade para criar automaticamente a coleção se ela não existir, conforme mostrado no seguinte exemplo:

```

private ICollection<CourseAssignment> _courseAssignments;
public ICollection<CourseAssignment> CourseAssignments
{
    get
    {
        return _courseAssignments ?? (_courseAssignments = new List<CourseAssignment>());
    }
    set
    {
        _courseAssignments = value;
    }
}

```

Se você modificar a propriedade `CourseAssignments` dessa forma, poderá remover o código de inicialização de propriedade explícita no controlador.

Em `Views/Instructor/Create.cshtml`, adicione uma caixa de texto de local do escritório e caixas de seleção para cursos antes do botão Enviar. Como no caso da página Editar, [corrija a formatação se o Visual Studio reformatar o código quando você o colar](#).

```

<div class="form-group">
    <label asp-for="OfficeAssignment.Location" class="control-label"></label>
    <input asp-for="OfficeAssignment.Location" class="form-control" />
    <span asp-validation-for="OfficeAssignment.Location" class="text-danger" />
</div>

<div class="form-group">
    <div class="col-md-offset-2 col-md-10">
        <table>
            <tr>
                @{
                    int cnt = 0;
                    List<ContosoUniversity.Models.SchoolViewModels.AssignedCourseData> courses =
                    ViewBag.Courses;

                    foreach (var course in courses)
                    {
                        if (cnt++ % 3 == 0)
                        {
                            @:</tr><tr>
                        }
                        @:<td>
                            <input type="checkbox"
                                name="selectedCourses"
                                value="@course.CourseID"
                                @(Html.Raw(course.Assigned ? "checked=\"checked\"" : ""))
                                @course.CourseID @: @course.Title
                            @:</td>
                    }
                    @:</tr>
                }
            </table>
        </div>
    </div>

```

Faça o teste executando o aplicativo e criando um instrutor.

## Manipulando transações

Conforme explicado no [tutorial do CRUD](#), o Entity Framework implementa transações de forma implícita. Para cenários em que você precisa de mais controle – por exemplo, se desejar incluir operações feitas fora do Entity Framework em uma transação –, consulte [Transações](#).

## Resumo

Agora você concluiu a introdução ao trabalho com os dados relacionados. No próximo tutorial, você verá como lidar com conflitos de simultaneidade.

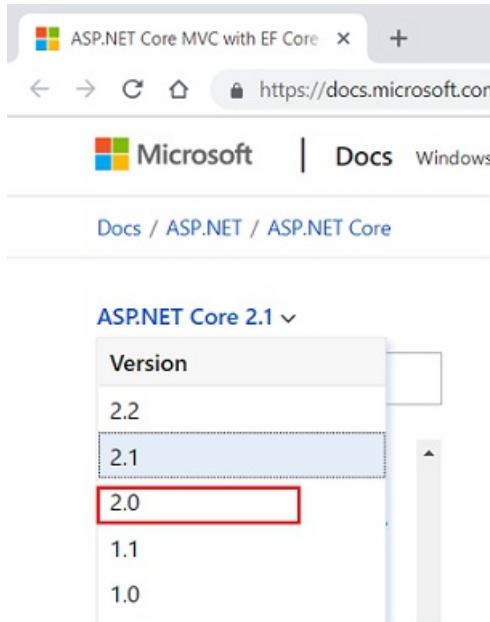
[ANTERIOR](#)

[PRÓXIMO](#)

# ASP.NET Core MVC com EF Core – simultaneidade – 8 de 10

30/10/2018 • 33 minutes to read • [Edit Online](#)

Este tutorial não foi atualizado para o ASP.NET Core 2.1. A versão deste tutorial para o ASP.NET Core 2.0 pode ser consultada selecionando **ASP.NET Core 2.0** acima do sumário ou na parte superior da página:



A screenshot of a web browser displaying the Microsoft Docs website for ASP.NET Core. The URL in the address bar is https://docs.microsoft.com. The page title is "ASP.NET Core MVC with EF Core". Below the title, there's a Microsoft logo and navigation links for "Docs" and "Windows". Under "Docs", the path "Docs / ASP.NET / ASP.NET Core" is shown. On the right side of the page, there's a dropdown menu titled "ASP.NET Core 2.1". The menu has a list of versions: 2.2, 2.1, 2.0, 1.1, and 1.0. The "2.0" option is highlighted with a red border.

A versão deste tutorial do [Razor Pages](#) para o ASP.NET Core 2.1 conta com várias melhorias em relação à versão 2.0.

O tutorial do 2.0 ensina a usar o ASP.NET Core MVC e o Entity Framework Core com controladores e exibições. O tutorial do Razor Pages foi atualizado com os seguintes aprimoramentos:

- É mais fácil de acompanhar. Por exemplo, o código de scaffolding foi bastante simplificado.
- Fornece mais práticas recomendadas do EF Core.
- Usa consultas mais eficientes.
- Usa a API do EF Core mais recente.

Por [Tom Dykstra](#) e [Rick Anderson](#)

O aplicativo web de exemplo Contoso University demonstra como criar aplicativos web do ASP.NET Core MVC usando o Entity Framework Core e o Visual Studio. Para obter informações sobre a série de tutoriais, consulte [o primeiro tutorial da série](#).

Nos tutoriais anteriores, você aprendeu a atualizar dados. Este tutorial mostra como lidar com conflitos quando os mesmos usuários atualizam a mesma entidade simultaneamente.

Você criará páginas da Web que funcionam com a entidade Department e tratará erros de simultaneidade. As ilustrações a seguir mostram as páginas Editar e Excluir, incluindo algumas mensagens exibidas se ocorre um conflito de simultaneidade.

Departmen X Edit - Cont X + - ×

← → ⌂ localhost:5813/Depatr ⚡ ⭐ ...

Contoso University Ⓛ

# Edit

## Department

The record you attempted to edit was modified by another user after you got the original value. The edit operation was canceled and the current values in the database have been displayed. If you still want to edit this record, click the Save button again. Otherwise click the Back to List hyperlink.

**Name**  
English

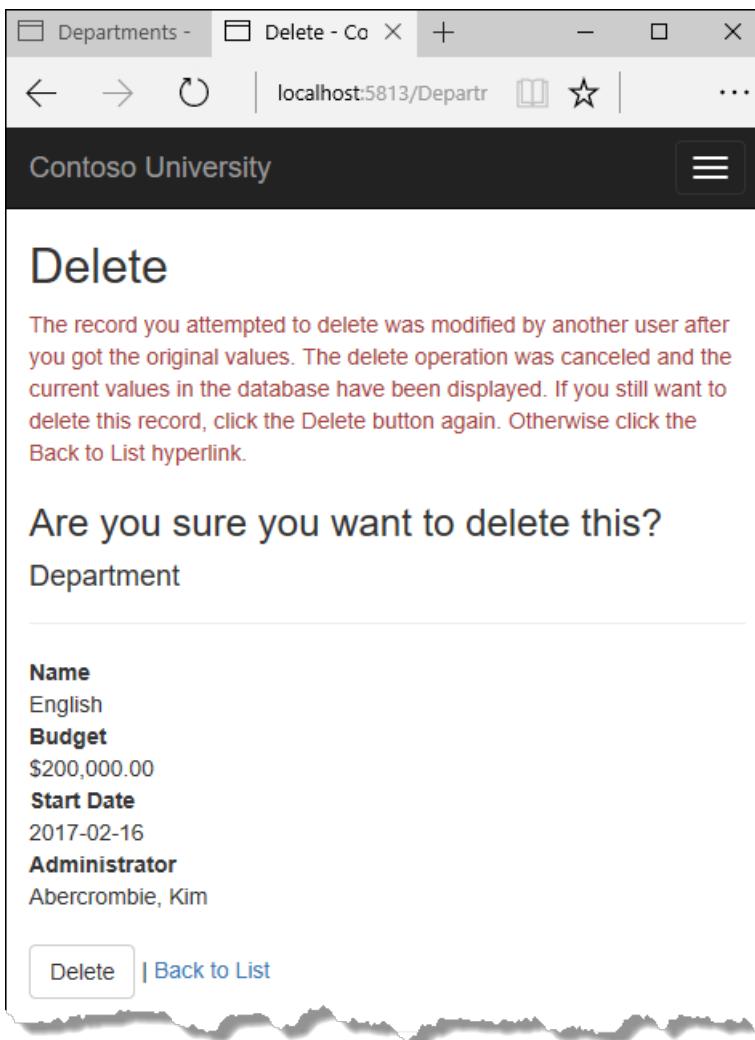
**Budget**  
200000.00  
Current value: \$50,000.00

**Start Date**  
9/1/2007

**InstructorID**  
Abercrombie, Kim ▾

Save





## Conflitos de simultaneidade

Um conflito de simultaneidade ocorre quando um usuário exibe dados de uma entidade para editá-los e, em seguida, outro usuário atualiza os mesmos dados da entidade antes que a primeira alteração do usuário seja gravada no banco de dados. Se você não habilitar a detecção desses conflitos, a última pessoa que atualizar o banco de dados substituirá as outras alterações do usuário. Em muitos aplicativos, esse risco é aceitável: se houver poucos usuários ou poucas atualizações ou se não for realmente crítico se algumas alterações forem substituídas, o custo de programação para simultaneidade poderá superar o benefício. Nesse caso, você não precisa configurar o aplicativo para lidar com conflitos de simultaneidade.

### Simultaneidade pessimista (bloqueio)

Se o aplicativo precisar evitar a perda accidental de dados em cenários de simultaneidade, uma maneira de fazer isso será usar bloqueios de banco de dados. Isso é chamado de simultaneidade pessimista. Por exemplo, antes de ler uma linha de um banco de dados, você solicita um bloqueio para o acesso somente leitura ou de atualização. Se você bloquear uma linha para o acesso de atualização, nenhum outro usuário terá permissão para bloquear a linha para o acesso somente leitura ou de atualização, porque ele obterá uma cópia dos dados que estão sendo alterados. Se você bloquear uma linha para o acesso somente leitura, outros também poderão bloqueá-la para o acesso somente leitura, mas não para atualização.

O gerenciamento de bloqueios traz desvantagens. Ele pode ser complexo de ser programado. Exige recursos de gerenciamento de banco de dados significativos e pode causar problemas de desempenho, conforme o número de usuários de um aplicativo aumenta. Por esses motivos, nem todos os sistemas de gerenciamento de banco de dados dão suporte à simultaneidade pessimista. O Entity Framework Core não fornece nenhum suporte interno para ele, e este tutorial não mostra como implementá-lo.

### Simultaneidade otimista

A alternativa à simultaneidade pessimista é a simultaneidade otimista. Simultaneidade otimista significa permitir que conflitos de simultaneidade ocorram e responder adequadamente se eles ocorrerem. Por exemplo, Alice visita a página Editar Departamento e altera o valor do Orçamento para o departamento de inglês de US\$ 350.000,00 para US\$ 0,00.

The screenshot shows a web browser window titled "Edit - Cont". The address bar displays "localhost:581". The main content area is titled "Edit" and "Department". It contains several form fields:

- Budget**: A text input field containing "0", which is highlighted with a red border.
- Administrator**: A dropdown menu showing "Abercrombie, Kim".
- Name**: A text input field containing "English".
- Start Date**: A text input field containing "9/1/2007".

A "Save" button is located at the bottom left of the form. The browser interface includes standard navigation buttons (back, forward, search, etc.) and a menu icon.

Antes que Alice clique em **Salvar**, Julio visita a mesma página e altera o campo Data de Início de 1/9/2007 para 1/9/2013.

A screenshot of a web browser window titled 'Edit - Contoso'. The address bar shows 'localhost:581'. The main content area has a dark header with 'Contoso University' and a menu icon. Below it, the word 'Edit' is displayed in large letters. Underneath, the word 'Department' is shown. There are several input fields: 'Budget' (350000.00), 'Administrator' (Abercrombie, Kim), 'Name' (English), and 'Start Date' (9/1/2013). A red box highlights the 'Start Date' input field. At the bottom right is a 'Save' button.

Alice clica em **Salvar** primeiro e vê a alteração quando o navegador retorna à página Índice.

A screenshot of a web browser window titled 'Departments - Contoso'. The address bar shows 'localhost:5813/Departments'. The main content area has a dark header with 'Contoso University'. Below it, the word 'Departments' is displayed in large letters. There is a 'Create New' link. A table lists departments with columns: Name, Budget, Administrator, and Start Date. The first row shows 'English' with '\$0.00', 'Abercrombie, Kim', and '2007-09-01'. To the right of the 'Start Date' column are 'Edit | Details | Delete' links. The 'Start Date' value is now '9/1/2013'.

Name	Budget	Administrator	Start Date	
English	\$0.00	Abercrombie, Kim	2007-09-01	Edit   Details   Delete
Mathematics	\$100000.00	Eekhout, Fadi	2007-09-01	Edit   Details   Delete

Em seguida, Julio clica em **Salvar** em uma página Editar que ainda mostra um orçamento de US\$ 350.000,00. O que acontece em seguida é determinado pela forma como você lida com conflitos de simultaneidade.

Algumas das opções incluem o seguinte:

- Controle qual propriedade um usuário modificou e atualize apenas as colunas correspondentes no banco de dados.

No cenário de exemplo, nenhum dado é perdido, porque propriedades diferentes foram atualizadas pelos dois usuários. Na próxima vez que alguém navegar pelo departamento de inglês, verá as alterações de Alice e de Julio – a data de início de 1/9/2013 e um orçamento de zero dólar. Esse método de atualização pode reduzir a quantidade de conflitos que podem resultar em perda de dados, mas ele não poderá evitar a

perda de dados se forem feitas alterações concorrentes à mesma propriedade de uma entidade. Se o Entity Framework funciona dessa maneira depende de como o código de atualização é implementado.

Geralmente, isso não é prático em um aplicativo Web, porque pode exigir que você mantenha grandes quantidades de estado para manter o controle de todos os valores de propriedade originais de uma entidade, bem como novos valores. A manutenção de grandes quantidades de estado pode afetar o desempenho do aplicativo, pois exige recursos do servidor ou deve ser incluída na própria página da Web (por exemplo, em campos ocultos) ou em um cookie.

- Você não pode deixar a alteração de Julio substituir a alteração de Alice.

Na próxima vez que alguém navegar pelo departamento de inglês, verá 1/9/2013 e o valor de US\$ 350.000,00 restaurado. Isso é chamado de um cenário *O cliente vence* ou *O último vence*. (Todos os valores do cliente têm precedência sobre o conteúdo do armazenamento de dados.) Conforme observado na introdução a esta seção, se você não fizer nenhuma codificação para a manipulação de simultaneidade, isso ocorrerá automaticamente.

- Você pode impedir que as alterações de Julio sejam atualizadas no banco de dados.

Normalmente, você exibirá uma mensagem de erro, mostrará a ele o estado atual dos dados e permitirá a ele aplicar as alterações novamente se ele ainda desejar fazê-las. Isso é chamado de um cenário *O armazenamento vence*. (Os valores do armazenamento de dados têm precedência sobre os valores enviados pelo cliente.) Você implementará o cenário O armazenamento vence neste tutorial. Esse método garante que nenhuma alteração é substituída sem que um usuário seja alertado sobre o que está acontecendo.

## Detectando conflitos de simultaneidade

Resolva conflitos manipulando exceções `DbConcurrencyException` geradas pelo Entity Framework. Para saber quando gerar essas exceções, o Entity Framework precisa poder detectar conflitos. Portanto, é necessário configurar o banco de dados e o modelo de dados de forma adequada. Algumas opções para habilitar a detecção de conflitos incluem as seguintes:

- Na tabela de banco de dados, inclua uma coluna de acompanhamento que pode ser usada para determinar quando uma linha é alterada. Em seguida, configure o Entity Framework para incluir essa coluna na cláusula Where de comandos SQL Update ou Delete.

O tipo de dados da coluna de acompanhamento é normalmente `rowversion`. O valor `rowversion` é um número sequencial que é incrementado sempre que a linha é atualizada. Em um comando Update ou Delete, a cláusula Where inclui o valor original da coluna de acompanhamento (a versão de linha original). Se a linha que está sendo atualizada tiver sido alterada por outro usuário, o valor da coluna `rowversion` será diferente do valor original; portanto, a instrução Update ou Delete não poderá encontrar a linha a ser atualizada devido à cláusula Where. Quando o Entity Framework descobre que nenhuma linha foi atualizada pelo comando Update ou Delete (ou seja, quando o número de linhas afetadas é zero), ele interpreta isso como um conflito de simultaneidade.

- Configure o Entity Framework para incluir os valores originais de cada coluna na tabela na cláusula Where de comandos Update e Delete.

Como a primeira opção, se nada na linha for alterado desde que a linha tiver sido lida pela primeira vez, a cláusula Where não retornará uma linha a ser atualizada, o que o Entity Framework interpretará como um conflito de simultaneidade. Para tabelas de banco de dados que têm muitas colunas, essa abordagem pode resultar em cláusulas Where muito grandes e pode exigir que você mantenha grandes quantidades de estado. Conforme observado anteriormente, manter grandes quantidades de estado pode afetar o desempenho do aplicativo. Portanto, essa abordagem geralmente não é recomendada e não é o método usado neste tutorial.

Caso deseje implementar essa abordagem, precisará marcar todas as propriedades de chave não primária

na entidade para as quais você deseja controlar a simultaneidade adicionando o atributo `ConcurrencyCheck` a elas. Essa alteração permite que o Entity Framework inclua todas as colunas na cláusula Where do SQL de instruções Update e Delete.

No restante deste tutorial, você adicionará uma propriedade de acompanhamento `rowversion` à entidade Department, criará um controlador e exibições e testará para verificar se tudo está funcionando corretamente.

## Adicionar uma propriedade de controle à entidade Department

Em `Models/Department.cs`, adicione uma propriedade de controle chamada RowVersion:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Department
    {
        public int DepartmentID { get; set; }

        [StringLength(50, MinimumLength = 3)]
        public string Name { get; set; }

        [DataType(DataType.Currency)]
        [Column(TypeName = "money")]
        public decimal Budget { get; set; }

        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        [Display(Name = "Start Date")]
        public DateTime StartDate { get; set; }

        public int? InstructorID { get; set; }

        [Timestamp]
        public byte[] RowVersion { get; set; }

        public Instructor Administrator { get; set; }
        public ICollection<Course> Courses { get; set; }
    }
}
```

O atributo `Timestamp` especifica que essa coluna será incluída na cláusula Where de comandos Update e Delete enviados ao banco de dados. O atributo é chamado `Timestamp` porque as versões anteriores do SQL Server usavam um tipo de dados `timestamp` do SQL antes de o `rowversion` do SQL substituí-lo. O tipo do .NET para `rowversion` é uma matriz de bytes.

Se preferir usar a API fluente, use o método `IsConcurrencyToken` (em `Data/SchoolContext.cs`) para especificar a propriedade de acompanhamento, conforme mostrado no seguinte exemplo:

```
modelBuilder.Entity<Department>()
    .Property(p => p.RowVersion).IsConcurrencyToken();
```

Adicionando uma propriedade, você alterou o modelo de banco de dados e, portanto, precisa fazer outra migração.

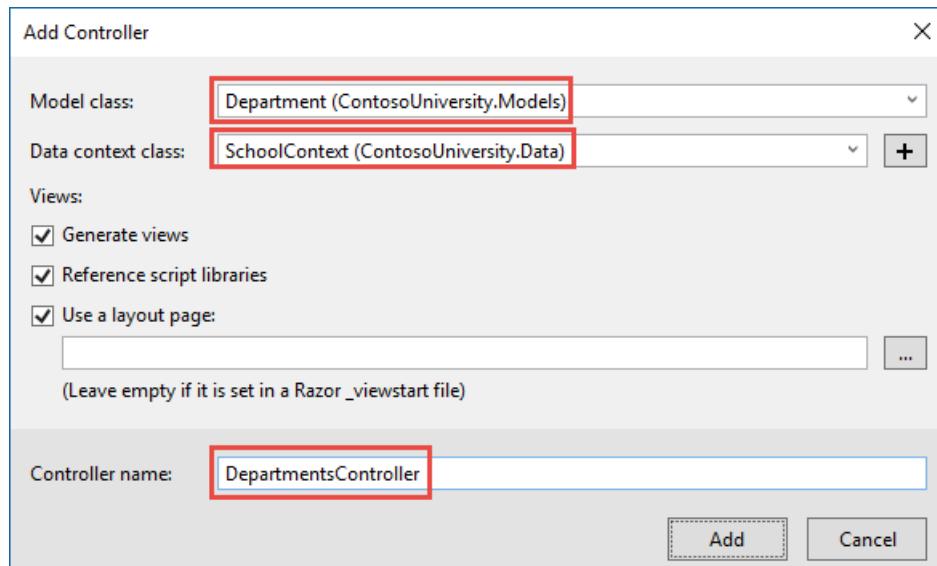
Salve as alterações e compile o projeto e, em seguida, insira os seguintes comandos na janela Comando:

```
dotnet ef migrations add RowVersion
```

```
dotnet ef database update
```

## Criar um controlador e exibições Departamentos

Gere por scaffolding um controlador e exibições Departamentos, como você fez anteriormente para Alunos, Cursos e Instrutores.



No arquivo *DepartmentsController.cs*, altere todas as quatro ocorrências de "FirstMidName" para "FullName", de modo que as listas suspensas do administrador do departamento contenham o nome completo do instrutor em vez de apenas o sobrenome.

```
ViewData["InstructorID"] = new SelectList(_context.Instructors, "ID", "FullName", department.InstructorID);
```

## Atualizar a exibição Índice de Departamentos

O mecanismo de scaffolding criou uma coluna RowVersion na exibição Índice, mas esse campo não deve ser exibido.

Substitua o código em *Views/Departments/Index.cshtml* pelo código a seguir.

```

@model IEnumerable<ContosoUniversity.Models.Department>

 @{
     ViewData["Title"] = "Departments";
 }

<h2>Departments</h2>

<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Name)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Budget)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.StartDate)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Administrator)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.Name)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Budget)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.StartDate)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Administrator.FullName)
                </td>
                <td>
                    <a asp-action="Edit" asp-route-id="@item.DepartmentID">Edit</a> |
                    <a asp-action="Details" asp-route-id="@item.DepartmentID">Details</a> |
                    <a asp-action="Delete" asp-route-id="@item.DepartmentID">Delete</a>
                </td>
            </tr>
        }
    </tbody>
</table>

```

Isso altera o título "Departamentos", exclui a coluna RowVersion e mostra o nome completo em vez de o nome do administrador.

## Atualizar os métodos Edit no controlador Departamentos

Nos métodos `HttpGet` `Edit` e `Details`, adicione `AsNoTracking`. No método `HttpGet` `Edit`, adicione o carregamento adianteado ao Administrador.

```

var department = await _context.Departments
    .Include(i => i.Administrator)
    .AsNoTracking()
    .SingleOrDefaultAsync(m => m.DepartmentID == id);

```

Substitua o código existente do método `HttpPost` [Edit](#) pelo seguinte código:

```

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int? id, byte[] rowVersion)
{
    if (id == null)
    {
        return NotFound();
    }

    var departmentToUpdate = await _context.Departments.Include(i => i.Administrator).SingleOrDefaultAsync(m => m.DepartmentID == id);

    if (departmentToUpdate == null)
    {
        Department deletedDepartment = new Department();
        await TryUpdateModelAsync(deletedDepartment);
        ModelState.AddModelError(string.Empty,
            "Unable to save changes. The department was deleted by another user.");
        ViewData["InstructorID"] = new SelectList(_context.Instructors, "ID", "FullName",
deletedDepartment.InstructorID);
        return View(deletedDepartment);
    }

    _context.Entry(departmentToUpdate).Property("RowVersion").OriginalValue = rowVersion;

    if (await TryUpdateModelAsync<Department>(
        departmentToUpdate,
        "",
        s => s.Name, s => s.StartDate, s => s.Budget, s => s.InstructorID))
    {
        try
        {
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        catch (DbUpdateConcurrencyException ex)
        {
            var exceptionEntry = ex.Entries.Single();
            var clientValues = (Department)exceptionEntry.Entity;
            var databaseEntry = exceptionEntry.GetDatabaseValues();
            if (databaseEntry == null)
            {
                ModelState.AddModelError(string.Empty,
                    "Unable to save changes. The department was deleted by another user.");
            }
            else
            {
                var databaseValues = (Department)databaseEntry.ToObject();

                if (databaseValues.Name != clientValues.Name)
                {
                    ModelState.AddModelError("Name", $"Current value: {databaseValues.Name}");
                }
                if (databaseValues.Budget != clientValues.Budget)
                {
                    ModelState.AddModelError("Budget", $"Current value: {databaseValues.Budget:c}");
                }
                if (databaseValues.StartDate != clientValues.StartDate)
                {

```

```

        ModelState.AddModelError("StartDate", $"Current value: {databaseValues.StartDate:d}");
    }
    if (databaseValues.InstructorID != clientValues.InstructorID)
    {
        Instructor databaseInstructor = await _context.Instructors.SingleOrDefaultAsync(i => i.ID
== databaseValues.InstructorID);
        ModelState.AddModelError("InstructorID", $"Current value:
{databaseInstructor?.FullName}");
    }

    ModelState.AddModelError(string.Empty, "The record you attempted to edit "
        + "was modified by another user after you got the original value. The "
        + "edit operation was canceled and the current values in the database "
        + "have been displayed. If you still want to edit this record, click "
        + "the Save button again. Otherwise click the Back to List hyperlink.");
    departmentToUpdate.RowVersion = (byte[])databaseValues.RowVersion;
    ModelState.Remove("RowVersion");
}
}
}
ViewData["InstructorID"] = new SelectList(_context.Instructors, "ID", "FullName",
departmentToUpdate.InstructorID);
return View(departmentToUpdate);
}

```

O código começa com a tentativa de ler o departamento a ser atualizado. Se o método `SingleOrDefaultAsync` retornar nulo, isso indicará que o departamento foi excluído por outro usuário. Nesse caso, o código usa os valores de formulário postados para criar uma entidade de departamento, de modo que a página Editar possa ser exibida novamente com uma mensagem de erro. Como alternativa, você não precisará recriar a entidade de departamento se exibir apenas uma mensagem de erro sem exibir novamente os campos de departamento.

A exibição armazena o valor `RowVersion` original em um campo oculto e esse método recebe esse valor no parâmetro `rowVersion`. Antes de chamar `SaveChanges`, você precisa colocar isso no valor da propriedade `RowVersion` original na coleção `OriginalValues` da entidade.

```
_context.Entry(departmentToUpdate).Property("RowVersion").OriginalValue = rowVersion;
```

Em seguida, quando o Entity Framework criar um comando SQL UPDATE, esse comando incluirá uma cláusula WHERE que procura uma linha que tem o valor `RowVersion` original. Se nenhuma linha for afetada pelo comando UPDATE (nenhuma linha tem o valor `RowVersion` original), o Entity Framework gerará uma exceção `DbUpdateConcurrencyException`.

O código no bloco catch dessa exceção obtém a entidade Department afetada que tem os valores atualizados da propriedade `Entries` no objeto de exceção.

```
var exceptionEntry = ex.Entries.Single();
```

A coleção `Entries` terá apenas um objeto `EntityEntry`. Use esse objeto para obter os novos valores inseridos pelo usuário e os valores de banco de dados atuais.

```
var clientValues = (Department)exceptionEntry.Entity;
var databaseEntry = exceptionEntry.GetDatabaseValues();
```

O código adiciona uma mensagem de erro personalizada a cada coluna que tem valores de banco de dados diferentes do que o usuário inseriu na página Editar (apenas um campo é mostrado aqui para fins de brevidade).

```
var databaseValues = (Department)databaseEntry.ToObject();

if (databaseValues.Name != clientValues.Name)
{
    ModelState.AddModelError("Name", $"Current value: {databaseValues.Name}");
}
```

Por fim, o código define o valor `RowVersion` do `departmentToUpdate` com o novo valor recuperado do banco de dados. Esse novo valor `RowVersion` será armazenado no campo oculto quando a página Editar for exibida novamente, e na próxima vez que o usuário clicar em **Salvar**, somente os erros de simultaneidade que ocorrem desde a nova exibição da página Editar serão capturados.

```
departmentToUpdate.RowVersion = (byte[])databaseValues.RowVersion;
ModelState.Remove("RowVersion");
```

A instrução `ModelState.Remove` é obrigatória porque `ModelState` tem o valor `RowVersion` antigo. Na exibição, o valor `ModelState` de um campo tem precedência sobre os valores de propriedade do modelo, quando ambos estão presentes.

## Atualizar a exibição Editar Departamento

Em `Views/Departments/Edit.cshtml`, faça as seguintes alterações:

- Adicione um campo oculto para salvar o valor da propriedade `RowVersion`, imediatamente após o campo oculto da propriedade `DepartmentID`.
- Adicione uma opção "Selecionar Administrador" à lista suspensa.

```

@model ContosoUniversity.Models.Department

 @{
     ViewData["Title"] = "Edit";
 }

<h2>Edit</h2>

<h4>Department</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Edit">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <input type="hidden" asp-for="DepartmentID" />
            <input type="hidden" asp-for="RowVersion" />
            <div class="form-group">
                <label asp-for="Name" class="control-label"></label>
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Budget" class="control-label"></label>
                <input asp-for="Budget" class="form-control" />
                <span asp-validation-for="Budget" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="StartDate" class="control-label"></label>
                <input asp-for="StartDate" class="form-control" />
                <span asp-validation-for="StartDate" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="InstructorID" class="control-label"></label>
                <select asp-for="InstructorID" class="form-control" asp-items="ViewBag.InstructorID">
                    <option value="">-- Select Administrator --</option>
                </select>
                <span asp-validation-for="InstructorID" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Save" class="btn btn-default" />
            </div>
        </form>
    </div>
</div>

<div>
    <a asp-action="Index">Back to List</a>
</div>

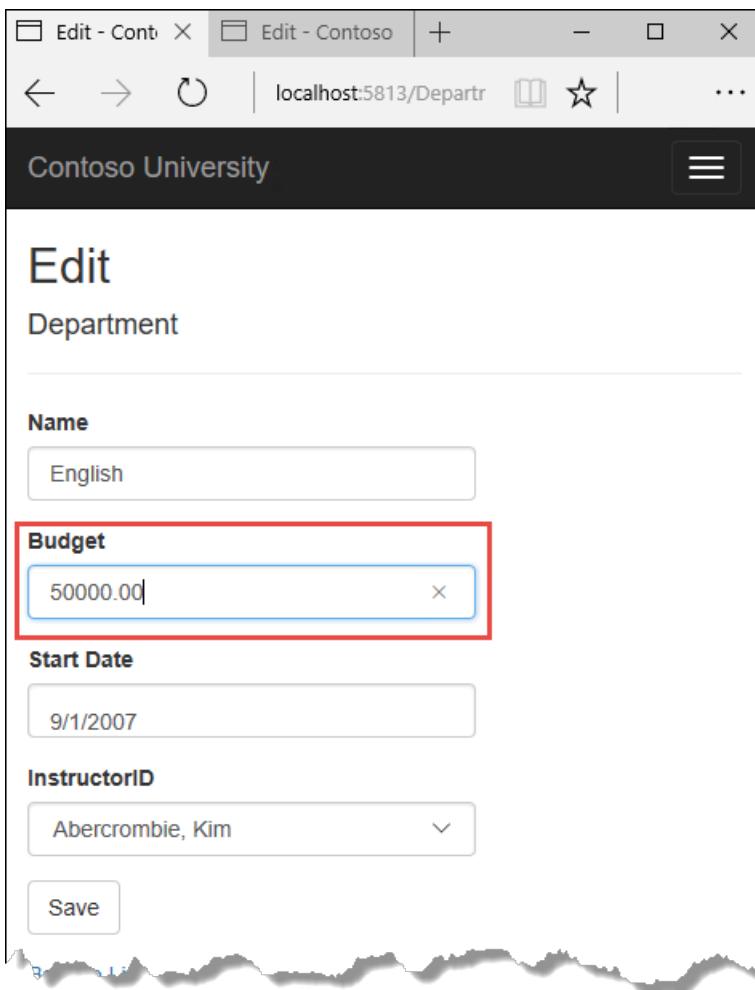
@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}

```

## Testar conflitos de simultaneidade na página Editar

Execute o aplicativo e acesse a página Índice de Departamentos. Clique com o botão direito do mouse na hiperlink **Editar** do departamento de inglês e selecione **Abrir em uma nova guia** e, em seguida, clique no hiperlink **Editar** no departamento de inglês. As duas guias do navegador agora exibem as mesmas informações.

Altere um campo na primeira guia do navegador e clique em **Salvar**.



The screenshot shows a web browser window with two tabs: 'Edit - Contoso' and 'Edit - Contoso'. The active tab displays the URL 'localhost:5813/Departments'. The page title is 'Contoso University'. The main content is an 'Edit' form for a 'Department'. The form fields include:

- Name:** English
- Budget:** 50000.00 (This field is highlighted with a red border)
- Start Date:** 9/1/2007
- InstructorID:** Abercrombie, Kim

A 'Save' button is located at the bottom left of the form.

O navegador mostra a página Índice com o valor alterado.

Altere um campo na segunda guia do navegador.

Departments - Edit - Cont X + - ×

localhost:5813/Departr

Contoso University

# Edit

## Department

Name

Budget

 x

Start Date

InstructorID

 ▼

Save

Clique em **Salvar**. Você verá uma mensagem de erro:

The record you attempted to edit was modified by another user after you got the original value. The edit operation was canceled and the current values in the database have been displayed. If you still want to edit this record, click the Save button again. Otherwise click the Back to List hyperlink.

**Name**  
English

**Budget**  
200000.00  
Current value: \$50,000.00

**Start Date**  
9/1/2007

**InstructorID**  
Abercrombie, Kim

**Save**

Clique em **Salvar** novamente. O valor inserido na segunda guia do navegador foi salvo. Você verá os valores salvos quando a página Índice for exibida.

## Atualizar a página Excluir

Para a página Excluir, o Entity Framework detecta conflitos de simultaneidade causados pela edição por outra pessoa do departamento de maneira semelhante. Quando o método `HttpGet Delete` exibe a exibição de confirmação, a exibição inclui o valor `RowVersion` original em um campo oculto. Em seguida, esse valor estará disponível para o método `HttpPost Delete` chamado quando o usuário confirmar a exclusão. Quando o Entity Framework cria o comando SQL `DELETE`, ele inclui uma cláusula `WHERE` com o valor `RowVersion` original. Se o comando não resultar em nenhuma linha afetada (o que significa que a linha foi alterada após a exibição da página Confirmação de exclusão), uma exceção de simultaneidade será gerada e o método `HttpGet Delete` será chamado com um sinalizador de erro definido como verdadeiro para exibir novamente a página de confirmação com uma mensagem de erro. Também é possível que nenhuma linha tenha sido afetada porque a linha foi excluída por outro usuário; portanto, nesse caso, nenhuma mensagem de erro é exibida.

### Atualizar os métodos Excluir no controlador Departamentos

Em `DepartmentsController.cs`, substitua o método `HttpGet Delete` pelo seguinte código:

```

public async Task<IActionResult> Delete(int? id, bool? concurrencyError)
{
    if (id == null)
    {
        return NotFound();
    }

    var department = await _context.Departments
        .Include(d => d.Administrator)
        .AsNoTracking()
        .SingleOrDefaultAsync(m => m.DepartmentID == id);
    if (department == null)
    {
        if (concurrencyError.GetValueOrDefault())
        {
            return RedirectToAction(nameof(Index));
        }
        return NotFound();
    }

    if (concurrencyError.GetValueOrDefault())
    {
        ViewData["ConcurrencyErrorMessage"] = "The record you attempted to delete "
            + "was modified by another user after you got the original values. "
            + "The delete operation was canceled and the current values in the "
            + "database have been displayed. If you still want to delete this "
            + "record, click the Delete button again. Otherwise "
            + "click the Back to List hyperlink.";
    }

    return View(department);
}

```

O método aceita um parâmetro opcional que indica se a página está sendo exibida novamente após um erro de simultaneidade. Se esse sinalizador é verdadeiro e o departamento especificado não existe mais, isso indica que ele foi excluído por outro usuário. Nesse caso, o código redireciona para a página Índice. Se esse sinalizador é verdadeiro e o departamento existe, isso indica que ele foi alterado por outro usuário. Nesse caso, o código envia uma mensagem de erro para a exibição usando `ViewData`.

Substitua o código no método `HttpPost` `Delete` (chamado `DeleteConfirmed`) pelo seguinte código:

```

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Delete(Department department)
{
    try
    {
        if (await _context.Departments.AnyAsync(m => m.DepartmentID == department.DepartmentID))
        {
            _context.Departments.Remove(department);
            await _context.SaveChangesAsync();
        }
        return RedirectToAction(nameof(Index));
    }
    catch (DbUpdateConcurrencyException /* ex */)
    {
        //Log the error (uncomment ex variable name and write a log.)
        return RedirectToAction(nameof(Delete), new { concurrencyError = true, id = department.DepartmentID });
    }
}

```

No código gerado por scaffolding que acabou de ser substituído, esse método aceitou apenas uma ID de registro:

```
public async Task<IActionResult> DeleteConfirmed(int id)
```

Você alterou esse parâmetro para uma instância da entidade Departamento criada pelo associador de modelos. Isso concede o acesso ao EF ao valor da propriedade RowVersion, além da chave de registro.

```
public async Task<IActionResult> Delete(Department department)
```

Você também alterou o nome do método de ação de `DeleteConfirmed` para `Delete`. O código gerado por scaffolding usou o nome `DeleteConfirmed` para fornecer ao método `HttpPost` uma assinatura exclusiva. (O CLR exige que métodos sobrecarregados tenham diferentes parâmetros de método.) Agora que as assinaturas são exclusivas, você pode continuar com a convenção MVC e usar o mesmo nome para os métodos de exclusão `HttpPost` e `HttpGet`.

Se o departamento já foi excluído, o método `AnyAsync` retorna falso e o aplicativo apenas volta para o método de Índice.

Se um erro de simultaneidade é capturado, o código exibe novamente a página Confirmação de exclusão e fornece um sinalizador que indica que ela deve exibir uma mensagem de erro de simultaneidade.

### Atualizar a exibição Excluir

Em `Views/Departments/Delete.cshtml`, substitua o código gerado por scaffolding pelo seguinte código que adiciona um campo de mensagem de erro e campos ocultos às propriedades `DepartmentID` e `RowVersion`. As alterações são realçadas.

```

@model ContosoUniversity.Models.Department

 @{
     ViewData["Title"] = "Delete";
 }

<h2>Delete</h2>

<p class="text-danger">@ViewData["ConcurrencyErrorMessage"]</p>

<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>Department</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Name)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Name)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Budget)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Budget)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.StartDate)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.StartDate)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Administrator)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Administrator.FullName)
        </dd>
    </dl>

    <form asp-action="Delete">
        <input type="hidden" asp-for="DepartmentID" />
        <input type="hidden" asp-for="RowVersion" />
        <div class="form-actions no-color">
            <input type="submit" value="Delete" class="btn btn-default" /> |
            <a asp-action="Index">Back to List</a>
        </div>
    </form>
</div>

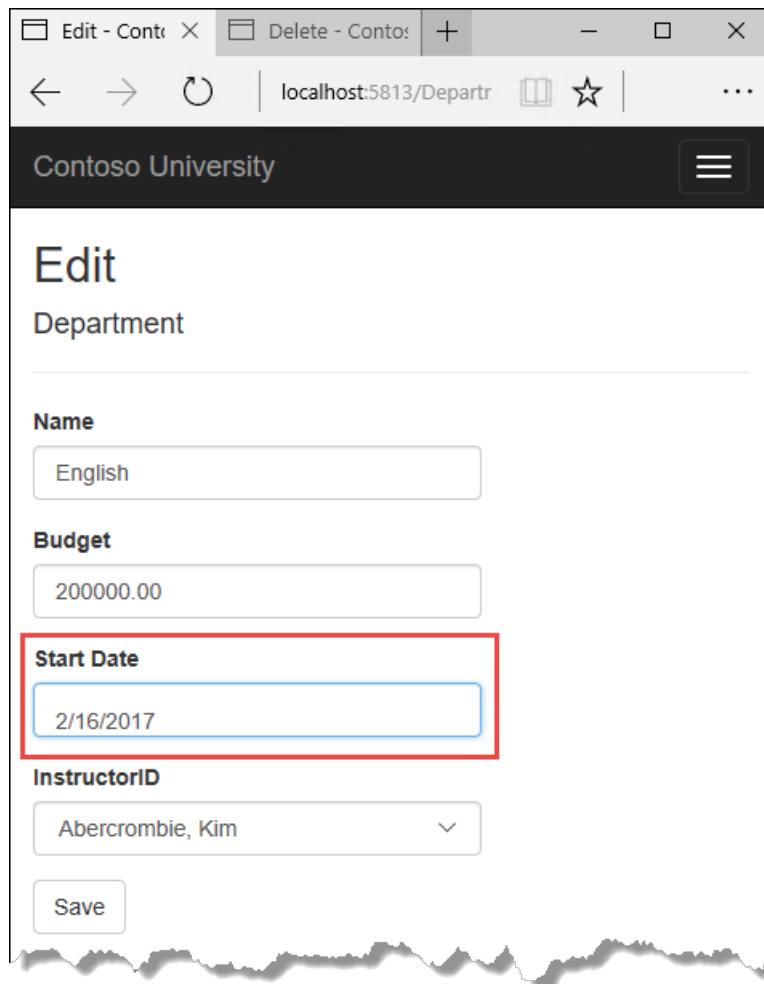
```

Isso faz as seguintes alterações:

- Adiciona uma mensagem de erro entre os cabeçalhos `h2` e `h3`.
- Substitua `FirstMidName` por `FullName` no campo **Administrador**.
- Remove o campo `RowVersion`.
- Adiciona um campo oculto à propriedade `RowVersion`.

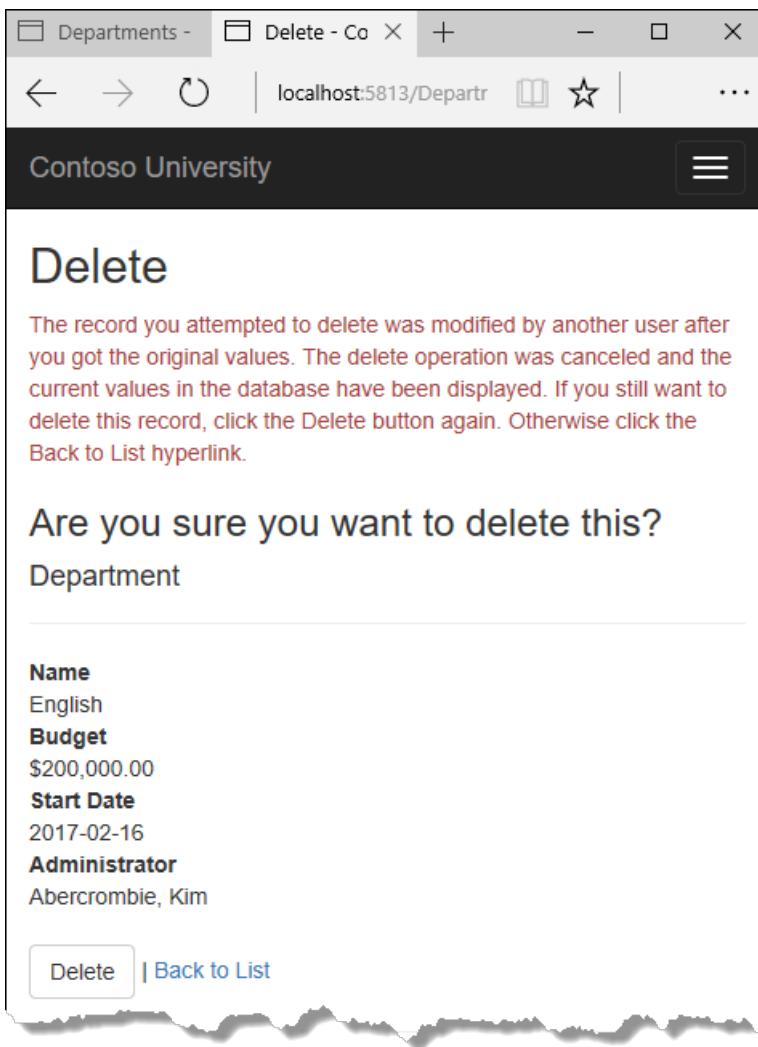
Execute o aplicativo e acesse a página Índice de Departamentos. Clique com o botão direito do mouse na hiperlink **Excluir** do departamento de inglês e selecione **Abrir em uma nova guia** e, em seguida, na primeira guia, clique no hiperlink **Editar** no departamento de inglês.

Na primeira janela, altere um dos valores e clique em **Salvar**:



The screenshot shows a web browser window with the address bar displaying "localhost:5813/Departments". The main content area is titled "Edit" and "Department". It contains several input fields: "Name" (value: English), "Budget" (value: 200000.00), "Start Date" (value: 2/16/2017, highlighted with a red border), "InstructorID" (dropdown menu showing "Abercrombie, Kim"), and a "Save" button.

Na segunda guia, clique em **Excluir**. Você verá a mensagem de erro de simultaneidade e os valores de Departamento serão atualizados com o que está atualmente no banco de dados.



Se você clicar em **Excluir** novamente, será redirecionado para a página Índice, que mostra que o departamento foi excluído.

## Exibições Atualizar Detalhes e Criar

Opcionalmente, você pode limpar o código gerado por scaffolding nas exibições Detalhes e Criar.

Substitua o código em *Views/Departments/Details.cshtml* para excluir a coluna RowVersion e mostrar o nome completo do Administrador.

```

@model ContosoUniversity.Models.Department

 @{
    ViewData["Title"] = "Details";
}

<h2>Details</h2>

<div>
    <h4>Department</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Name)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Name)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Budget)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Budget)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.StartDate)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.StartDate)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Administrator)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Administrator.FullName)
        </dd>
    </dl>
</div>
<div>
    <a asp-action="Edit" asp-route-id="@Model.DepartmentID">Edit</a> |
    <a asp-action="Index">Back to List</a>
</div>

```

Substitua o código em *Views/Departments/Create.cshtml* para adicionar uma opção Selecionar à lista suspensa.

```

@model ContosoUniversity.Models.Department

 @{
     ViewData["Title"] = "Create";
 }

<h2>Create</h2>

<h4>Department</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Name" class="control-label"></label>
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Budget" class="control-label"></label>
                <input asp-for="Budget" class="form-control" />
                <span asp-validation-for="Budget" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="StartDate" class="control-label"></label>
                <input asp-for="StartDate" class="form-control" />
                <span asp-validation-for="StartDate" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="InstructorID" class="control-label"></label>
                <select asp-for="InstructorID" class="form-control" asp-items="ViewBag.InstructorID">
                    <option value="">-- Select Administrator --</option>
                </select>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-default" />
            </div>
        </form>
    </div>
</div>
<div>
    <a asp-action="Index">Back to List</a>
</div>

@section Scripts {
    @await Html.RenderPartialAsync("_ValidationScriptsPartial");
}

```

## Resumo

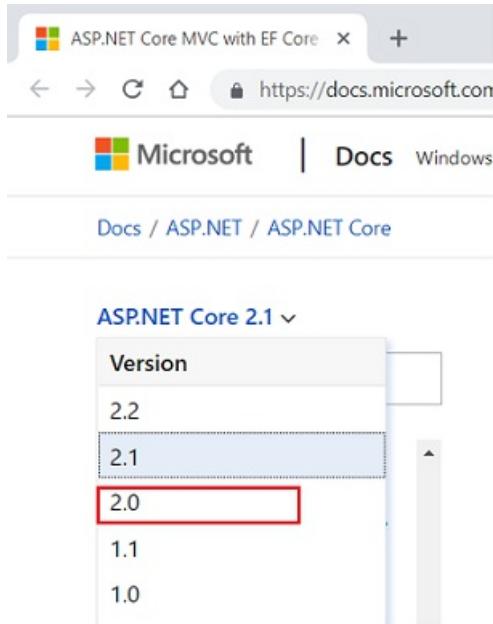
Isso conclui a introdução à manipulação de conflitos de simultaneidade. Para obter mais informações sobre como lidar com a simultaneidade no EF Core, consulte [Conflitos de simultaneidade](#). O próximo tutorial mostra como implementar a herança de tabela por hierarquia para as entidades Instructor e Student.

[ANTERIOR](#)
[PRÓXIMO](#)

# ASP.NET Core MVC com EF Core – herança – 9 de 10

30/10/2018 • 15 minutes to read • [Edit Online](#)

Este tutorial não foi atualizado para o ASP.NET Core 2.1. A versão deste tutorial para o ASP.NET Core 2.0 pode ser consultada selecionando **ASP.NET Core 2.0** acima do sumário ou na parte superior da página:



A versão deste tutorial do [Razor Pages](#) para o ASP.NET Core 2.1 conta com várias melhorias em relação à versão 2.0.

O tutorial do 2.0 ensina a usar o ASP.NET Core MVC e o Entity Framework Core com controladores e exibições. O tutorial do Razor Pages foi atualizado com os seguintes aprimoramentos:

- É mais fácil de acompanhar. Por exemplo, o código de scaffolding foi bastante simplificado.
- Fornece mais práticas recomendadas do EF Core.
- Usa consultas mais eficientes.
- Usa a API do EF Core mais recente.

Por [Tom Dykstra](#) e [Rick Anderson](#)

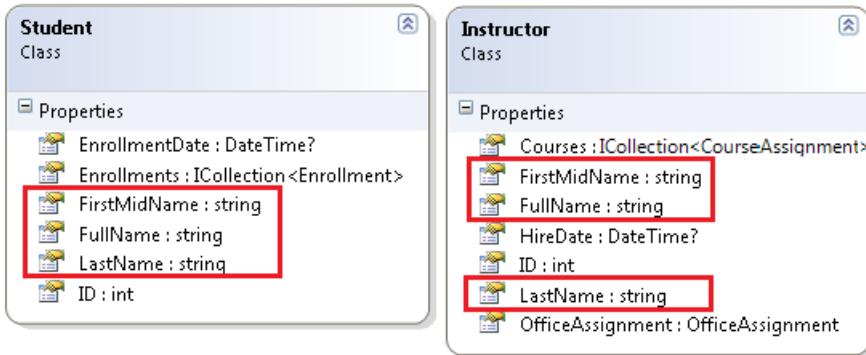
O aplicativo web de exemplo Contoso University demonstra como criar aplicativos web do ASP.NET Core MVC usando o Entity Framework Core e o Visual Studio. Para obter informações sobre a série de tutoriais, consulte [primeiro tutorial na série](#).

No tutorial anterior, você tratou exceções de simultaneidade. Este tutorial mostrará como implementar a herança no modelo de dados.

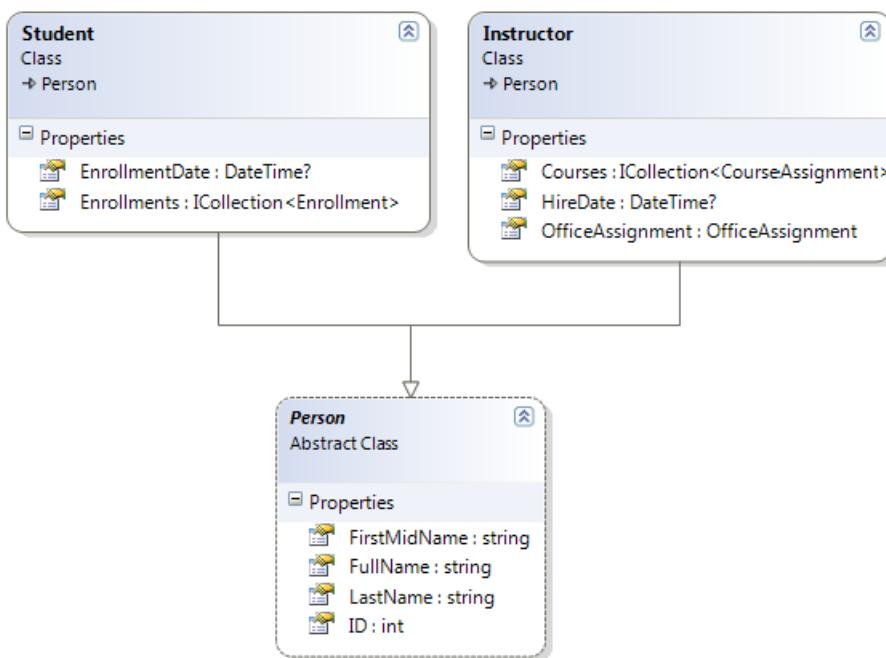
Na programação orientada a objeto, você pode usar a herança para facilitar a reutilização de código. Neste tutorial, você alterará as classes `Instructor` e `Student`, de modo que elas derivem de uma classe base `Person` que contém propriedades, como `LastName`, comuns a instrutores e alunos. Você não adicionará nem alterará as páginas da Web, mas alterará uma parte do código, e essas alterações serão refletidas automaticamente no banco de dados.

# Opções para o mapeamento de herança para as tabelas de banco de dados

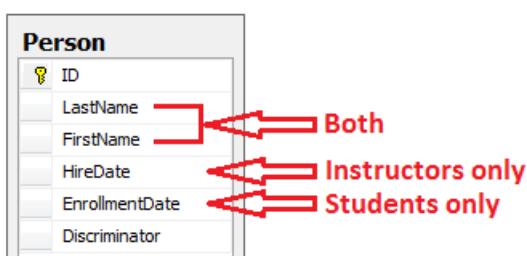
As classes `Instructor` e `Student` no modelo de dados Escola têm várias propriedades idênticas:



Suponha que você deseja eliminar o código redundante para as propriedades compartilhadas pelas entidades `Instructor` e `Student`. Ou você deseja escrever um serviço que pode formatar nomes sem se preocupar se o nome foi obtido de um instrutor ou um aluno. Você pode criar uma classe base `Person` que contém apenas essas propriedades compartilhadas e, em seguida, fazer com que as classes `Instructor` e `Student` herdem dessa classe base, conforme mostrado na seguinte ilustração:

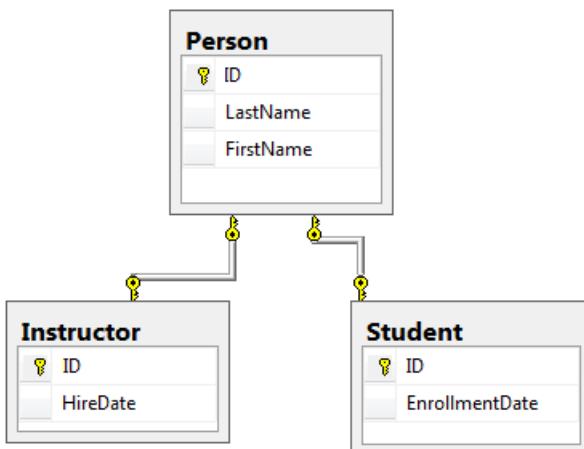


Há várias maneiras pelas quais essa estrutura de herança pode ser representada no banco de dados. Você pode ter uma tabela `Person` que inclui informações sobre alunos e instrutores em uma única tabela. Algumas das colunas podem se aplicar somente a instrutores (`HireDate`), algumas somente a alunos (`EnrollmentDate`) e outras a ambos (`LastName`, `FirstName`). Normalmente, você terá uma coluna discriminatória para indicar qual tipo cada linha representa. Por exemplo, a coluna discriminatória pode ter "Instrutor" para instrutores e "Aluno" para alunos.



Esse padrão de geração de uma estrutura de herança de entidade com base em uma tabela de banco de dados individual é chamado de herança TPH (tabela por hierarquia).

Uma alternativa é fazer com que o banco de dados se pareça mais com a estrutura de herança. Por exemplo, você pode ter apenas os campos de nome na tabela Pessoa e as tabelas separadas Instrutor e Aluno com os campos de data.



Esse padrão de criação de uma tabela de banco de dados para cada classe de entidade é chamado de herança TPT (tabela por tipo).

Outra opção é mapear todos os tipos não abstratos para tabelas individuais. Todas as propriedades de uma classe, incluindo propriedades herdadas, são mapeadas para colunas da tabela correspondente. Esse padrão é chamado de herança TPC (Tabela por Classe Concreta). Se você tiver implementado a herança TPC para as classes Person, Student e Instructor, conforme mostrado anteriormente, as tabelas Aluno e Instrutor não parecerão diferentes após a implementação da herança do que antes.

Em geral, os padrões de herança TPC e TPH oferecem melhor desempenho do que os padrões de herança TPT, porque os padrões TPT podem resultar em consultas de junção complexas.

Este tutorial demonstra como implementar a herança TPH. TPH é o único padrão de herança compatível com o Entity Framework Core. O que você fará é criar uma classe `Person`, alterar as classes `Instructor` e `Student` para que elas derivem de `Person`, adicionar a nova classe ao `DbContext` e criar uma migração.

#### TIP

Considere a possibilidade de salvar uma cópia do projeto antes de fazer as alterações a seguir. Em seguida, se você tiver problemas e precisar recomeçar, será mais fácil começar do projeto salvo, em vez de reverter as etapas executadas para este tutorial ou voltar ao início da série inteira.

## Criar a classe Person

Na pasta Models, crie Person.cs e substitua o código de modelo pelo seguinte código:

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public abstract class Person
    {
        public int ID { get; set; }

        [Required]
        [StringLength(50)]
        [Display(Name = "Last Name")]
        public string LastName { get; set; }

        [Required]
        [StringLength(50, ErrorMessage = "First name cannot be longer than 50 characters.")]
        [Column("FirstName")]
        [Display(Name = "First Name")]
        public string FirstMidName { get; set; }

        [Display(Name = "Full Name")]
        public string FullName
        {
            get
            {
                return LastName + ", " + FirstMidName;
            }
        }
    }
}

```

## Fazer com que as classes Student e Instructor herdem de Person

Em *Instructor.cs*, derive a classe Instructor da classe Person e remova os campos de nome e chave. O código será semelhante ao seguinte exemplo:

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Instructor : Person
    {
        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        [Display(Name = "Hire Date")]
        public DateTime HireDate { get; set; }

        public ICollection<CourseAssignment> CourseAssignments { get; set; }
        public OfficeAssignment OfficeAssignment { get; set; }
    }
}

```

Faça as mesmas alterações em *Student.cs*.

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Student : Person
    {
        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        [Display(Name = "Enrollment Date")]
        public DateTime EnrollmentDate { get; set; }

        public ICollection<Enrollment> Enrollments { get; set; }
    }
}

```

## Adicionar o tipo de entidade Person ao modelo de dados

Adicione o tipo de entidade Person a *SchoolContext.cs*. As novas linhas são realçadas.

```

using ContosoUniversity.Models;
using Microsoft.EntityFrameworkCore;

namespace ContosoUniversity.Data
{
    public class SchoolContext : DbContext
    {
        public SchoolContext(DbContextOptions<SchoolContext> options) : base(options)
        {
        }

        public DbSet<Course> Courses { get; set; }
        public DbSet<Enrollment> Enrollments { get; set; }
        public DbSet<Student> Students { get; set; }
        public DbSet<Department> Departments { get; set; }
        public DbSet<Instructor> Instructors { get; set; }
        public DbSet<OfficeAssignment> OfficeAssignments { get; set; }
        public DbSet<CourseAssignment> CourseAssignments { get; set; }
        public DbSet<Person> People { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Course>().ToTable("Course");
            modelBuilder.Entity<Enrollment>().ToTable("Enrollment");
            modelBuilder.Entity<Student>().ToTable("Student");
            modelBuilder.Entity<Department>().ToTable("Department");
            modelBuilder.Entity<Instructor>().ToTable("Instructor");
            modelBuilder.Entity<OfficeAssignment>().ToTable("OfficeAssignment");
            modelBuilder.Entity<CourseAssignment>().ToTable("CourseAssignment");
            modelBuilder.Entity<Person>().ToTable("Person");

            modelBuilder.Entity<CourseAssignment>()
                .HasKey(c => new { c.CourseID, c.InstructorID });
        }
    }
}

```

Isso é tudo o que o Entity Framework precisa para configurar a herança de tabela por hierarquia. Como você verá, quando o banco de dados for atualizado, ele terá uma tabela Pessoa no lugar das tabelas Aluno e Instrutor.

# Criar e personalizar o código de migração

Salve as alterações e compile o projeto. Em seguida, abra a janela Comando na pasta do projeto e insira o seguinte comando:

```
dotnet ef migrations add Inheritance
```

Não execute o comando `database update` ainda. Esse comando resultará em perda de dados porque ele removerá a tabela Instrutor e renomeará a tabela Aluno como Pessoa. Você precisa fornecer o código personalizado para preservar os dados existentes.

Abra *Migrations/<timestamp>\_Inheritance.cs* e substitua o método `Up` pelo seguinte código:

```
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropForeignKey(
        name: "FK_Enrollment_Student_StudentID",
        table: "Enrollment");

    migrationBuilder DropIndex(name: "IX_Enrollment_StudentID", table: "Enrollment");

    migrationBuilder.RenameTable(name: "Instructor", newName: "Person");
    migrationBuilder.AddColumn<DateTime>(name: "EnrollmentDate", table: "Person", nullable: true);
    migrationBuilder.AddColumn<string>(name: "Discriminator", table: "Person", nullable: false, maxLength: 128, defaultValue: "Instructor");
    migrationBuilder.AlterColumn<DateTime>(name: "HireDate", table: "Person", nullable: true);
    migrationBuilder.AddColumn<int>(name: "OldId", table: "Person", nullable: true);

    // Copy existing Student data into new Person table.
    migrationBuilder.Sql("INSERT INTO dbo.Person (LastName, FirstName, HireDate, EnrollmentDate, Discriminator, OldId) SELECT LastName, FirstName, null AS HireDate, EnrollmentDate, 'Student' AS Discriminator, ID AS OldId FROM dbo.Student");
    // Fix up existing relationships to match new PK's.
    migrationBuilder.Sql("UPDATE dbo.Enrollment SET StudentId = (SELECT ID FROM dbo.Person WHERE OldId = Enrollment.StudentId AND Discriminator = 'Student')");

    // Remove temporary key
    migrationBuilder.DropColumn(name: "OldID", table: "Person");

    migrationBuilder.DropTable(
        name: "Student");

    migrationBuilder.CreateIndex(
        name: "IX_Enrollment_StudentID",
        table: "Enrollment",
        column: "StudentID");

    migrationBuilder.AddForeignKey(
        name: "FK_Enrollment_Person_StudentID",
        table: "Enrollment",
        column: "StudentID",
        principalTable: "Person",
        principalColumn: "ID",
        onDelete: ReferentialAction.Cascade);
}
```

Este código é responsável pelas seguintes tarefas de atualização de banco de dados:

- Remove as restrições de chave estrangeira e índices que apontam para a tabela Aluno.
- Renomeia a tabela Instrutor como Pessoa e faz as alterações necessárias para que ela armazene dados de Aluno:

- Adiciona EnrollmentDate que permite valor nulo para os alunos.
- Adiciona a coluna Discriminatória para indicar se uma linha refere-se a um aluno ou um instrutor.
- Faz com que HireDate permita valor nulo, pois linhas de alunos não terão datas de contratação.
- Adiciona um campo temporário que será usado para atualizar chaves estrangeiras que apontam para alunos. Quando você copiar os alunos para a tabela Person, eles receberão novos valores de chave primária.
- Copia os dados da tabela Aluno para a tabela Pessoa. Isso faz com que os alunos recebam novos valores de chave primária.
- Corrigi valores de chave estrangeira que apontam para alunos.
- Recria restrições de chave estrangeira e índices, agora apontando-os para a tabela Person.

(Se você tiver usado o GUID, em vez de inteiro como o tipo de chave primária, os valores de chave primária dos alunos não precisarão ser alterados e várias dessas etapas poderão ser omitidas.)

Execute o comando `database update` :

```
dotnet ef database update
```

(Em um sistema de produção, você fará as alterações correspondentes no método `Down`, caso já tenha usado isso para voltar à versão anterior do banco de dados. Para este tutorial, você não usará o método `Down`.)

#### **NOTE**

É possível receber outros erros ao fazer alterações de esquema em um banco de dados que contém dados existentes. Se você receber erros de migração que não consegue resolver, altere o nome do banco de dados na cadeia de conexão ou exclua o banco de dados. Com um novo banco de dados, não há nenhum dado a ser migrado e o comando de atualização de banco de dados terá uma probabilidade maior de ser concluído sem erros. Para excluir o banco de dados, use o SSOX ou execute o comando `database drop` da CLI.

## Testar com a herança implementada

Execute o aplicativo e teste várias páginas. Tudo funciona da mesma maneira que antes.

No **Pesquisador de Objetos do SQL Server**, expanda **Data Connections/SchoolContext** e, em seguida, **Tabelas** e você verá que as tabelas Aluno e Instrutor foram substituídas por uma tabela Pessoa. Abra o designer de tabela Pessoa e você verá que ela contém todas as colunas que costumavam estar nas tabelas Aluno e Instrutor.

**dbo.Person [Design]**

Update | Script File: dbo.Person.sql

	Name	Data Type	Allow Nulls	Default
ID	int	<input type="checkbox"/>		
FirstName	nvarchar(50)	<input type="checkbox"/>		
HireDate	datetime2(7)	<input checked="" type="checkbox"/>		
LastName	nvarchar(50)	<input type="checkbox"/>		
EnrollmentDate	datetime2(7)	<input checked="" type="checkbox"/>		
Discriminator	nvarchar(128)	<input type="checkbox"/>	(N'Instructor')	

**Keys (1)**  
PK\_Instructor (Primary Key)

**Check Constraints (0)**

**Indexes (0)**

**Foreign Keys (0)**

**Triggers (0)**

**Design** **T-SQL**

```

1  CREATE TABLE [dbo].[Person] (
2      [ID]             INT            IDENTITY (1, 1) NOT NULL,
3      [FirstName]      NVARCHAR (50) NOT NULL,
4      [HireDate]       DATETIME2 (7) NULL,
5      [LastName]       NVARCHAR (50) NOT NULL,
6      [EnrollmentDate] DATETIME2 (7) NULL,
7      [Discriminator] NVARCHAR (128) DEFAULT (N'Instructor') NOT NULL,
8      CONSTRAINT [PK_Instructor] PRIMARY KEY CLUSTERED ([ID] ASC)
9  );
10 
```

100 %

Connection Ready | (localdb)\MSSQLLocalDB | REDMOND\tdykstra | aspnet-ContosoUniversi...

Clique com o botão direito do mouse na tabela Person e, em seguida, clique em **Mostrar Dados da Tabela** para ver a coluna discriminatória.

**dbo.Person [Data]**

Max Rows: 1000

	ID	FirstName	HireDate	LastName	Enrollme...	Discriminator
1	1	Kim	3/11/1995...	Abercrombie	NULL	Instructor
2	2	Fadi	7/6/2002 ...	Fakhouri	NULL	Instructor
3	3	Roger	7/1/1998 ...	Harui	NULL	Instructor
4	4	Candace	1/15/2001...	Kapoor	NULL	Instructor
5	5	Roger	2/12/2004...	Zheng	NULL	Instructor
7	7	Nancy	8/17/2016...	Davolio	NULL	Instructor
8	8	Carson	NULL	Alexander	9/1/2010 ...	Student
9	9	Meredith	NULL	Alonso	9/1/2012 ...	Student
10	10	Arturo	NULL	Anand	9/1/2013 ...	Student
11	11	Gytis	NULL	Barzdukas	9/1/2012 ...	Student
12	12	Yan	NULL	Li	9/1/2012	Student

## Resumo

Você implementou a herança de tabela por hierarquia para as classes `Person`, `Student` e `Instructor`. Para obter mais informações sobre a herança no Entity Framework Core, consulte [Herança](#). No próximo tutorial, você verá como lidar com uma variedade de cenários relativamente avançados do Entity Framework.

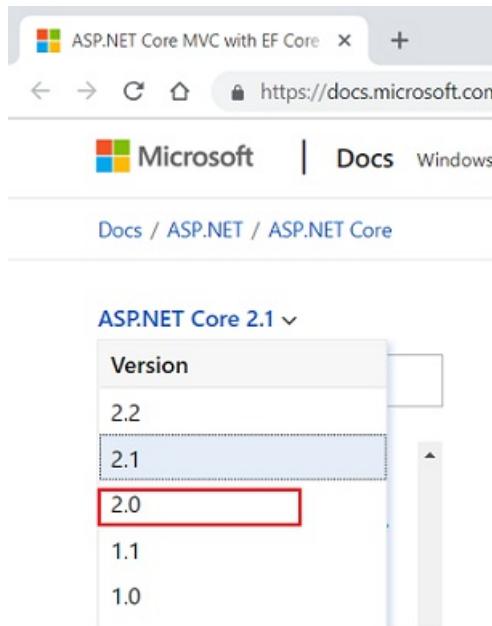
[ANTERIOR](#)

[PRÓXIMO](#)

# ASP.NET Core MVC com EF Core – avançado – 10 de 10

21/01/2019 • 25 minutes to read • [Edit Online](#)

Este tutorial não foi atualizado para o ASP.NET Core 2.1. A versão deste tutorial para o ASP.NET Core 2.0 pode ser consultada selecionando **ASP.NET Core 2.0** acima do sumário ou na parte superior da página:



A versão deste tutorial do [Razor Pages](#) para o ASP.NET Core 2.1 conta com várias melhorias em relação à versão 2.0.

O tutorial do 2.0 ensina a usar o ASP.NET Core MVC e o Entity Framework Core com controladores e exibições. O tutorial do Razor Pages foi atualizado com os seguintes aprimoramentos:

- É mais fácil de acompanhar. Por exemplo, o código de scaffolding foi bastante simplificado.
- Fornece mais práticas recomendadas do EF Core.
- Usa consultas mais eficientes.
- Usa a API do EF Core mais recente.

Por [Tom Dykstra](#) e [Rick Anderson](#)

O aplicativo web de exemplo Contoso University demonstra como criar aplicativos web do ASP.NET Core MVC usando o Entity Framework Core e o Visual Studio. Para obter informações sobre a série de tutoriais, consulte [o primeiro tutorial da série](#).

No tutorial anterior, você implementou a herança de tabela por hierarquia. Este tutorial apresenta vários tópicos que são úteis para consideração quando você vai além dos conceitos básicos de desenvolvimento de aplicativos Web ASP.NET Core que usam o Entity Framework Core.

## Consultas SQL brutas

Uma das vantagens de usar o Entity Framework é que ele evita vincular o código de forma muito próxima a um método específico de armazenamento de dados. Ele faz isso pela geração de consultas SQL e comandos para você, que também libera você da necessidade de escrevê-los. Mas há casos excepcionais em que você precisa executar consultas SQL específicas criadas manualmente. Para esses cenários, a API do Code First do Entity

Framework inclui métodos que permitem passar comandos SQL diretamente para o banco de dados. Você tem as seguintes opções no EF Core 1.0:

- Use o método `DbSet.FromSql` para consultas que retornam tipos de entidade. Os objetos retornados precisam ser do tipo esperado pelo objeto `DbSet` e são controlados automaticamente pelo contexto de banco de dados, a menos que você [desative o controle](#).
- Use o `Database.ExecuteSqlCommand` para comandos que não sejam de consulta.

Caso precise executar uma consulta que retorna tipos que não são entidades, use o ADO.NET com a conexão de banco de dados fornecida pelo EF. Os dados retornados não são controlados pelo contexto de banco de dados, mesmo se esse método é usado para recuperar tipos de entidade.

Como é sempre verdadeiro quando você executa comandos SQL em um aplicativo Web, é necessário tomar precauções para proteger o site contra ataques de injeção de SQL. Uma maneira de fazer isso é usar consultas parametrizadas para garantir que as cadeias de caracteres enviadas por uma página da Web não possam ser interpretadas como comandos SQL. Neste tutorial, você usará consultas parametrizadas ao integrar a entrada do usuário a uma consulta.

## Chamar uma consulta que retorna entidades

A classe `DbSet< TEntity >` fornece um método que você pode usar para executar uma consulta que retorna uma entidade do tipo  `TEntity` . Para ver como isso funciona, você alterará o código no método `Details` do controlador Departamento.

Em `DepartmentsController.cs`, no método `Details`, substitua o código que recupera um departamento com uma chamada de método `FromSql`, conforme mostrado no seguinte código realçado:

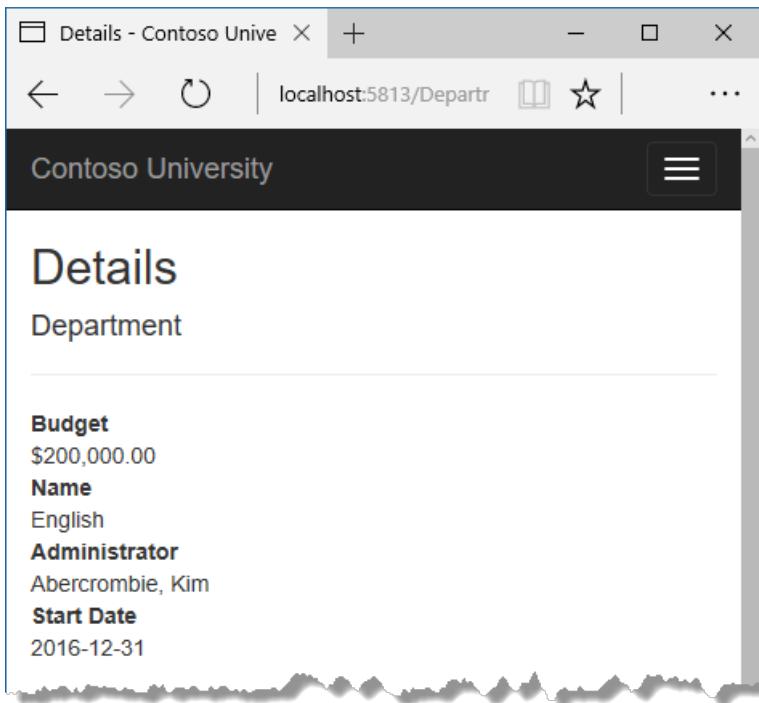
```
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    string query = "SELECT * FROM Department WHERE DepartmentID = {0}";
    var department = await _context.Departments
        .FromSql(query, id)
        .Include(d => d.Administrator)
        .AsNoTracking()
        .SingleOrDefaultAsync();

    if (department == null)
    {
        return NotFound();
    }

    return View(department);
}
```

Para verificar se o novo código funciona corretamente, selecione a guia **Departamentos** e, em seguida, **Detalhes** de um dos departamentos.



## Chamar uma consulta que retorna outros tipos

Anteriormente, você criou uma grade de estatísticas de alunos para a página Sobre que mostrava o número de alunos para cada data de registro. Você obteve os dados do conjunto de entidades Students (`_context.Students`) e usou o LINQ para projetar os resultados em uma lista de objetos de modelo de exibição `EnrollmentDateGroup`. Suponha que você deseja gravar o próprio SQL em vez de usar LINQ. Para fazer isso, você precisa executar uma consulta SQL que retorna algo diferente de objetos de entidade. No EF Core 1.0, uma maneira de fazer isso é escrever um código ADO.NET e obter a conexão de banco de dados do EF.

Em `HomeController.cs`, substitua o método `About` pelo seguinte código:

```

public async Task<ActionResult> About()
{
    List<EnrollmentDateGroup> groups = new List<EnrollmentDateGroup>();
    var conn = _context.Database.GetDbConnection();
    try
    {
        await conn.OpenAsync();
        using (var command = conn.CreateCommand())
        {
            string query = "SELECT EnrollmentDate, COUNT(*) AS StudentCount "
                + "FROM Person "
                + "WHERE Discriminator = 'Student' "
                + "GROUP BY EnrollmentDate";
            command.CommandText = query;
            DbDataReader reader = await command.ExecuteReaderAsync();

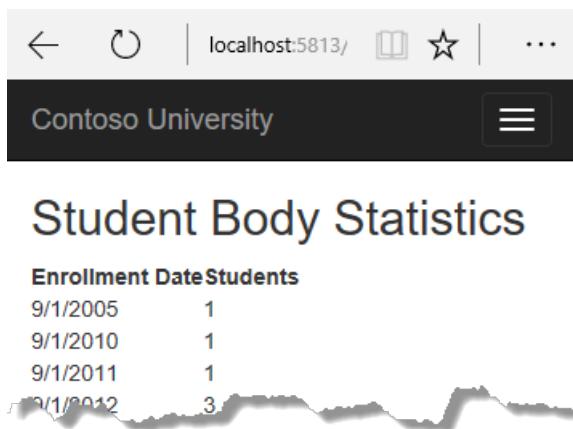
            if (reader.HasRows)
            {
                while (await reader.ReadAsync())
                {
                    var row = new EnrollmentDateGroup { EnrollmentDate = reader.GetDateTime(0), StudentCount
= reader.GetInt32(1) };
                    groups.Add(row);
                }
            }
            reader.Dispose();
        }
    }
    finally
    {
        conn.Close();
    }
    return View(groups);
}

```

Adicionar uma instrução using:

```
using System.Data.Common;
```

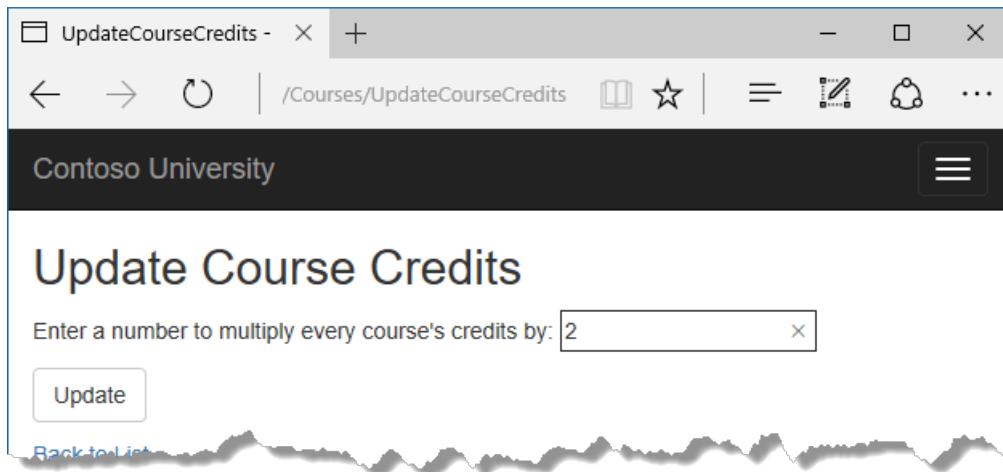
Execute o aplicativo e accese a página Sobre. Ela exibe os mesmos dados que antes.



## Chamar uma consulta de atualização

Suponha que os administradores do Contoso University desejem executar alterações globais no banco de dados, como alterar o número de créditos para cada curso. Se a universidade tiver uma grande quantidade de cursos, poderá ser ineficiente recuperá-los como entidades e alterá-los individualmente. Nesta seção, você implementará uma página da Web que permite ao usuário especificar um fator pelo qual alterar o número de créditos para

todos os cursos e você fará a alteração executando uma instrução SQL UPDATE. A página da Web será semelhante à seguinte ilustração:



Em *CoursesController.cs*, adicione métodos `UpdateCourseCredits` para `HttpGet` e `HttpPost`:

```
public IActionResult UpdateCourseCredits()
{
    return View();
}

[HttpPost]
public async Task<IActionResult> UpdateCourseCredits(int? multiplier)
{
    if (multiplier != null)
    {
        ViewData["RowsAffected"] =
            await _context.Database.ExecuteSqlCommandAsync(
                "UPDATE Course SET Credits = Credits * {0}",
                parameters: multiplier);
    }
    return View();
}
```

Quando o controlador processa uma solicitação `HttpGet`, nada é retornado em `ViewData["RowsAffected"]`, e a exibição mostra uma caixa de texto vazia e um botão Enviar, conforme mostrado na ilustração anterior.

Quando o botão **Atualizar** recebe um clique, o método `HttpPost` é chamado e multiplicador tem o valor inserido na caixa de texto. Em seguida, o código executa o SQL que atualiza os cursos e retorna o número de linhas afetadas para a exibição em `ViewData`. Quando a exibição obtém um valor `RowsAffected`, ela mostra o número de linhas atualizadas.

No **Gerenciador de Soluções**, clique com o botão direito do mouse na pasta *Views/Courses* e, em seguida, clique em **Adicionar > Novo Item**.

Na caixa de diálogo **Adicionar Novo Item**, clique em **ASP.NET** em **Instalado** no painel esquerdo, clique em **Página de Exibição MVC** e nomeie a nova exibição *UpdateCourseCredits.cshtml*.

Em *Views/Courses/UpdateCourseCredits.cshtml*, substitua o código de modelo pelo seguinte código:

```

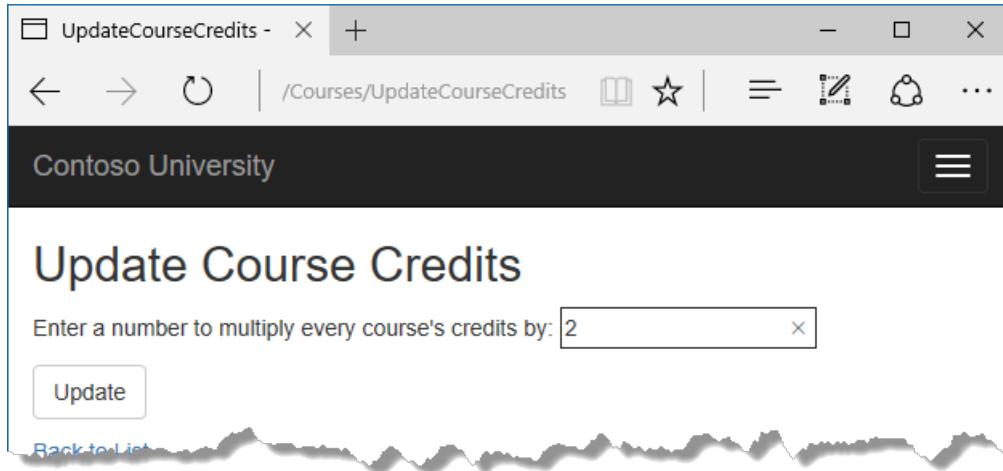
@{
    ViewBag.Title = "UpdateCourseCredits";
}

<h2>Update Course Credits</h2>

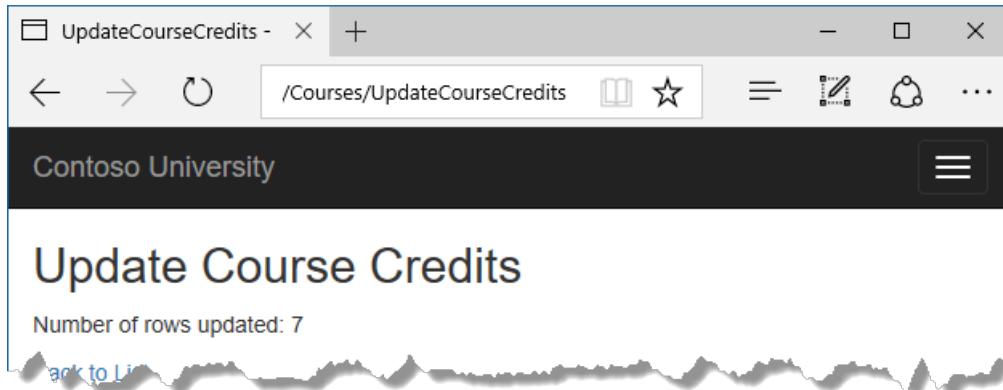
@if (ViewData["RowsAffected"] == null)
{
    <form asp-action="UpdateCourseCredits">
        <div class="form-actions no-color">
            <p>
                Enter a number to multiply every course's credits by: @Html.TextBox("multiplier")
            </p>
            <p>
                <input type="submit" value="Update" class="btn btn-default" />
            </p>
        </div>
    </form>
}
@if (ViewData["RowsAffected"] != null)
{
    <p>
        Number of rows updated: @ViewData["RowsAffected"]
    </p>
}
<div>
    @Html.ActionLink("Back to List", "Index")
</div>

```

Execute o método `UpdateCourseCredits` selecionando a guia **Cursos**, adicionando, em seguida, `/UpdateCourseCredits` ao final da URL na barra de endereços do navegador (por exemplo: <http://localhost:5813/Courses/UpdateCourseCredits>). Insira um número na caixa de texto:



Clique em **Atualizar**. O número de linhas afetadas é exibido:



Clique em **Voltar para a Lista** para ver a lista de cursos com o número revisado de créditos.

Observe que o código de produção deve garantir que as atualizações sempre resultem em dados válidos. O código simplificado mostrado aqui pode multiplicar o número de créditos o suficiente para resultar em números maiores que 5. (A propriedade `Credits` tem um atributo `[Range(0, 5)]`.) A consulta de atualização funciona, mas os dados inválidos podem causar resultados inesperados em outras partes do sistema que supõem que o número de créditos seja 5 ou inferior.

Para obter mais informações sobre consultas SQL brutas, consulte [Consultas SQL brutas](#).

## Examinar o SQL enviado ao banco de dados

Às vezes, é útil poder ver as consultas SQL reais que são enviadas ao banco de dados. A funcionalidade de log interno do ASP.NET Core é usada automaticamente pelo EF Core para gravar logs que contêm o SQL de consultas e atualizações. Nesta seção, você verá alguns exemplos de log do SQL.

Abra `StudentsController.cs` e, no método `Details`, defina um ponto de interrupção na instrução

```
if (student == null).
```

Execute o aplicativo no modo de depuração e acesse a página Detalhes de um aluno.

Acesse a janela de **Saída** mostrando a saída de depuração e você verá a consulta:

```
Microsoft.EntityFrameworkCore.Database.Command:Information: Executed DbCommand (56ms) [Parameters=@__id_0='?'], CommandType='Text', CommandTimeout='30']
SELECT TOP(2) [s].[ID], [s].[Discriminator], [s].[FirstName], [s].[LastName], [s].[EnrollmentDate]
FROM [Person] AS [s]
WHERE ([s].[Discriminator] = N'Student') AND ([s].[ID] = @__id_0)
ORDER BY [s].[ID]
Microsoft.EntityFrameworkCore.Database.Command:Information: Executed DbCommand (122ms) [Parameters=@__id_0='?'], CommandType='Text', CommandTimeout='30'
SELECT [s.Enrollments].[EnrollmentID], [s.Enrollments].[CourseID], [s.Enrollments].[Grade], [s.Enrollments].[StudentID], [e.Course].[CourseID], [e.Course].[Credits], [e.Course].[DepartmentID], [e.Course].[Title]
FROM [Enrollment] AS [s.Enrollments]
INNER JOIN [Course] AS [e.Course] ON [s.Enrollments].[CourseID] = [e.Course].[CourseID]
INNER JOIN (
    SELECT TOP(1) [s0].[ID]
    FROM [Person] AS [s0]
    WHERE ([s0].[Discriminator] = N'Student') AND ([s0].[ID] = @__id_0)
    ORDER BY [s0].[ID]
) AS [t] ON [s.Enrollments].[StudentID] = [t].[ID]
ORDER BY [t].[ID]
```

Você observará algo aqui que pode ser surpreendente: o SQL seleciona até 2 linhas (`TOP(2)`) da tabela Person.

O método `SingleOrDefaultAsync` não é resolvido para uma 1 linha no servidor. Eis o porquê:

- Se a consulta retorna várias linhas, o método retorna nulo.
- Para determinar se a consulta retorna várias linhas, o EF precisa verificar se ela retorna pelo menos 2.

Observe que você não precisa usar o modo de depuração e parar em um ponto de interrupção para obter a saída de log na janela de **Saída**. É apenas um modo conveniente de parar o log no ponto em que você deseja examinar a saída. Se você não fizer isso, o log continuará e você precisará rolar para baixo para localizar as partes de seu interesse.

## Padrões de repositório e unidade de trabalho

Muitos desenvolvedores escrevem um código para implementar padrões de repositório e unidade de trabalho como um wrapper em torno do código que funciona com o Entity Framework. Esses padrões destinam-se a criar uma camada de abstração entre a camada de acesso a dados e a camada da lógica de negócios de um aplicativo.

A implementação desses padrões pode ajudar a isolar o aplicativo de alterações no armazenamento de dados e pode facilitar o teste de unidade automatizado ou TDD (desenvolvimento orientado por testes). No entanto, escrever um código adicional para implementar esses padrões nem sempre é a melhor escolha para aplicativos que usam o EF, por vários motivos:

- A própria classe de contexto do EF isola o código de código específico a um armazenamento de dados.
- A classe de contexto do EF pode atuar como uma classe de unidade de trabalho para as atualizações de banco de dados feitas com o EF.
- O EF inclui recursos para implementar o TDD sem escrever um código de repositório.

Para obter informações sobre como implementar os padrões de repositório e unidade de trabalho, consulte [a versão do Entity Framework 5 desta série de tutoriais](#).

O Entity Framework Core implementa um provedor de banco de dados em memória que pode ser usado para teste. Para obter mais informações, confira [Testar com InMemory](#).

## Detecção automática de alterações

O Entity Framework determina como uma entidade foi alterada (e, portanto, quais atualizações precisam ser enviadas ao banco de dados), comparando os valores atuais de uma entidade com os valores originais. Os valores originais são armazenados quando a entidade é consultada ou anexada. Alguns dos métodos que causam a detecção automática de alterações são os seguintes:

- `DbContext.SaveChanges`
- `DbContext.Entry`
- `ChangeTracker.Entries`

Se você estiver controlando um grande número de entidades e chamar um desses métodos muitas vezes em um loop, poderá obter melhorias significativas de desempenho desativando temporariamente a detecção automática de alterações usando a propriedade `ChangeTracker.AutoDetectChangesEnabled`. Por exemplo:

```
_context.ChangeTracker.AutoDetectChangesEnabled = false;
```

## Código-fonte e planos de desenvolvimento do Entity Framework Core

O código-fonte do Entity Framework Core está em <https://github.com/aspnet/EntityFrameworkCore>. O repositório do EF Core contém builds noturnos, acompanhamento de questões, especificações de recurso, notas de reuniões de design e [o roteiro para desenvolvimento futuro](#). Arquive ou encontre bugs e contribua.

Embora o código-fonte seja aberto, há suporte completo para o Entity Framework Core como um produto Microsoft. A equipe do Microsoft Entity Framework mantém controle sobre quais contribuições são aceitas e testa todas as alterações de código para garantir a qualidade de cada versão.

## Fazer engenharia reversa do banco de dados existente

Para fazer engenharia reversa de um modelo de dados, incluindo classes de entidade de um banco de dados existente, use o comando `scaffold-dbcontext`. Consulte o [tutorial de introdução](#).

## Usar o LINQ dinâmico para simplificar o código de seleção de classificação

O [terceiro tutorial desta série](#) mostra como escrever um código LINQ embutindo nomes de colunas em código

em uma instrução `switch`. Com duas colunas para escolha, isso funciona bem, mas se você tiver muitas colunas, o código poderá ficar detalhado. Para resolver esse problema, use o método `EF.Property` para especificar o nome da propriedade como uma cadeia de caracteres. Para usar essa abordagem, substitua o método `Index` no `StudentsController` pelo código a seguir.

```
public async Task<IActionResult> Index(
    string sortOrder,
    string currentFilter,
    string searchString,
    int? page)
{
    ViewData["CurrentSort"] = sortOrder;
    ViewData["NameSortParm"] =
        String.IsNullOrEmpty(sortOrder) ? "LastName_desc" : "";
    ViewData["DateSortParm"] =
        sortOrder == "EnrollmentDate" ? "EnrollmentDate_desc" : "EnrollmentDate";

    if (searchString != null)
    {
        page = 1;
    }
    else
    {
        searchString = currentFilter;
    }

    ViewData["CurrentFilter"] = searchString;

    var students = from s in _context.Students
                   select s;

    if (!String.IsNullOrEmpty(searchString))
    {
        students = students.Where(s => s.LastName.Contains(searchString)
                             || s.FirstMidName.Contains(searchString));
    }

    if (string.IsNullOrEmpty(sortOrder))
    {
        sortOrder = "LastName";
    }

    bool descending = false;
    if (sortOrder.EndsWith("_desc"))
    {
        sortOrder = sortOrder.Substring(0, sortOrder.Length - 5);
        descending = true;
    }

    if (descending)
    {
        students = students.OrderByDescending(e => EF.Property<object>(e, sortOrder));
    }
    else
    {
        students = students.OrderBy(e => EF.Property<object>(e, sortOrder));
    }

    int pageSize = 3;
    return View(await PaginatedList<Student>.CreateAsync(students.AsNoTracking(),
        page ?? 1, pageSize));
}
```

## Próximas etapas

Isso conclui esta série de tutoriais sobre como usar o Entity Framework Core em um aplicativo ASP.NET Core MVC.

Para obter mais informações sobre o EF Core, consulte a [documentação do Entity Framework Core](#). Também há um livro disponível: [Entity Framework Core in Action](#) (Entity Framework Core em ação).

Para obter informações sobre como implantar um aplicativo Web, confira [Hospedar e implantar o ASP.NET Core](#).

Para obter informações sobre outros tópicos relacionados ao ASP.NET Core MVC, como autenticação e autorização, confira [Introdução ao ASP.NET Core](#).

## Agradecimentos

Tom Dykstra e Rick Anderson (twitter @RickAndMSFT) escreveram este tutorial. Rowan Miller, Diego Vega e outros membros da equipe do Entity Framework auxiliaram com revisões de código e ajudaram com problemas de depuração que surgiram durante a codificação para os tutoriais.

## Erros comuns

### **ContosoUniversity.dll usada por outro processo**

Mensagem de erro:

Não é possível abrir '...bin\Debug\netcoreapp1.0\ContosoUniversity.dll' para gravação – 'O processo não pode acessar o arquivo '...bin\Debug\netcoreapp1.0\ContosoUniversity.dll' porque ele está sendo usado por outro processo.

Solução:

Pare o site no IIS Express. Acesse a Bandeja do Sistema do Windows, localize o IIS Express e clique com o botão direito do mouse em seu ícone, selecione o site da Contoso University e, em seguida, clique em **Parar Site**.

### **Migração gerada por scaffolding sem nenhum código nos métodos Up e Down**

Possível causa:

Os comandos da CLI do EF não fecham e salvam arquivos de código automaticamente. Se você tiver alterações não salvas ao executar o comando `migrations add`, o EF não encontrará as alterações.

Solução:

Execute o comando `migrations remove`, salve as alterações de código e execute o comando `migrations add` novamente.

### **Erros durante a execução da atualização de banco de dados**

É possível receber outros erros ao fazer alterações de esquema em um banco de dados que contém dados existentes. Se você receber erros de migração que não consegue resolver, altere o nome do banco de dados na cadeia de conexão ou exclua o banco de dados. Com um novo banco de dados, não há nenhum dado a ser migrado e o comando de atualização de banco de dados terá uma probabilidade muito maior de ser concluído sem erros.

A abordagem mais simples é renomear o banco de dados em `appsettings.json`. Na próxima vez que você executar `database update`, um novo banco de dados será criado.

Para excluir um banco de dados no SSOX, clique com o botão direito do mouse no banco de dados, clique **Excluir** e, em seguida, na caixa de diálogo **Excluir Banco de Dados**, selecione **Fechar conexões existentes** e clique em **OK**.

Para excluir um banco de dados usando a CLI, execute o comando `database drop` da CLI:

```
dotnet ef database drop
```

### Erro ao localizar a instância do SQL Server

Mensagem de erro:

Ocorreu um erro relacionado à rede ou específico a uma instância ao estabelecer uma conexão com o SQL Server. O servidor não foi encontrado ou não estava acessível. Verifique se o nome da instância está correto e se o SQL Server está configurado para permitir conexões remotas. (provedor: Adaptadores de Rede do SQL, erro: 26 – Erro ao localizar a instância/o servidor especificado)

Solução:

Verifique a cadeia de conexão. Se você excluiu o arquivo de banco de dados manualmente, altere o nome do banco de dados na cadeia de caracteres de construção para começar novamente com um novo banco de dados.

[ANTERIOR](#)

# Introdução ao ASP.NET Core e ao Entity Framework 6

30/10/2018 • 7 minutes to read • [Edit Online](#)

Por [Paweł Grudzień](#), [Damien Pontifex](#) e [Tom Dykstra](#)

Este artigo mostra como usar o Entity Framework 6 em um aplicativo ASP.NET Core.

## Visão geral

Para usar o Entity Framework 6, o projeto precisa ser compilado no .NET Framework, pois o Entity Framework 6 não dá suporte ao .NET Core. Caso precise de recursos de multiplataforma, faça upgrade para o [Entity Framework Core](#).

A maneira recomendada de usar o Entity Framework 6 em um aplicativo ASP.NET Core é colocar o contexto e as classes de modelo do EF6 em um projeto de biblioteca de classes direcionado à estrutura completa. Adicione uma referência à biblioteca de classes do projeto ASP.NET Core. Consulte a [solução de exemplo do Visual Studio com projetos EF6 e ASP.NET Core](#).

Não é possível colocar um contexto do EF6 em um projeto ASP.NET Core, pois projetos .NET Core não dão suporte a todas as funcionalidades exigidas pelo EF6, como *Enable-Migrations*, que é obrigatória.

Seja qual for o tipo de projeto em que você localize o contexto do EF6, somente ferramentas de linha de comando do EF6 funcionam com um contexto do EF6. Por exemplo, `Scaffold-DbContext` está disponível apenas no Entity Framework Core. Caso precise fazer engenharia reversa de um banco de dados para um modelo do EF6, consulte [Usar o Code First para um banco de dados existente](#).

## Referenciar a estrutura completa e o EF6 no projeto ASP.NET Core

O projeto ASP.NET Core precisa referenciar a estrutura .NET e o EF6. Por exemplo, o arquivo `.csproj` do projeto ASP.NET Core será semelhante ao exemplo a seguir (somente as partes relevantes do arquivo são mostradas).

```
<PropertyGroup>
  <TargetFramework>net452</TargetFramework>
  <PreserveCompilationContext>true</PreserveCompilationContext>
  <AssemblyName>MVCCore</AssemblyName>
  <OutputType>Exe</OutputType>
  <PackageId>MVCCore</PackageId>
</PropertyGroup>
```

Ao criar um novo projeto, use o modelo **Aplicativo Web ASP.NET Core (.NET Framework)**.

## Manipular as cadeias de conexão

As ferramentas de linha de comando do EF6 que você usará no projeto de biblioteca de classes do EF6 exigem um construtor padrão para que possam criar uma instância do contexto. No entanto, provavelmente, você desejará especificar a cadeia de conexão a ser usada no projeto ASP.NET Core, caso em que o construtor de contexto deve ter um parâmetro que permita passar a cadeia de conexão. Veja um exemplo.

```
public class SchoolContext : DbContext
{
    public SchoolContext(string connString) : base(connString)
    {
    }
}
```

Como o contexto do EF6 não tem um construtor sem parâmetros, o projeto do EF6 precisa fornecer uma implementação de [IDbContextFactory](#). As ferramentas de linha de comando do EF6 encontrarão e usarão essa implementação para que possam criar uma instância do contexto. Veja um exemplo.

```
public class SchoolContextFactory : IDbContextFactory<SchoolContext>
{
    public SchoolContext Create()
    {
        return new EF6.SchoolContext("Server=
(localdb)\mssqllocaldb;Database=EF6MVCCore;Trusted_Connection=True;MultipleActiveResultSets=true");
    }
}
```

Nesse código de exemplo, a implementação [IDbContextFactory](#) passa uma cadeia de conexão embutida em código. Essa é a cadeia de conexão que será usada pelas ferramentas de linha de comando. Recomendamos implementar uma estratégia para garantir que a biblioteca de classes use a mesma cadeia de conexão usada pelo aplicativo de chamada. Por exemplo, você pode obter o valor de uma variável de ambiente em ambos os projetos.

## Configurar a injecção de dependência no projeto ASP.NET Core

No arquivo *Startup.cs* do projeto Core, configure o contexto do EF6 para DI (injecção de dependência) em [ConfigureServices](#). Os objetos de contexto do EF devem ser delimitados em escopo a um tempo de vida por solicitação.

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddMvc();
    services.AddScoped<SchoolContext>(_ => new
    SchoolContext(Configuration.GetConnectionString("DefaultConnection")));
}
```

Em seguida, você pode obter uma instância do contexto nos controladores usando a DI. O código é semelhante ao que você escreverá para um contexto do EF Core:

```
public class StudentsController : Controller
{
    private readonly SchoolContext _context;

    public StudentsController(SchoolContext context)
    {
        _context = context;
    }
}
```

## Aplicativo de exemplo

Para obter um aplicativo de exemplo funcional, consulte a [solução de exemplo do Visual Studio](#) que acompanha este artigo.

Esta amostra pode ser criada do zero pelas seguintes etapas no Visual Studio:

- Crie uma solução.
- **Adicionar > Novo Projeto > Web > Aplicativo Web ASP.NET Core**
  - Na caixa de diálogo de seleção de modelo do projeto, selecione API e .NET Framework na lista suspensa
- **Adicionar > Novo Projeto > Windows Desktop > Biblioteca de Classes (.NET Framework)**
- No **PMC** (Console do Gerenciador de Pacotes) dos dois projetos, execute o comando  
`Install-Package Entityframework`.
- No projeto de biblioteca de classes, crie classes de modelo de dados e uma classe de contexto, bem como uma implementação de `IDbContextFactory`.
- No PMC do projeto de biblioteca de classes, execute os comandos `Enable-Migrations` e `Add-Migration Initial`. Caso tenha definido o projeto ASP.NET Core como o projeto de inicialização, adicione `-StartupProjectName EF6` a esses comandos.
- No projeto Core, adicione uma referência de projeto ao projeto de biblioteca de classes.
- No projeto Core, em `Startup.cs`, registre o contexto para DI.
- No projeto Core, em `appsettings.json`, adicione a cadeia de conexão.
- No projeto Core, adicione um controlador e exibições para verificar se é possível ler e gravar dados. (Observe que o scaffolding do ASP.NET Core MVC não funcionará com o contexto do EF6 referenciado da biblioteca de classes.)

## Resumo

Este artigo forneceu diretrizes básicas para usar o Entity Framework 6 em um aplicativo ASP.NET Core.

## Recursos adicionais

- [Entity Framework – configuração baseada em código](#)

# Use o Gulp no ASP.NET Core

28/11/2018 • 18 minutes to read • [Edit Online](#)

Por [Scott Addie](#), [Shayne Boyer](#), e [Alcatrão de David](#)

Em um aplicativo web moderno típico, o processo de compilação pode:

- Agrupar e minificar arquivos JavaScript e CSS.
- Execute ferramentas para chamar as tarefas de empacotamento e minimização antes de cada compilação.
- Arquivos de compilar menos ou SASS para CSS.
- Compile arquivos CoffeeScript ou TypeScript para JavaScript.

Um *executor de tarefas* é uma ferramenta que automatiza a essas tarefas de rotina de desenvolvimento e muito mais. Visual Studio fornece suporte interno para dois executores de tarefas de baseados em JavaScript populares: [Gulp](#) e [Grunt](#).

## Gulp

O gulp é baseado em JavaScript streaming build Kit de ferramentas para o código do lado do cliente.

Normalmente, ele é usado para transmitir arquivos do lado do cliente por meio de uma série de processos quando um evento específico é disparado em um ambiente de compilação. Por exemplo, o Gulp pode ser usado para automatizar [agrupamento e minificação](#) ou a limpeza de um ambiente de desenvolvimento antes de uma nova compilação.

Um conjunto de tarefas do Gulp é definido em *gulpfile.js*. O seguinte JavaScript inclui módulos de Gulp e especifica os caminhos de arquivo a ser referenciado de tarefas disponível em breve:

```
/// <binding Clean='clean' />
"use strict";

var gulp = require("gulp"),
    rimraf = require("rimraf"),
    concat = require("gulp-concat"),
    cssmin = require("gulp-cssmin"),
    uglify = require("gulp-uglify");

var paths = {
    webroot: "./wwwroot/"
};

paths.js = paths.webroot + "js/**/*.js";
paths.minJs = paths.webroot + "js/**/*.min.js";
paths.css = paths.webroot + "css/**/*.css";
paths.minCss = paths.webroot + "css/**/*.min.css";
paths.concatJsDest = paths.webroot + "js/site.min.js";
paths.concatCssDest = paths.webroot + "css/site.min.css";
```

O código acima Especifica quais módulos de nó são necessários. O `require` função importa cada módulo para que as tarefas dependentes podem utilizar seus recursos. Cada um dos módulos importados é atribuída a uma variável. Os módulos podem ser localizados por nome ou caminho. Neste exemplo, os módulos denominado `gulp`, `rimraf`, `gulp-concat`, `gulp-cssmin`, e `gulp-uglify` são recuperados por nome. Além disso, uma série de caminhos são criados para que os locais dos arquivos CSS e JavaScript podem ser reutilizados e referenciados dentro das tarefas. A tabela a seguir fornece descrições dos módulos do incluídos no *gulpfile.js*.

NOME DO MÓDULO	DESCRIÇÃO
Gulp	O sistema de compilação streaming Gulp. Para obter mais informações, consulte <a href="#">gulp</a> .
rimraf	Um módulo de exclusão do nó. Para obter mais informações, consulte <a href="#">rimraf</a> .
gulp-concat	Um módulo que concatena arquivos com base em caractere de nova linha do sistema operacional. Para obter mais informações, consulte <a href="#">gulp-concat</a> .
gulp cssmin	Um módulo que minimiza os arquivos CSS. Para obter mais informações, consulte <a href="#">gulp cssmin</a> .
tarefa uglify gulp	Um módulo que minimiza .js arquivos. Para obter mais informações, consulte <a href="#">tarefa uglify gulp</a> .

Depois que o requisito os módulos são importados, as tarefas podem ser especificadas. Aqui, há seis tarefas registrado, representado pelo código a seguir:

```
gulp.task("clean:js", done => rimraf(paths.concatJsDest, done));
gulp.task("clean:css", done => rimraf(paths.concatCssDest, done));
gulp.task("clean", gulp.series(["clean:js", "clean:css"]));

gulp.task("min:js", () => {
  return gulp.src([paths.js, "!" + paths.minJs], { base: "." })
    .pipe(concat(paths.concatJsDest))
    .pipe(uglify())
    .pipe(gulp.dest("."));
});

gulp.task("min:css", () => {
  return gulp.src([paths.css, "!" + paths.minCss])
    .pipe(concat(paths.concatCssDest))
    .pipe(cssmin())
    .pipe(gulp.dest("."));
});

gulp.task("min", gulp.series(["min:js", "min:css"]));

// A 'default' task is required by Gulp v4
gulp.task("default", gulp.series(["min"]));
```

A tabela a seguir fornece uma explicação sobre as tarefas especificadas no código acima:

NOME DA TAREFA	DESCRIÇÃO
Limpar: js	Uma tarefa que usa o módulo de exclusão do nó rimraf para remover a versão reduzida do arquivo js.
Limpar: css	Uma tarefa que usa o módulo de exclusão do nó rimraf para remover a versão reduzida do arquivo site CSS.
Limpar	Uma tarefa que chama o <code>clean:js</code> tarefa, seguida de <code>clean:css</code> tarefa.

NOME DA TAREFA	DESCRIÇÃO
min:js	Uma tarefa que minimiza e concatena todos os arquivos.js na pasta js. A. Min arquivos são excluídos.
min:CSS	Uma tarefa que minimiza e concatena todos os arquivos.CSS dentro da pasta de css. A. min.css arquivos são excluídos.
min	Uma tarefa que chama o <code>min:js</code> tarefa, seguida de <code>min:css</code> tarefa.

## Execução de tarefas padrão

Se você já não tiver criado um novo aplicativo Web, crie um novo projeto de aplicativo Web ASP.NET no Visual Studio.

1. Abra o *Package.json* arquivo (Adicionar se não existe) e adicione o seguinte.

```
{
  "devDependencies": {
    "gulp": "^4.0.0",
    "gulp-concat": "2.6.1",
    "gulp-cssmin": "0.2.0",
    "gulp-uglify": "3.0.0",
    "rimraf": "2.6.1"
  }
}
```

2. Adicionar um novo arquivo JavaScript ao seu projeto e denomine *gulpfile.js*, em seguida, copie o código a seguir.

```

/// <binding Clean='clean' />
"use strict";

const gulp = require("gulp"),
  rimraf = require("rimraf"),
  concat = require("gulp-concat"),
  cssmin = require("gulp-cssmin"),
  uglify = require("gulp-uglify");

const paths = {
  webroot: "./wwwroot/"
};

paths.js = paths.webroot + "js/**/*.js";
paths.minJs = paths.webroot + "js/**/*.min.js";
paths.css = paths.webroot + "css/**/*.css";
paths.minCss = paths.webroot + "css/**/*.min.css";
paths.concatJsDest = paths.webroot + "js/site.min.js";
paths.concatCssDest = paths.webroot + "css/site.min.css";

gulp.task("clean:js", done => rimraf(paths.concatJsDest, done));
gulp.task("clean:css", done => rimraf(paths.concatCssDest, done));
gulp.task("clean", gulp.series(["clean:js", "clean:css"]));

gulp.task("min:js", () => {
  return gulp.src([paths.js, "!" + paths.minJs], { base: "." })
    .pipe(concat(paths.concatJsDest))
    .pipe(uglify())
    .pipe(gulp.dest("."));
});

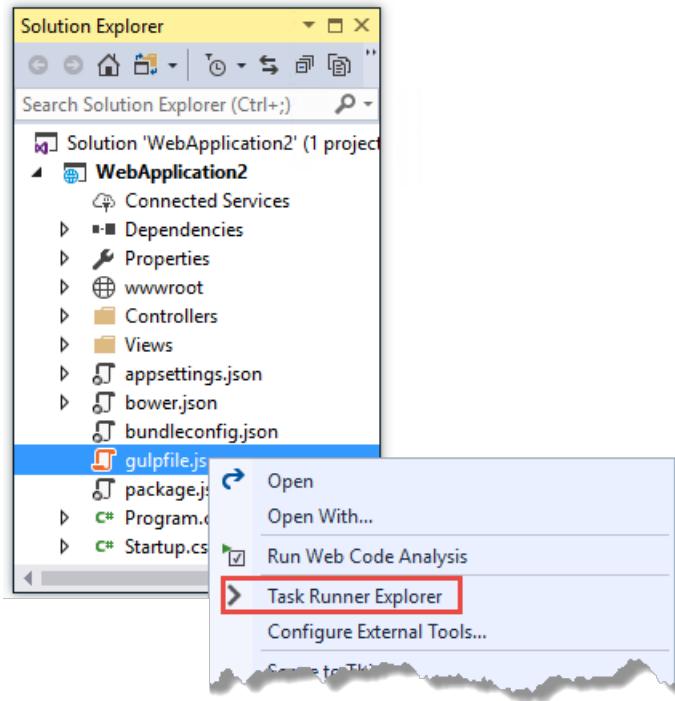
gulp.task("min:css", () => {
  return gulp.src([paths.css, "!" + paths.minCss])
    .pipe(concat(paths.concatCssDest))
    .pipe(cssmin())
    .pipe(gulp.dest("."));
});

gulp.task("min", gulp.series(["min:js", "min:css"]));

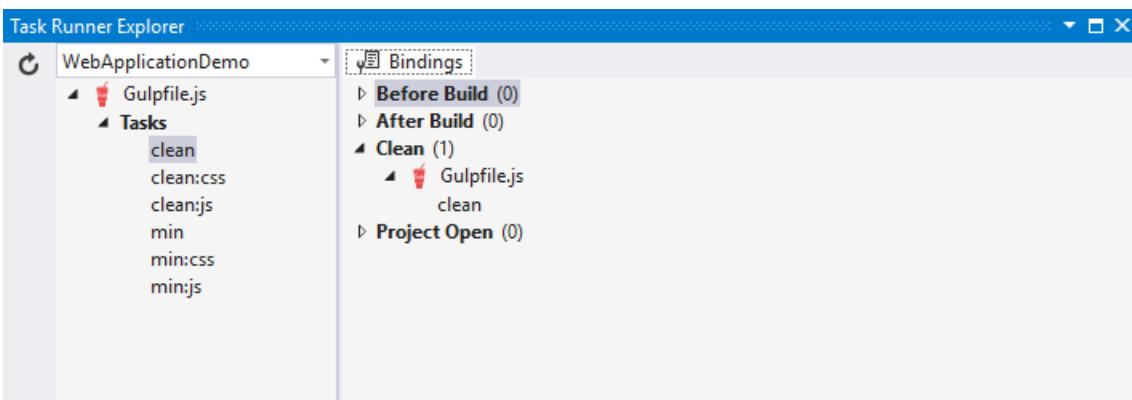
// A 'default' task is required by Gulp v4
gulp.task("default", gulp.series(["min"]));

```

3. Na **Gerenciador de soluções**, clique com botão direito *gulpfile.js* selecione **Task Runner Explorer**.



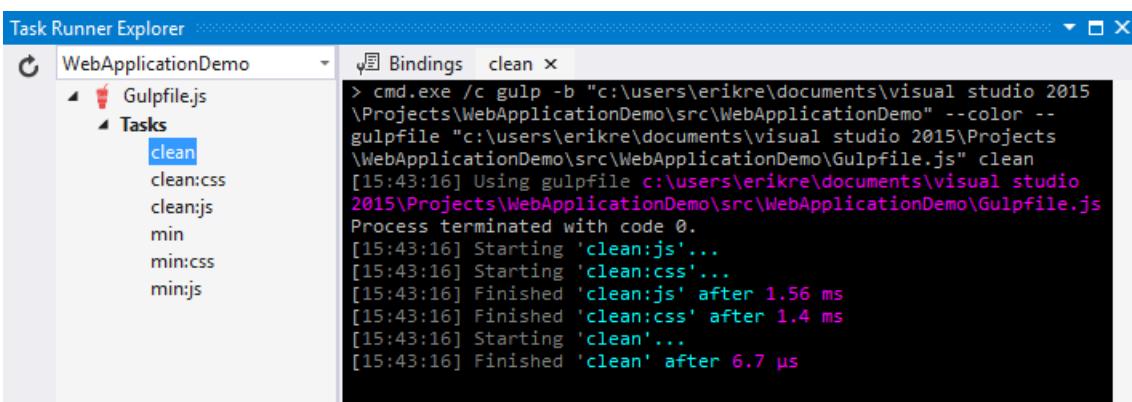
**Explorador do Executador de tarefas** mostra a lista de tarefas de Gulp. (Talvez você precise clicar o **Refresh** botão que aparece à esquerda do nome do projeto.)



#### IMPORTANT

O **Task Runner Explorer** item de menu de contexto será exibida apenas se *gulpfile.js* está no diretório do projeto raiz.

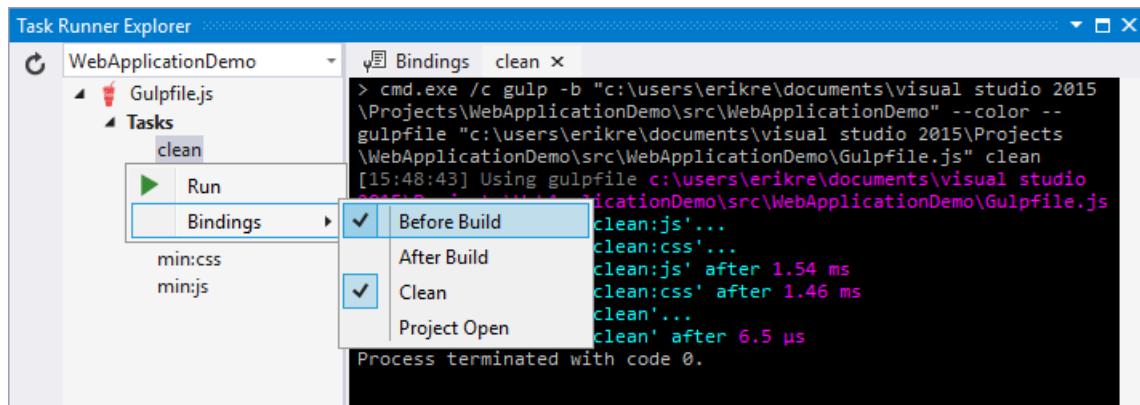
- Sob **tarefas** na **Task Runner Explorer**, clique com botão direito **limpa** e selecione **executar** no menu pop-up.



**Explorador do Executador de tarefas** criará uma nova guia chamada **limpa** e execute a tarefa limpar

conforme definido na `gulpfile.js`.

5. Com o botão direito do **limpa** da tarefa e, em seguida, selecione **associações > antes da compilação**.



O **antes da compilação** associação configura a tarefa Limpar para serem executados automaticamente antes de cada compilação do projeto.

As associações que você configura com **Task Runner Explorer** são armazenadas na forma de um comentário na parte superior da sua `gulpfile.js` e entrarão em vigor apenas no Visual Studio. Uma alternativa que não requer o Visual Studio é configurar a execução automática de tarefas de gulp em seu `.csproj` arquivo. Por exemplo, colocar isso no seu `.csproj` arquivo:

```
<Target Name="MyPreCompileTarget" BeforeTargets="Build">
  <Exec Command="gulp clean" />
</Target>
```

Agora a tarefa de limpeza é executada quando você executar o projeto no Visual Studio ou de um prompt de comando usando o **execução dotnet** comando (executar `npm install` primeiro).

## Definir e executar uma nova tarefa

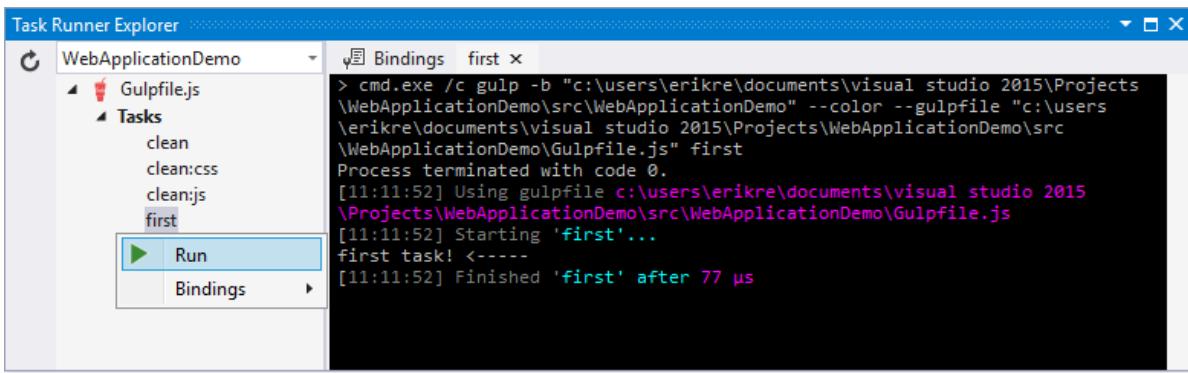
Para definir uma nova tarefa Gulp, modifique `gulpfile.js`.

1. Adicione o seguinte JavaScript ao final da `gulpfile.js`:

```
gulp.task('first', done => {
  console.log('first task! -----');
  done(); // signal completion
});
```

Essa tarefa é denominada `first`, e ele simplesmente exibe uma cadeia de caracteres.

2. Salve `gulpfile.js`.
3. Na **Gerenciador de soluções**, clique com botão direito `gulpfile.js` e selecione **Task Runner Explorer**.
4. Na **Task Runner Explorer**, clique com botão direito `primeiro` e selecione **executar**.



O texto de saída é exibido. Para obter exemplos com base em cenários comuns, consulte [receitas do Gulp](#).

## Definir e executar tarefas em uma série

Quando você executa várias tarefas, as tarefas executadas simultaneamente por padrão. No entanto, se você precisar executar tarefas em uma ordem específica, você deve especificar quando cada tarefa é concluída, bem como quais tarefas dependem da conclusão de outra tarefa.

1. Para definir uma série de tarefas a serem executados em ordem, substitua a `first` que você adicionou acima na tarefa `gulpfile.js` com o seguinte:

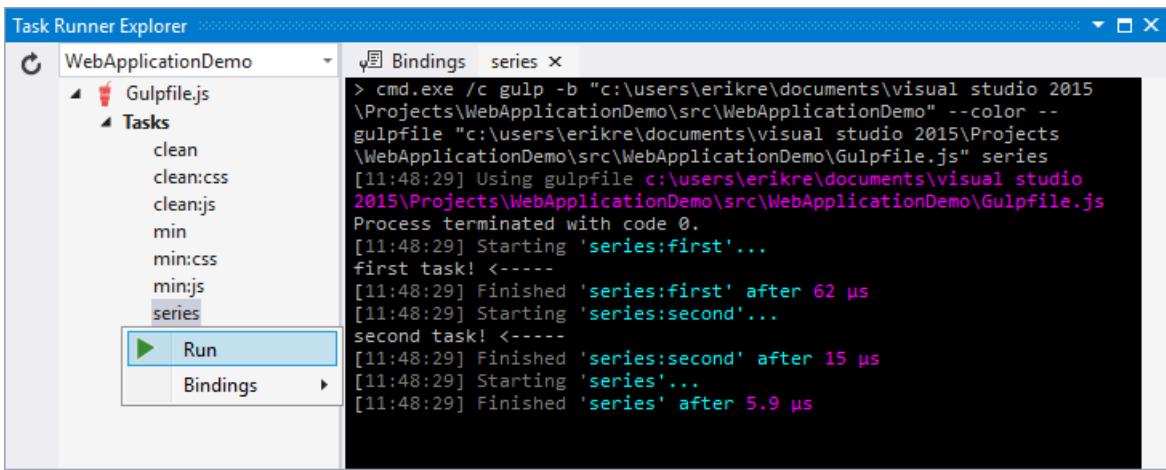
```
gulp.task('series:first', done => {
  console.log('first task! <----');
  done(); // signal completion
});
gulp.task('series:second', done => {
  console.log('second task! <----');
  done(); // signal completion
});

gulp.task('series', gulp.series(['series:first', 'series:second']), () => { });

// A 'default' task is required by Gulp v4
gulp.task('default', gulp.series('series'));
```

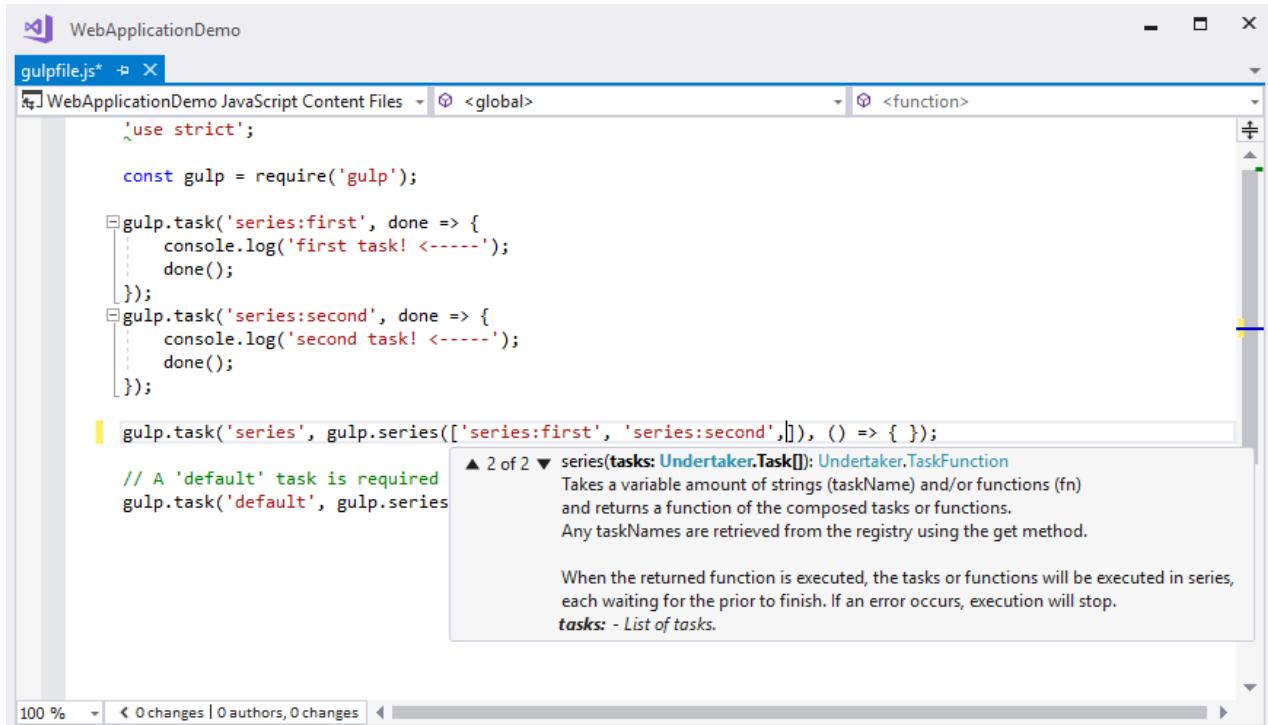
Agora você tem três tarefas: `series:first`, `series:second`, e `series`. O `series:second` tarefa inclui um segundo parâmetro que especifica uma matriz de tarefas para serem executados e concluídos antes do `series:second` tarefa será executada. Conforme especificado no código acima, somente o `series:first` tarefa deve ser concluída antes do `series:second` tarefa será executada.

2. Salve `gulpfile.js`.
3. Na **Gerenciador de soluções**, clique com botão direito `gulpfile.js` e selecione **Task Runner Explorer** se ainda não estiver aberto.
4. Na **Task Runner Explorer**, clique com botão direito **série** e selecione **executar**.



## IntelliSense

IntelliSense fornece conclusão de código, descrições de parâmetro e outros recursos para aumentar a produtividade e diminuir a erros. As tarefas de gulp são escritas em JavaScript; Portanto, o IntelliSense pode fornecer assistência durante o desenvolvimento. Conforme você trabalha com JavaScript, o IntelliSense lista os objetos, funções, propriedades e parâmetros que estão disponíveis com base em seu contexto atual. Selecione uma opção de codificação na lista pop-up fornecida pelo IntelliSense para concluir o código.



Para obter mais informações sobre o IntelliSense, consulte [JavaScript IntelliSense](#).

## Ambientes de desenvolvimento, preparo e produção

Quando o Gulp é usado para otimizar arquivos do lado do cliente para produção e preparo, os arquivos processados são salvos em um local de preparo e produção local. O `layout.cshtml` arquivo de usar o **ambiente** auxiliar para fornecer duas versões diferentes de arquivos CSS de marcação. Uma versão de arquivos CSS é para o desenvolvimento e a outra versão é otimizada para produção e preparo. No Visual Studio 2017, quando você altera a **ASPNETCORE\_ENVIRONMENT** variável de ambiente `Production`, Visual Studio criará o aplicativo Web e um link para os arquivos CSS minimizados. A marcação a seguir mostra a **ambiente** que contém as marcações de link para os auxiliares de marca a `Development` CSS arquivos e o minificado `Staging, Production` arquivos CSS.

```

<environment names="Development">
  <script src="~/lib/jquery/dist/jquery.js"></script>
  <script src="~/lib/bootstrap/dist/js/bootstrap.js"></script>
  <script src="~/js/site.js" asp-append-version="true"></script>
</environment>
<environment names="Staging,Production">
  <script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-2.2.0.min.js"
    asp-fallback-src="~/lib/jquery/dist/jquery.min.js"
    asp-fallback-test="window.jQuery"
    crossorigin="anonymous"
    integrity="sha384-K+ctZQ+LL8q6tP7I94W+qzQsfRV2a+AfhII9k8z8l9ggpc8X+Ytst4yBo/hH+8Fk">
  </script>
  <script src="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/bootstrap.min.js"
    asp-fallback-src="~/lib/bootstrap/dist/js/bootstrap.min.js"
    asp-fallback-test="window.jQuery && window.jQuery.fn && window.jQuery.fn.modal"
    crossorigin="anonymous"
    integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuiWVxZxUPnCJA7l2mCWNIPG9mGCD8wGNICPD7Txa">
  </script>
  <script src="~/js/site.min.js" asp-append-version="true"></script>
</environment>

```

## Alternância entre ambientes

Para alternar entre a compilação para ambientes diferentes, modifique a **ASPNETCORE\_ENVIRONMENT** valor da variável de ambiente.

1. No **Task Runner Explorer**, verifique se que o **min** foi definida para executar tarefas **antes da compilação**.
2. Na **Gerenciador de soluções**, clique com botão direito no nome do projeto e selecione **propriedades**.  
A folha de propriedades para o aplicativo Web é exibida.
3. Clique na guia **Depurar**.
4. Definir o valor de : **ambiente de hospedagem** variável de ambiente **Production**.
5. Pressione **F5** para executar o aplicativo em um navegador.
6. Na janela do navegador, a página com o botão direito e selecione **Exibir código-fonte** para exibir o HTML da página.  
Observe que os links de folha de estilos apontam para os arquivos CSS minificados.
7. Feche o navegador para parar o aplicativo Web.
8. No Visual Studio, retorne para a folha de propriedades para o aplicativo Web e altere o : **ambiente de hospedagem** variável de ambiente de volta para **Development**.
9. Pressione **F5** para executar o aplicativo em um navegador novamente.
10. Na janela do navegador, a página com o botão direito e selecione **Exibir código-fonte** para ver o HTML da página.  
Observe que os links de folha de estilos apontam para as versões unminified dos arquivos CSS.

Para obter mais informações relacionadas aos ambientes no ASP.NET Core, consulte [usar vários ambientes](#).

## Detalhes da tarefa e o módulo

Uma tarefa Gulp está registrada com um nome de função. É possível especificar dependências se outras tarefas devem ser executadas antes da tarefa atual. Funções adicionais permitem que você execute e assistir as tarefas de

Gulp, bem como definir a origem (*src*) e de destino (*dest*) dos arquivos que está sendo modificados. Estas são as funções de API do Gulp primárias:

FUNÇÃO GULP	SINTAXE	DESCRIÇÃO
tarefa	<pre>gulp.task(name[, deps], fn) { }</pre>	O <code>task</code> função cria uma tarefa. O <code>name</code> parâmetro define o nome da tarefa. O <code>deps</code> parâmetro contém uma matriz de tarefas a serem concluídas antes que essa tarefa é executada. O <code>fn</code> parâmetro representa uma função de retorno de chamada que executa as operações da tarefa.
Inspeção	<pre>gulp.watch(glob [, opts], tasks) { }</pre>	O <code>watch</code> function monitora arquivos e execuções de tarefas quando ocorre uma alteração de arquivo. O <code>glob</code> parâmetro é um <code>string</code> ou <code>array</code> que determina quais arquivos assistir. O <code>opts</code> parâmetro fornece monitoramento opções de arquivo adicional.
src	<pre>gulp.src(globs[, options]) { }</pre>	O <code>src</code> função fornece os arquivos que correspondem os valores de glob. O <code>glob</code> parâmetro é um <code>string</code> ou <code>array</code> que determina quais arquivos para ler. O <code>options</code> parâmetro fornece opções de arquivo adicionais.
dest	<pre>gulp.dest(path[, options]) { }</pre>	O <code>dest</code> função define um local para o qual os arquivos podem ser gravados. O <code>path</code> parâmetro é uma cadeia de caracteres ou uma função que determina a pasta de destino. O <code>options</code> parâmetro é um objeto que especifica as opções de pasta de saída.

Para obter informações de referência de API Gulp, consulte [Gulp Docs API](#).

## Receitas do gulp

A comunidade do Gulp fornece Gulp [receitas](#). Essas receitas consistem em tarefas de Gulp para lidar com cenários comuns.

## Recursos adicionais

- [Documentação do gulp](#)
- [Agrupamento e minificação no ASP.NET Core](#)
- [Usar o Grunt no ASP.NET Core](#)

# Use o assistente no núcleo do ASP.NET

22/06/2018 • 17 minutes to read • [Edit Online](#)

Por [Noel arroz](#)

Pesado é um executor de tarefas do JavaScript que automatiza minimização de script, a compilação TypeScript, ferramentas de "pano" de qualidade do código, pré-processadores de CSS e praticamente qualquer tarefa repetitiva que precisa fazer para dar suporte ao desenvolvimento de cliente. Pesado tem suporte total no Visual Studio, embora os modelos de projeto do ASP.NET usam Gulp por padrão (consulte [usar Gulp](#)).

Este exemplo usa um projeto vazio do ASP.NET Core como ponto de partida, para mostrar como automatizar o processo de compilação do cliente desde o início.

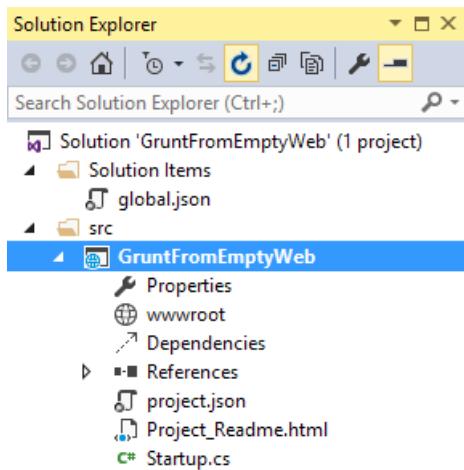
O exemplo concluído limpa o diretório de implantação de destino, combina arquivos JavaScript, verifica a qualidade do código, condensa o conteúdo do arquivo JavaScript e implanta para a raiz do seu aplicativo web. Usaremos os seguintes pacotes:

- **Assistente de**: pacote de executor de tarefas a pesado.
- **Limpeza de Contribuidor pesado**: um plug-in que remove os arquivos ou diretórios.
- **Assistente de Contribuidor de jshint**: um plug-in que analisa a qualidade do código JavaScript.
- **Assistente de Contribuidor de concat**: um plug-in que une os arquivos em um único arquivo.
- **uglify pesado-Contribuidor**: um plug-in que minimiza o JavaScript para reduzir o tamanho.
- **Observação de Contribuidor pesado**: um plug-in que observa a atividade de arquivos.

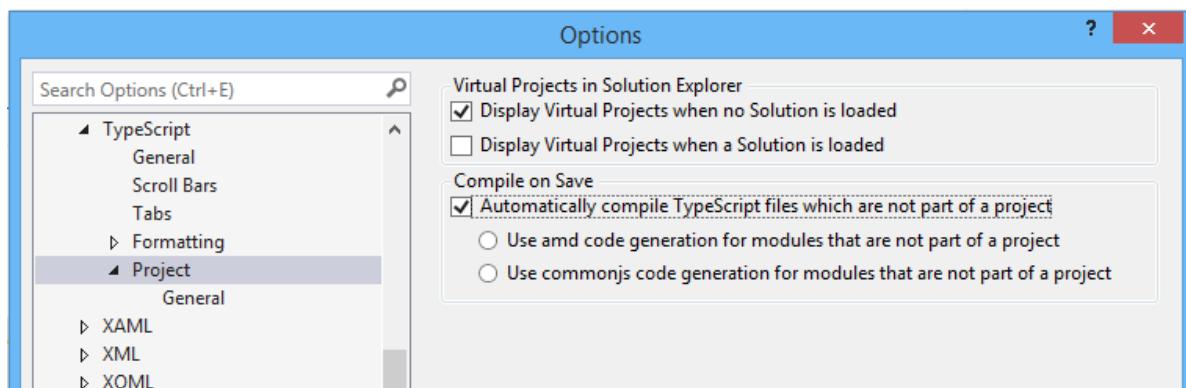
## Preparando o aplicativo

Para começar, configure um novo aplicativo web vazio e adicionar arquivos de exemplo de TypeScript. Arquivos typeScript são compilados automaticamente para JavaScript usando as configurações do Visual Studio e serão nossos matérias-primas para processar usando o assistente.

1. No Visual Studio, crie um novo `ASP .NET Web Application`.
2. No **novo projeto ASP.NET** caixa de diálogo, selecione o ASP.NET Core **vazio** modelo e clique no botão **Okey**.
3. No Gerenciador de soluções, revise a estrutura do projeto. O `\src` pasta inclui vazio `wwwroot` e `Dependencies` nós.



4. Adicionar uma nova pasta chamada **TypeScript** para o diretório do projeto.
5. Antes de adicionar todos os arquivos, certifique-se de que o Visual Studio tem a opção 'Compilar ao salvar' para arquivos TypeScript check. Navegue até **ferramentas > opções > Editor de texto > Typescript > Projeto:**



6. Clique com botão direito do **TypeScript** diretório e selecione **Adicionar > Novo Item** no menu de contexto. Selecione o **arquivo JavaScript** item e nomeie o arquivo *Tastes.ts* (Observe o \*extensão. TS). Copie a linha de código do TypeScript abaixo no arquivo (quando você salva, um novo *Tastes.js* arquivo aparecerá com a origem de JavaScript).

```
enum Tastes { Sweet, Sour, Salty, Bitter }
```

7. Adicionar um segundo arquivo para o **TypeScript** diretório e nomeie-o **Food.ts**. Copie o código abaixo para o arquivo.

```

class Food {
    constructor(name: string, calories: number) {
        this._name = name;
        this._calories = calories;
    }

    private _name: string;
    get Name() {
        return this._name;
    }

    private _calories: number;
    get Calories() {
        return this._calories;
    }

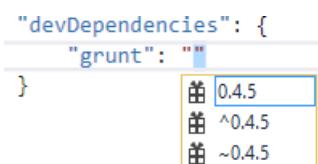
    private _taste: Tastes;
    get Taste(): Tastes { return this._taste }
    set Taste(value: Tastes) {
        this._taste = value;
    }
}

```

## Configurando NPM

Em seguida, configure NPM para baixar pesado e tarefas do assistente.

1. No Gerenciador de soluções, clique com o botão direito e selecione **Adicionar > Novo Item** no menu de contexto. Selecione o **arquivo de configuração NPM** item, deixe o nome padrão, *Package. JSON* e clique no **adicionar** botão.
2. No *Package. JSON* arquivo, dentro de `devDependencies` chaves de objeto, digite "pesado". Selecione `grunt` o Intellisense de lista e pressione a tecla Enter. Visual Studio será colocada entre aspas no nome do pacote pesado e adicionar dois-pontos. À direita dos dois pontos, selecione a versão estável mais recente do pacote da parte superior da lista do Intellisense (pressione `Ctrl-Space` se o Intellisense não aparece).



### NOTE

Usa NPM [controle de versão semântico](#) para organizar as dependências. Controle de versão semântico, também conhecido como SemVer, identifica os pacotes com o esquema de numeração ... IntelliSense simplifica o controle de versão semântico, mostrando apenas algumas opções comuns. O item superior na lista do Intellisense (0.4.5 no exemplo acima) é considerado a versão estável mais recente do pacote. O símbolo de acento circunflexo (^) corresponde a mais recente versão principal e o til (~) corresponde a versão secundária mais recente. Consulte o [referência de analisador NPM semver versão](#) como um guia para a expressividade completa que fornece SemVer.

3. Adicionar mais dependências carregar pesadom-Contribuidor -\* pacotes para *limpa*, *jshint*, *concat*, *uglifye* *inspecionar* conforme mostrado no exemplo a seguir. As versões não precisam coincidir com o exemplo.

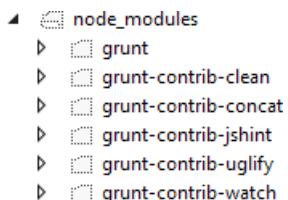
```

"devDependencies": {
  "grunt": "0.4.5",
  "grunt-contrib-clean": "0.6.0",
  "grunt-contrib-jshint": "0.11.0",
  "grunt-contrib-concat": "0.5.1",
  "grunt-contrib-uglify": "0.8.0",
  "grunt-contrib-watch": "0.6.1"
}

```

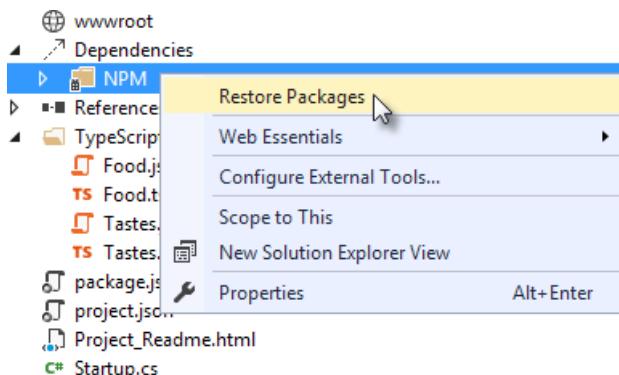
#### 4. Salve o *Package.json* arquivo.

Baixarão os pacotes para cada item devDependencies, juntamente com todos os arquivos que requer que cada pacote. Você pode encontrar os arquivos de pacote no `node_modules` diretório, permitindo que o **Mostrar todos os arquivos** botão no Gerenciador de soluções.



#### NOTE

Se você precisar, você pode restaurar manualmente as dependências no Gerenciador de soluções clicando em `Dependencies\NPM` e selecionando o **restaurar pacotes** opção de menu.



## Assistente de configuração

Pesado estiver configurado para usar um manifesto chamado *Gruntfile.js* que define, carrega e registra as tarefas que podem ser executadas manualmente ou configuradas para ser executado automaticamente com base em eventos no Visual Studio.

1. Clique com o botão direito e selecione **Adicionar > Novo Item**. Selecione o **arquivo de configuração Grunt** opção, deixe o nome padrão, *Gruntfile.js* e clique no **adicionar** botão.

O código inicial inclui uma definição de módulo e o `grunt.initConfig()` método. O `initConfig()` é usado para definir opções para cada pacote, e o restante do módulo serão carregadas e registrar tarefas.

```

module.exports = function (grunt) {
  grunt.initConfig({
  });
};

```

2. Dentro de `initConfig()` método, adicionar opções para o `clean` conforme mostrado no exemplo de

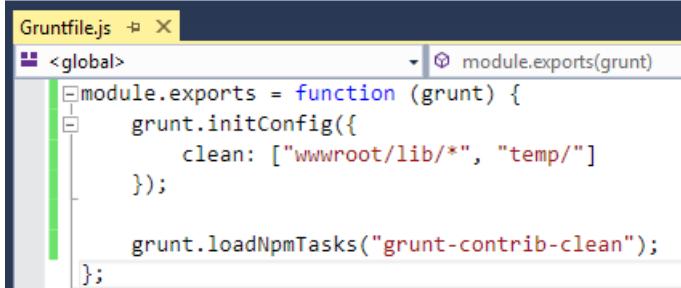
tarefa *Gruntfile.js* abaixo. A tarefa de limpeza aceita uma matriz de cadeias de caracteres de diretório. Essa tarefa remove arquivos de `wwwroot/lib` e o diretório `temp/inteiro`.

```
module.exports = function (grunt) {
  grunt.initConfig({
    clean: ["wwwroot/lib/*", "temp/"],
  });
};
```

3. Abaixo do método `initConfig()`, adicione uma chamada para `grunt.loadNpmTasks()`. Isso fará a tarefa executável do Visual Studio.

```
  grunt.loadNpmTasks("grunt-contrib-clean");
```

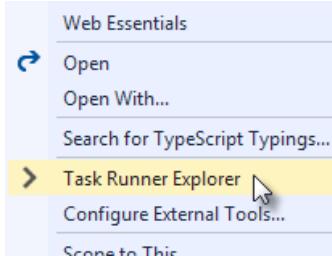
4. Salvar *Gruntfile.js*. O arquivo deve ser semelhante a captura de tela abaixo.



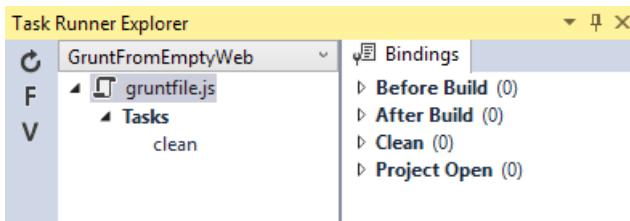
```
Gruntfile.js  X
global < module.exports(grunt)
module.exports = function (grunt) {
  grunt.initConfig({
    clean: ["wwwroot/lib/*", "temp/"]
  });

  grunt.loadNpmTasks("grunt-contrib-clean");
};
```

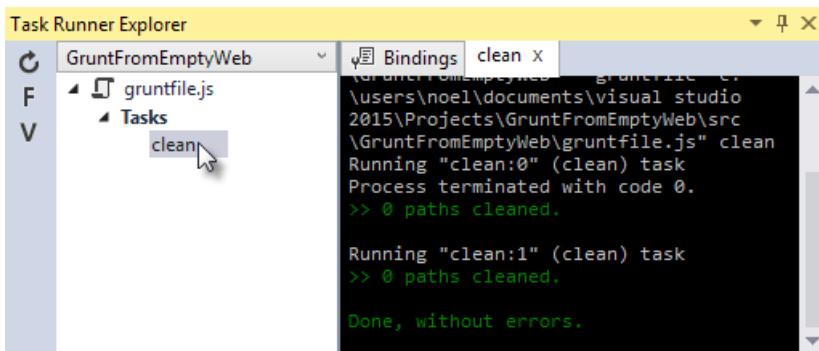
5. Clique com botão direito *Gruntfile.js* e selecione **Explorador do Executador de tarefas** no menu de contexto. A janela Explorador do Executador de tarefas será aberto.



6. Verifique `clean` mostra em **tarefas** no Explorador do Executador de tarefas.



7. A tarefa de limpeza e selecione **executar** no menu de contexto. Uma janela de comando exibe o progresso da tarefa.



#### NOTE

Não existem arquivos ou diretórios para limpar ainda. Se desejar, você pode criá-las manualmente no Gerenciador de soluções e, em seguida, executar a tarefa de limpeza como um teste.

8. No método initConfig(), adicione uma entrada para `concat` usando o código abaixo.

O `src` matriz de propriedade lista arquivos combinar, na ordem em que eles devem ser combinados. O `dest` propriedade atribui o caminho para o arquivo combinado que é produzido.

```
concat: {  
  all: {  
    src: ['TypeScript/Tastes.js', 'TypeScript/Food.js'],  
    dest: 'temp/combined.js'  
  }  
},
```

#### NOTE

O `all` propriedade no código acima é o nome de um destino. Destinos são usados em algumas tarefas pesado para permitir que vários ambientes de desenvolvimento. Você pode exibir os destinos internos usando o Intellisense ou atribuir seu próprio.

9. Adicionar o `jshint` tarefas usando o código abaixo.

O utilitário de qualidade do código jshint é executado em todos os arquivos JavaScript encontrado no diretório temp.

```
jshint: {  
  files: ['temp/*.js'],  
  options: {  
    '-W069': false,  
  }  
},
```

#### NOTE

A opção "-W069" é um erro gerado pelo jshint quando colchete de JavaScript usa a sintaxe para atribuir uma propriedade em vez da notação de ponto, ou seja, `Tastes["Sweet"]` em vez de `Tastes.Sweet`. A opção desativa o aviso para permitir que o restante do processo para continuar.

10. Adicionar o `uglify` tarefas usando o código abaixo.

A tarefa minimiza o `combined.js` arquivo encontrado no diretório temp e cria o arquivo de resultado no

wwwroot/lib seguindo a convenção de nomenclatura padrão <nome de arquivo>.min.js.

```
uglify: {
  all: {
    src: ['temp/combined.js'],
    dest: 'wwwroot/lib/combined.min.js'
  }
},
```

11. Sob o grunt.loadNpmTasks() de chamada que carrega a limpeza de Contribuidor pesado, incluem a mesma chamada para jshint, concat e uglify usando o código abaixo.

```
grunt.loadNpmTasks('grunt-contrib-jshint');
grunt.loadNpmTasks('grunt-contrib-concat');
grunt.loadNpmTasks('grunt-contrib-uglify');
```

12. Salvar Gruntfile.js. O arquivo deve ser algo como o exemplo a seguir.



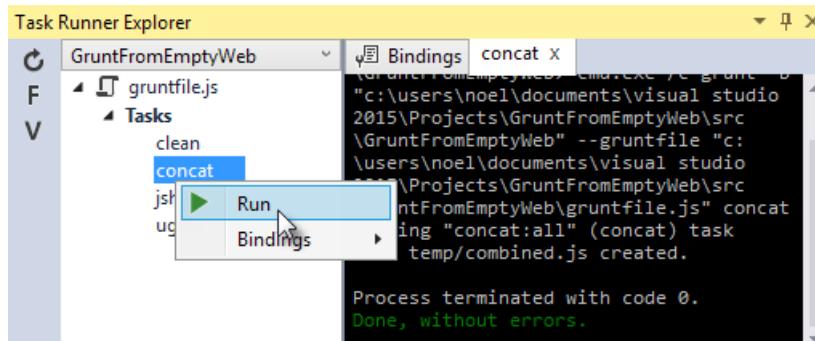
```
Gruntfile.js
module.exports = function (grunt) {
  grunt.initConfig({
    clean: ["wwwroot/lib/*", "temp/*"],
    concat: {
      all: {
        src: ['TypeScript/Tastes.js', 'TypeScript/Food.js'],
        dest: 'temp/combined.js'
      }
    },
    jshint: { files: ['temp/*.js'], options: { '-W069': false } },
    uglify: {
      all: {
        src: ['temp/combined.js'],
        dest: 'wwwroot/lib/combined.min.js'
      }
    }
  });

  grunt.loadNpmTasks('grunt-contrib-clean');
  grunt.loadNpmTasks('grunt-contrib-jshint');
  grunt.loadNpmTasks('grunt-contrib-concat');
  grunt.loadNpmTasks('grunt-contrib-uglify');
};

110 %
```

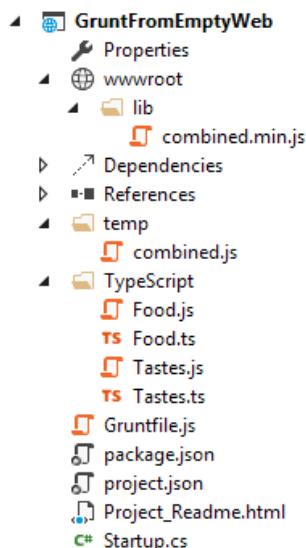
13. Observe que a lista de tarefas do Gerenciador de executor inclui `clean`, `concat`, `jshint` e `uglify` tarefas.

Execute cada tarefa na ordem e observar os resultados no Gerenciador de soluções. Cada tarefa deve ser executada sem erros.



A tarefa concat cria um novo `combined.js` de arquivo e o coloca na pasta temp. A tarefa de jshint

simplesmente executa e não produz saída. A tarefa uglify cria um novo *combined.min.js* de arquivo e o coloca na wwwroot/lib. Após a conclusão, a solução deve ser semelhante a captura de tela abaixo:



#### NOTE

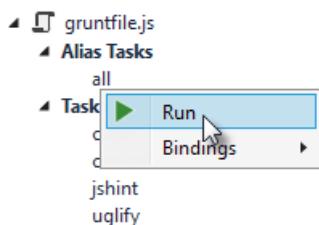
Para obter mais informações sobre as opções para cada pacote, visite <https://www.npmjs.com/> e o nome do pacote na caixa de pesquisa na página principal de pesquisa. Por exemplo, você pode pesquisar o pacote limpeza de Contribuidor pesado para obter um link de documentação que explica a todos os seus parâmetros.

### Todos juntos agora

Use o Assistente de `registerTask()` método para executar uma série de tarefas em uma determinada sequência. Por exemplo, para executar o exemplo etapas acima na ordem limpa -> concat -> jshint -> uglify, adicione o código abaixo para o módulo. O código deve ser adicionado no mesmo nível como as chamadas `loadNpmTasks()`, fora `initConfig`.

```
grunt.registerTask("all", ['clean', 'concat', 'jshint', 'uglify']);
```

A nova tarefa aparece no Explorador do Executador de tarefas em tarefas de Alias. Pode-se com o botão direito e executá-lo como faria com outras tarefas. O `all` tarefa executará `clean`, `concat`, `jshint` e `uglify`, em ordem.



## Monitorando alterações

Um `watch` tarefa fica de olho em arquivos e diretórios. O relógio dispara tarefas automaticamente se detectar alterações. Adicione o código abaixo para `initConfig` para observar as alterações \*no diretório `TypeScript.js`. Se um arquivo JavaScript for alterado, `watch` executará o `all` tarefa.

```
watch: {
  files: ["TypeScript/*.js"],
  tasks: ["all"]
}
```

Adicionar uma chamada para `loadNpmTasks()` para mostrar o `watch` tarefa no Explorador do Executador de tarefas.

```
grunt.loadNpmTasks('grunt-contrib-watch');
```

Clique na tarefa de inspeção no Explorador do Executador de tarefas e selecione Executar no menu de contexto. A janela de comando que mostra a execução de tarefa watch exibirá um "Aguardando..." . Abra um dos arquivos TypeScript, adicione um espaço e, em seguida, salve o arquivo. Isso disparar a tarefa de inspeção e disparar outras tarefas executadas em ordem. Captura de tela abaixo mostra uma exemplo de execução.

```
Bindings watch (running) X
2015\Projects\GruntFromEmptyWeb\src
\GruntFromEmptyWeb" --gruntfile "c:
\users\noel\documents\visual studio
2015\Projects\GruntFromEmptyWeb\src
\GruntFromEmptyWeb\gruntfile.js" watch
Running "watch" task
Waiting...
>> File "TypeScript\Tastes.js" changed.
Running "clean:0" (clean) task
>> 1 path cleaned.

Running "clean:1" (clean) task
>> 1 path cleaned.

Running "concat:all" (concat) task
File temp/combined.js created.

Running "jshint:files" (jshint) task
>> 1 file lint free.

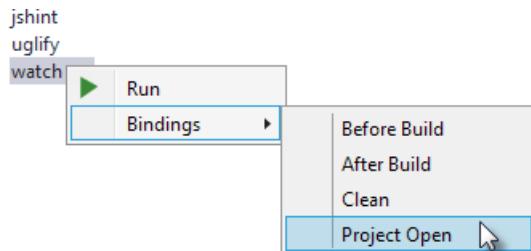
Running "uglify:all" (uglify) task
>> 1 file created.

Done, without errors.
Completed in 1.236s at Fri Mar 13 2015
17:12:36 GMT-0700 (Pacific Daylight
Time) - Waiting...
```

## Associação a eventos do Visual Studio

A menos que você deseja iniciar manualmente as tarefas toda vez que você trabalha no Visual Studio, você pode associar tarefas a serem **antes de criar, depois de criar, limpar**, e **Projeto aberto** eventos.

Vamos associar `watch` para que ele seja executado sempre que o Visual Studio abrirá. No Explorador do Executador de tarefas, a tarefa de inspeção e selecione **associações > Abrir projeto** no menu de contexto.



Descarregar e recarregar o projeto. Quando o projeto é carregado novamente, a tarefa de inspeção iniciará a execução automaticamente.

## Resumo

Pesado é um executor de tarefa avançada que pode ser usado para automatizar a maioria das tarefas de compilação do cliente. Pesado aproveita NPM para fornecer seus pacotes e recursos de ferramentas de integração com o Visual Studio. Explorador do Executador de tarefas do Visual Studio detecta alterações nos arquivos de configuração e fornece uma interface conveniente para executar tarefas, exibir tarefas em execução e associar tarefas a eventos do Visual Studio.

## Recursos adicionais

- [Usar o Gulp](#)

# Aquisição de biblioteca do lado do cliente no ASP.NET Core com LibMan

31/08/2018 • 2 minutes to read • [Edit Online](#)

Por [Scott Addie](#)

O Gerenciador de Bibliotecas (LibMan) é uma ferramenta leve de aquisição de bibliotecas do lado do cliente. O LibMan baixa bibliotecas e estruturas populares do sistema de arquivos ou de uma [CDN \(rede de distribuição de conteúdo\)](#). As CDNs compatíveis incluem [CDNJS](#) e [unpkg](#). Os arquivos de biblioteca selecionados são buscados e colocados no local adequado dentro do projeto do ASP.NET Core.

## Casos de uso do LibMan

O LibMan oferece os seguintes benefícios:

- Somente os arquivos de biblioteca necessários são baixados.
- Ferramentas adicionais, tais como [Node.js](#), [npm](#) e [WebPack](#), não são necessárias para adquirir um subconjunto de arquivos em uma biblioteca.
- Arquivos podem ser colocados em um local específico sem recorrer a tarefas de build ou à cópia manual de arquivos.

Para obter mais informações sobre os benefícios do LibMan, assista a [Modern front-end web development in Visual Studio 2017: LibMan segment](#) (Desenvolvimento de front-end da Web moderno no Visual Studio 2017: segmento LibMan).

O LibMan não é um sistema de gerenciamento de pacotes. Se você já está usando um gerenciador de pacotes, assim como o npm ou [yarn](#), continue fazendo isso. O LibMan não foi desenvolvido para substituir essas ferramentas.

## Recursos adicionais

- [LibMan de uso com o ASP.NET Core no Visual Studio](#)
- [Use a interface de linha de comando LibMan \(CLI\) com o ASP.NET Core](#)
- [Repositório do GitHub do LibMan](#)

# Use a interface de linha de comando LibMan (CLI) com o ASP.NET Core

27/09/2018 • 14 minutes to read • [Edit Online](#)

Por [Scott Addie](#)

O [LibMan](#) CLI é uma ferramenta de plataforma cruzada que tem suporte em qualquer lugar, há suporte para .NET Core.

## Pré-requisitos

- [SDK do .NET Core 2.1 ou posteriores](#)

## Instalação

Para instalar a CLI LibMan:

```
dotnet tool install -g Microsoft.Web.LibraryManager.Cli
```

Um [ferramenta Global do .NET Core](#) for instalado a partir de [Microsoft.Web.LibraryManager.Cli](#) pacote do NuGet.

Para instalar a CLI LibMan de uma fonte específica de pacote do NuGet:

```
dotnet tool install -g Microsoft.Web.LibraryManager.Cli --version 1.0.94-g606058a278 --add-source C:\Temp\
```

No exemplo anterior, uma ferramenta Global do .NET Core é instalada do computador Windows local C:\Temp\Microsoft.Web.LibraryManager.Cli.1.0.94-g606058a278.nupkg arquivo.

## Uso

Após a instalação bem-sucedida da CLI, o comando a seguir pode ser usado:

```
libman
```

Para exibir a versão da CLI instalada:

```
libman --version
```

Para exibir os comandos da CLI disponíveis:

```
libman --help
```

O comando anterior exibe uma saída semelhante à seguinte:

```

1.0.163+g45474d37ed

Usage: libman [options] [command]

Options:
  --help|-h Show help information
  --version Show version information

Commands:
  cache      List or clean libman cache contents
  clean       Deletes all library files defined in libman.json from the project
  init        Create a new libman.json
  install     Add a library definition to the libman.json file, and download the
              library to the specified location
  restore     Downloads all files from provider and saves them to specified
              destination
  uninstall   Deletes all files for the specified library from their specified
              destination, then removes the specified library definition from
              libman.json
  update      Updates the specified library

Use "libman [command] --help" for more information about a command.

```

As seções a seguir descrevem os comandos da CLI disponíveis.

## Iniciar LibMan no projeto

O `libman init` comando cria um `libman.json` arquivo se ele não existir. O arquivo é criado com o conteúdo do modelo de item padrão.

### Sinopse

```

libman init [-d|--default-destination] [-p|--default-provider] [--verbosity]
libman init [-h|--help]

```

### Opções

As seguintes opções estão disponíveis para o `libman init` comando:

- `-d|--default-destination <PATH>`

Um caminho relativo à pasta atual. Arquivos de biblioteca são instalados nesse local, se nenhum `destination` propriedade está definida para uma biblioteca no `libman.json`. O `<PATH>` valor é gravado para o `defaultDestination` propriedade de `libman.json`.

- `-p|--default-provider <PROVIDER>`

O provedor a ser usado se nenhum provedor é definido para uma determinada biblioteca. O `<PROVIDER>` valor é gravado para o `defaultProvider` propriedade de `libman.json`. Substitua `<PROVIDER>` com um dos seguintes valores:

- `cdnjs`
- `filesystem`
- `unpkg`

- `-h|--help`

Mostre informações de Ajuda.

- `--verbosity <LEVEL>`

Defina o detalhamento da saída. Substitua <LEVEL> com um dos seguintes valores:

- quiet
- normal
- detailed

## Exemplos

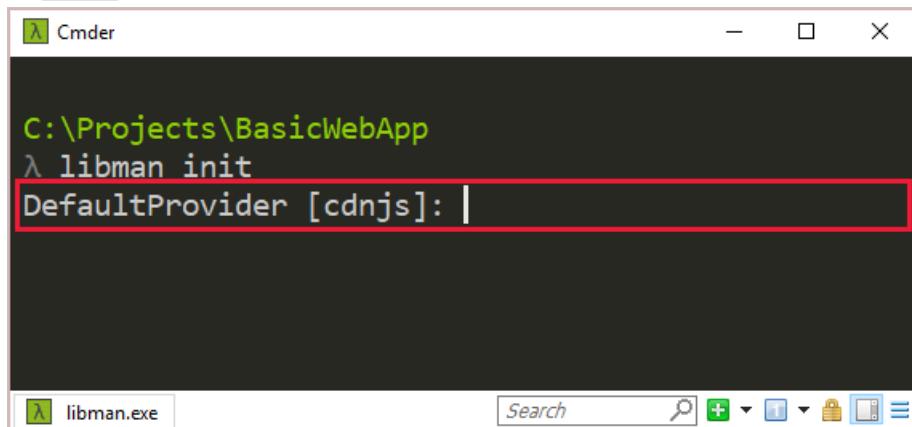
Para criar uma *libman.json* arquivo em um projeto ASP.NET Core:

- Navegue até a raiz do projeto.
- Execute o seguinte comando:

```
libman init
```

- Digite o nome do provedor padrão, ou pressione `Enter` para usar o provedor do CDNJS padrão. Os valores válidos incluem:

- cdnjs
- filesystem
- unpkg



Um *libman.json* arquivo é adicionado à raiz do projeto com o seguinte conteúdo:

```
{
  "version": "1.0",
  "defaultProvider": "cdnjs",
  "libraries": []
}
```

## Adicione arquivos de biblioteca

O `libman install` comando baixa e instala os arquivos de biblioteca no projeto. Um *libman.json* arquivo é adicionado, se ele existir. O *libman.json* arquivo é modificado para armazenar detalhes de configuração para os arquivos de biblioteca.

### Sinopse

```
libman install <LIBRARY> [-d|--destination] [--files] [-p|--provider] [--verbosity]
libman install [-h|--help]
```

### Arguments

## LIBRARY

O nome da biblioteca para instalar. Esse nome pode incluir a notação de número de versão (por exemplo, `@1.2.0`).

## Opções

As seguintes opções estão disponíveis para o `libman install` comando:

- `-d|--destination <PATH>`

O local para instalar a biblioteca. Se não for especificado, o local padrão é usado. Se nenhuma `defaultDestination` propriedade é especificada em `libman.json`, essa opção é necessária.

- `--files <FILE>`

Especifique o nome do arquivo para instalá-lo da biblioteca. Se não for especificado, todos os arquivos da biblioteca são instalados. Forneça uma `--files` opção por arquivo para ser instalado. Caminhos relativos são suportados também. Por exemplo: `--files dist/browser/signalr.js`.

- `-p|--provider <PROVIDER>`

O nome do provedor a ser usado para a aquisição de biblioteca. Substitua `<PROVIDER>` com um dos seguintes valores:

- `cdnjs`
- `filesystem`
- `unpkg`

Se não for especificado, o `defaultProvider` propriedade na `libman.json` é usado. Se nenhuma `defaultProvider` propriedade é especificada em `libman.json`, essa opção é necessária.

- `-h|--help`

Mostre informações de Ajuda.

- `--verbosity <LEVEL>`

Defina o detalhamento da saída. Substitua `<LEVEL>` com um dos seguintes valores:

- `quiet`
- `normal`
- `detailed`

## Exemplos

Considere o seguinte `libman.json` arquivo:

```
{  
  "version": "1.0",  
  "defaultProvider": "cdnjs",  
  "libraries": []  
}
```

Para instalar a versão do jQuery 3.2.1 `jQuery` do arquivo para o `wwwroot/scripts/jquery` pasta usando o provedor do CDNJS:

```
libman install jquery@3.2.1 --provider cdnjs --destination wwwroot/scripts/jquery --files jquery.min.js
```

O `libman.json` arquivo semelhante ao seguinte:

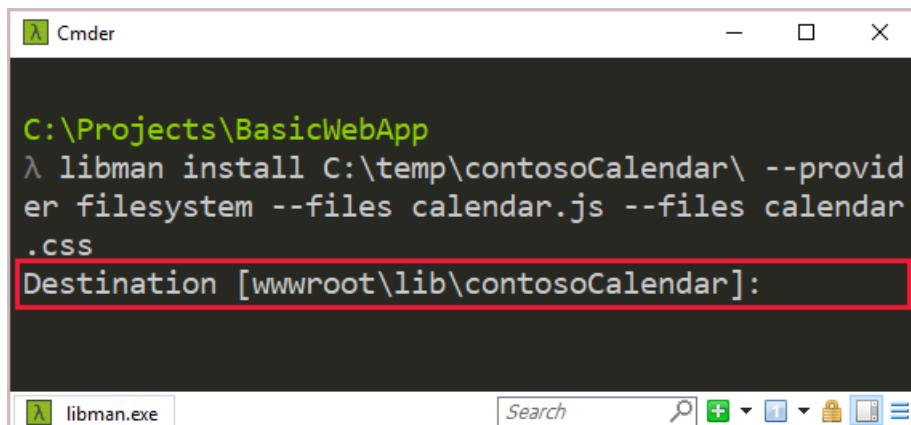
```
{  
  "version": "1.0",  
  "defaultProvider": "cdnjs",  
  "libraries": [  
    {  
      "library": "jquery@3.2.1",  
      "destination": "wwwroot/scripts/jquery",  
      "files": [  
        "jquery.min.js"  
      ]  
    }  
  ]  
}
```

Para instalar o *calendar.js* e *calendar.css* arquivos da *c:\temp\contosoCalendar\* usando o sistema de arquivos provedor:

```
libman install C:\temp\contosoCalendar\ --provider filesystem --files calendar.js --files calendar.css
```

O seguinte aviso é exibido por dois motivos:

- O *libman.json* arquivo não contém um `defaultDestination` propriedade.
- O `libman install` comando não contém o `-d|--destination` opção.



```
C:\Projects\BasicWebApp  
λ libman install C:\temp\contosoCalendar\ --provider filesystem --files calendar.js --files calendar.css  
Destination [wwwroot\lib\contosoCalendar]:
```

Depois de aceitar o destino padrão, o *libman.json* arquivo semelhante ao seguinte:

```
{  
  "version": "1.0",  
  "defaultProvider": "cdnjs",  
  "libraries": [  
    {  
      "library": "jquery@3.2.1",  
      "destination": "wwwroot/scripts/jquery",  
      "files": [  
        "jquery.min.js"  
      ]  
    },  
    {  
      "library": "C:\\temp\\contosoCalendar\\",  
      "provider": "filesystem",  
      "destination": "wwwroot/lib/contosoCalendar",  
      "files": [  
        "calendar.js",  
        "calendar.css"  
      ]  
    }  
  ]  
}
```

## Restaurar arquivos de biblioteca

O `libman restore` comando instala arquivos de biblioteca definidos nas `libman.json`. As seguintes regras se aplicam:

- Se nenhum `libman.json` arquivo existir na raiz do projeto, um erro será retornado.
- Se uma biblioteca Especifica um provedor, o `defaultProvider` propriedade na `libman.json` será ignorado.
- Se uma biblioteca Especifica um destino, o `defaultDestination` propriedade na `libman.json` será ignorado.

### Sinopse

```
libman restore [--verbosity]  
libman restore [-h|--help]
```

### Opções

As seguintes opções estão disponíveis para o `libman restore` comando:

- `-h|--help`

Mostra informações de Ajuda.

- `--verbosity <LEVEL>`

Defina o detalhamento da saída. Substitua `<LEVEL>` com um dos seguintes valores:

- `quiet`
- `normal`
- `detailed`

### Exemplos

Para restaurar os arquivos de biblioteca definidos em `libman.json`:

```
libman restore
```

# Excluir arquivos de biblioteca

O `libman clean` comando exclui os arquivos de biblioteca restaurados anteriormente por meio de LibMan. Pastas se tornar vazias depois dessa operação são excluídas. Os arquivos de biblioteca associados configurações na `libraries` propriedade de `libman.json` não são removidos.

## Sinopse

```
libman clean [--verbosity]
libman clean [-h|--help]
```

## Opções

As seguintes opções estão disponíveis para o `libman clean` comando:

- `-h|--help`

Mostre informações de Ajuda.

- `--verbosity <LEVEL>`

Defina o detalhamento da saída. Substitua `<LEVEL>` com um dos seguintes valores:

- `quiet`
- `normal`
- `detailed`

## Exemplos

Para excluir arquivos de biblioteca instalados por meio de LibMan:

```
libman clean
```

# Desinstalar os arquivos de biblioteca

O `libman uninstall` comando:

- Exclui todos os arquivos associados a biblioteca especificada do destino no `libman.json`.
- Remove a configuração de biblioteca associado de `libman.json`.

Ocorre um erro quando:

- Não `libman.json` arquivo existe na raiz do projeto.
- A biblioteca especificada não existe.

Se mais de uma biblioteca com o mesmo nome estiver instalada, solicitado para escolher um.

## Sinopse

```
libman uninstall <LIBRARY> [--verbosity]
libman uninstall [-h|--help]
```

## Arguments

`LIBRARY`

O nome da biblioteca para desinstalar. Esse nome pode incluir a notação de número de versão (por exemplo, `@1.2.0`).

## Opções

As seguintes opções estão disponíveis para o `libman uninstall` comando:

- `-h|--help`

Mostre informações de Ajuda.

- `--verbosity <LEVEL>`

Defina o detalhamento da saída. Substitua `<LEVEL>` com um dos seguintes valores:

- `quiet`
- `normal`
- `detailed`

## Exemplos

Considere o seguinte `libman.json` arquivo:

```
{
  "version": "1.0",
  "defaultProvider": "cdnjs",
  "libraries": [
    {
      "library": "jquery@3.3.1",
      "files": [
        "jquery.min.js",
        "jquery.js",
        "jquery.min.map"
      ],
      "destination": "wwwroot/lib/jquery/"
    },
    {
      "provider": "unpkg",
      "library": "bootstrap@4.1.3",
      "destination": "wwwroot/lib/bootstrap/"
    },
    {
      "provider": "filesystem",
      "library": "C:\\temp\\lodash\\",
      "files": [
        "lodash.js",
        "lodash.min.js"
      ],
      "destination": "wwwroot/lib/lodash/"
    }
  ]
}
```

- Para desinstalar o jQuery, qualquer um dos comandos a seguir bem-sucedida:

```
libman uninstall jquery
```

```
libman uninstall jquery@3.3.1
```

- Para desinstalar os arquivos de Lodash instalados por meio de `filesystem` provedor:

```
libman uninstall C:\\temp\\lodash\\
```

# Atualizar a versão da biblioteca

O `libman update` comando atualiza uma biblioteca instalada via LibMan à versão especificada.

Ocorre um erro quando:

- Não `libman.json` arquivo existe na raiz do projeto.
- A biblioteca especificada não existe.

Se mais de uma biblioteca com o mesmo nome estiver instalada, solicitado para escolher um.

## Sinopse

```
libman update <LIBRARY> [-pre] [--to] [--verbosity]
libman update [-h|--help]
```

## Arguments

`<LIBRARY>`

O nome da biblioteca para atualizar.

## Opções

As seguintes opções estão disponíveis para o `libman update` comando:

- `-pre`

Obter a versão de pré-lançamento mais recente da biblioteca.

- `--to <VERSION>`

Obtenha uma versão específica da biblioteca.

- `-h|--help`

Mostre informações de Ajuda.

- `--verbosity <LEVEL>`

Defina o detalhamento da saída. Substitua `<LEVEL>` com um dos seguintes valores:

- `quiet`
- `normal`
- `detailed`

## Exemplos

- Para atualizar o jQuery para a versão mais recente:

```
libman update jquery
```

- Para atualizar o jQuery versão 3.3.1:

```
libman update jquery --to 3.3.1
```

- Para atualizar o jQuery para a versão de pré-lançamento mais recente:

```
libman update jquery -pre
```

# Gerenciar o cache de biblioteca

O `libman cache` comando gerencia o cache de biblioteca LibMan. O `filesystem` provedor não usa o cache de biblioteca.

## Sinopse

```
libman cache clean [<PROVIDER>] [--verbosity]
libman cache list [--files] [--libraries] [--verbosity]
libman cache [-h|--help]
```

## Arguments

`PROVIDER`

Usado somente com o `clean` comando. Especifica para limpar o cache do provedor. Os valores válidos incluem:

- `cdnjs`
- `filesystem`
- `unpkg`

## Opções

As seguintes opções estão disponíveis para o `libman cache` comando:

- `--files`

Liste os nomes de arquivos que são armazenados em cache.

- `--libraries`

Liste os nomes de bibliotecas que são armazenados em cache.

- `-h|--help`

Mostre informações de Ajuda.

- `--verbosity <LEVEL>`

Defina o detalhamento da saída. Substitua `<LEVEL>` com um dos seguintes valores:

- `quiet`
- `normal`
- `detailed`

## Exemplos

- Para exibir os nomes de bibliotecas em cache por um provedor, use um dos seguintes comandos:

```
libman cache list
```

```
libman cache list --libraries
```

Uma saída semelhante à apresentada a seguir será exibida:

```
Cache contents:
```

```
-----
```

```
unpkg:
```

```
    knockout
    react
    vue
```

```
cdnjs:
```

```
    font-awesome
    jquery
    knockout
    lodash.js
    react
```

- Para exibir os nomes dos arquivos de biblioteca em cache por provedor:

```
libman cache list --files
```

Uma saída semelhante à apresentada a seguir será exibida:

```
Cache contents:
```

```
-----
```

```
unpkg:
```

```
    knockout:
        <list omitted for brevity>
    react:
        <list omitted for brevity>
    vue:
        <list omitted for brevity>
```

```
cdnjs:
```

```
    font-awesome
        metadata.json
    jquery
        metadata.json
        3.2.1\core.js
        3.2.1\jquery.js
        3.2.1\jquery.min.js
        3.2.1\jquery.min.map
        3.2.1\jquery.slim.js
        3.2.1\jquery.slim.min.js
        3.2.1\jquery.slim.min.map
        3.3.1\core.js
        3.3.1\jquery.js
        3.3.1\jquery.min.js
        3.3.1\jquery.min.map
        3.3.1\jquery.slim.js
        3.3.1\jquery.slim.min.js
        3.3.1\jquery.slim.min.map
    knockout
        metadata.json
        3.4.2\knockout-debug.js
        3.4.2\knockout-min.js
    lodash.js
        metadata.json
        4.17.10\lodash.js
        4.17.10\lodash.min.js
    react
        metadata.json
```

Observe que a saída anterior mostra que o jQuery versões 3.2.1 e 3.3.1 são armazenados em cache no provedor de CDNJS.

- Para esvaziar o cache de biblioteca para o provedor do CDNJS:

```
libman cache clean cdnjs
```

Depois de esvaziar o cache do provedor do CDNJS o `libman cache list` comando exibe o seguinte:

```
Cache contents:  
-----  
unpkg:  
    knockout  
    react  
    vue  
cdnjs:  
    (empty)
```

- Para esvaziar o cache para todos os provedores com suporte:

```
libman cache clean
```

Depois de esvaziar todos os caches de provedor, o `libman cache list` comando exibe o seguinte:

```
Cache contents:  
-----  
unpkg:  
    (empty)  
cdnjs:  
    (empty)
```

## Recursos adicionais

- [Instalar uma ferramenta Global](#)
- [LibMan de uso com o ASP.NET Core no Visual Studio](#)
- [Repositório do GitHub do LibMan](#)

# LibMan de uso com o ASP.NET Core no Visual Studio

30/10/2018 • 15 minutes to read • [Edit Online](#)

Por [Scott Addie](#)

O Visual Studio tem suporte interno para [LibMan](#) em projetos do ASP.NET Core, incluindo:

- Suporte para configurar e executar operações de restauração LibMan no build.
- Itens de menu para o disparo de operações de limpeza e LibMan restauração.
- Caixa de diálogo de pesquisa para localizar bibliotecas e adicionar os arquivos a um projeto.
- Suporte de edição para *libman.json*—o arquivo de manifesto LibMan.

[Exibir ou baixar o código de exemplo \(como fazer o download\)](#)

## Pré-requisitos

- Visual Studio 2017 versão 15,8 ou posterior com o **ASP.NET e desenvolvimento web** carga de trabalho

## Adicione arquivos de biblioteca

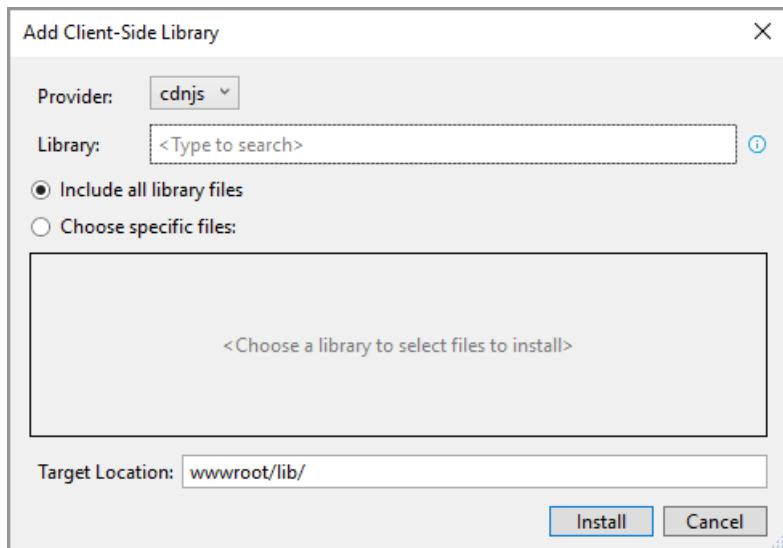
Arquivos de biblioteca podem ser adicionados a um projeto ASP.NET Core de duas maneiras diferentes:

1. [Use a caixa de diálogo Adicionar biblioteca de cliente](#)
2. [Configurar manualmente as entradas do arquivo de manifesto LibMan](#)

### Use a caixa de diálogo Adicionar biblioteca de cliente

Siga estas etapas para instalar uma biblioteca de cliente:

- Na **Gerenciador de soluções**, clique na pasta de projeto no qual os arquivos devem ser adicionados. Escolher **adicone > biblioteca de cliente**. O **adicionar a biblioteca do lado do cliente** caixa de diálogo aparece:



- Selecione o provedor de biblioteca do **provedor** lista suspensa. CDNJS é o provedor padrão.
- Digite o nome da biblioteca para buscar na **biblioteca** caixa de texto. IntelliSense fornece uma lista de bibliotecas, começando com o texto fornecido.

- Selecione a biblioteca da lista do IntelliSense. Observe que o nome da biblioteca é sufixado com o  símbolo e a versão estável mais recente conhecida para o provedor selecionado.
- Decida quais arquivos a serem incluídos:
  - Selecione o **incluir todos os arquivos de biblioteca** botão de opção para incluir todos os arquivos da biblioteca.
  - Selecione o **escolher arquivos específicos** botão de opção para incluir um subconjunto dos arquivos da biblioteca. Quando o botão de opção é selecionado, a árvore do seletor de arquivos está habilitada. Marque as caixas à esquerda dos nomes de arquivo para baixar.
- Especifique a pasta de projeto para armazenar os arquivos a **local de destino** caixa de texto. Como uma recomendação, armazene cada biblioteca em uma pasta separada.

Sugeridas **local de destino** pasta baseia-se no local a partir do qual a caixa de diálogo iniciado:

- Se iniciado a partir da raiz do projeto:
  - `wwwroot/lib` será usado se `wwwroot` existe.
  - `lib` será usado se `wwwroot` não existe.
- Se iniciado em uma pasta de projeto, o nome da pasta correspondente será usado.

A sugestão de pasta é sufixada com o nome da biblioteca. A tabela a seguir ilustra as sugestões de pasta durante a instalação do jQuery em um projeto de páginas do Razor.

INICIE O LOCAL	PASTA SUGERIDA
raiz do projeto (se <code>wwwroot</code> existe)	<code>wwwroot/lib/jquery /</code>
raiz do projeto (se <code>wwwroot</code> não existe)	<code>jquery/lib /</code>
<i>Páginas</i> pasta do projeto	<code>Páginas/jquery /</code>

- Clique o **instale** botão para baixar os arquivos, de acordo com a configuração no `libman.json`.
- Examine a **Gerenciador de biblioteca** feed da **saída** janela para obter detalhes de instalação. Por exemplo:

```
Restore operation started...
Restoring libraries for project LibManSample
Restoring library jquery@3.3.1... (LibManSample)
wwwroot/lib/jquery/jquery.min.js written to destination (LibManSample)
wwwroot/lib/jquery/jquery.js written to destination (LibManSample)
wwwroot/lib/jquery/jquery.map written to destination (LibManSample)
Restore operation completed
1 libraries restored in 2.32 seconds
```

## Configurar manualmente as entradas do arquivo de manifesto LibMan

Todas as operações de LibMan no Visual Studio são baseadas no conteúdo do manifesto de LibMan da raiz do projeto (`libman.json`). Você pode editar manualmente `libman.json` para configurar arquivos de biblioteca do projeto. O Visual Studio restaura todos os arquivos de biblioteca uma vez `libman.json` é salvo.

Para abrir `libman.json` para edição, existem as seguintes opções:

- Clique duas vezes o `libman.json` arquivo no **Gerenciador de soluções**.
- Clique com botão direito no projeto no **Gerenciador de soluções** e selecione **gerenciar bibliotecas de cliente**. †
- Selecione **gerenciar bibliotecas do lado do cliente** do Visual Studio **projeto** menu. †

† Se o `libman.json` arquivo ainda não existir na raiz do projeto, ele será criado com o conteúdo do modelo de item

padrão.

Visual Studio oferece um rico JSON, suporte, como colorização de edição, formatação, IntelliSense e validação de esquema. Esquema JSON do manifesto LibMan é encontrada em <http://json.schemastore.org/libman>.

Com o seguinte arquivo de manifesto, LibMan recupera arquivos de acordo com a configuração definida no `libraries` propriedade. Obter uma explicação dos literais de objeto definidos dentro `libraries` segue:

- Um subconjunto de `jQuery` versão 3.3.1 é recuperado do provedor CDNJS. O subconjunto é definido na `files` propriedade—`jQuery`, `obtive`, e `jquery.min.map`. Os arquivos são colocados no projeto do `wwwroot/lib/jquery` pasta.
- Na íntegra `Bootstrap` versão 4.1.3 é recuperado e colocado em um `wwwroot/lib/bootstrap` pasta. O literal de objeto `provider` substituições de propriedades de `defaultProvider` valor da propriedade. LibMan recupera os arquivos de inicialização do provedor de unpkg.
- Um subconjunto de `Lodash` foi aprovada por um corpo regulador dentro da organização. O `lodash.js` e `lodash.min.js` arquivos são recuperados do sistema de arquivos local no `c:\temp\lodash\`. Os arquivos são copiados para o projeto `wwwroot/lib/lodash` pasta.

```
{  
  "version": "1.0",  
  "defaultProvider": "cdnjs",  
  "libraries": [  
    {  
      "library": "jquery@3.3.1",  
      "files": [  
        "jquery.min.js",  
        "jquery.js",  
        "jquery.min.map"  
      ],  
      "destination": "wwwroot/lib/jquery/"  
    },  
    {  
      "provider": "unpkg",  
      "library": "bootstrap@4.1.3",  
      "destination": "wwwroot/lib/bootstrap/"  
    },  
    {  
      "provider": "filesystem",  
      "library": "C:\\temp\\\\lodash\\\\",  
      "files": [  
        "lodash.js",  
        "lodash.min.js"  
      ],  
      "destination": "wwwroot/lib/lodash/"  
    }  
  ]  
}
```

#### NOTE

LibMan só dá suporte a uma versão de cada biblioteca de cada provedor. O `libman.json` arquivo Falha na validação de esquema, se ele contiver duas bibliotecas com o mesmo nome de biblioteca para determinado provedor.

## Restaurar arquivos de biblioteca

Para restaurar os arquivos de biblioteca de dentro do Visual Studio, deve haver um válido `libman.json` arquivo na raiz do projeto. Arquivos restaurados são colocados no projeto no local especificado para cada biblioteca.

Arquivos de biblioteca podem ser restaurados em um projeto do ASP.NET Core de duas maneiras:

1. [Restaurar os arquivos durante a compilação](#)
2. [Restaurar os arquivos manualmente](#)

### Restaurar os arquivos durante a compilação

LibMan pode restaurar os arquivos de biblioteca definidas como parte do processo de compilação. Por padrão, o *restauração no build* comportamento está desabilitado.

Para ativar e testar o comportamento de restauração no build:

- Clique com botão direito *libman.json* na **Gerenciador de soluções** e selecione **habilitar bibliotecas de cliente restaurar no Build** no menu de contexto.
- Clique o **Sim** botão quando for solicitado a instalar um pacote do NuGet. O **Microsoft.Web.LibraryManager.Build** pacote do NuGet é adicionado ao projeto:

```
<PackageReference Include="Microsoft.Web.LibraryManager.Build" Version="1.0.113" />
```

- Compile o projeto para confirmar a restauração de arquivo LibMan ocorre. O **Microsoft.Web.LibraryManager.Build** pacote injeta um destino do MSBuild que executa LibMan durante a operação de compilação do projeto.
- Examine a **compilar** feed da **saída** janela para um log de atividades LibMan:

```
1>----- Build started: Project: LibManSample, Configuration: Debug Any CPU -----
1>
1>Restore operation started...
1>Restoring library jquery@3.3.1...
1>Restoring library bootstrap@4.1.3...
1>
1>2 libraries restored in 10.66 seconds
1>LibManSample -> C:\LibManSample\bin\Debug\netcoreapp2.1\LibManSample.dll
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

Quando o comportamento de restauração no build está habilitado, o *libman.json* menu de contexto exibe um **desabilitar bibliotecas de cliente restaurar no Build** opção. Esta opção remove o

**Microsoft.Web.LibraryManager.Build** referência do arquivo de projeto de pacote. Consequentemente, as bibliotecas do lado do cliente não são restauradas em cada compilação.

Independentemente da configuração de restauração no build, você pode restaurar manualmente a qualquer momento do *libman.json* menu de contexto. Para obter mais informações, consulte [restaurar os arquivos manualmente](#).

### Restaurar os arquivos manualmente

Para restaurar manualmente os arquivos de biblioteca:

- Para todos os projetos na solução:
  - Clique com botão direito no nome da solução no **Gerenciador de soluções**.
  - Selecione o **bibliotecas de cliente restaurar** opção.
- Para um projeto específico:
  - Clique com botão direito do *libman.json* arquivo no **Gerenciador de soluções**.
  - Selecione o **bibliotecas de cliente restaurar** opção.

Enquanto a operação de restauração está em execução:

- O ícone do Centro de Status da tarefa (TSC) na barra de status do Visual Studio será animado e lerá *iniciada a operação de restauração*. Clicando no ícone abre uma dica de ferramenta listando as tarefas em

segundo plano conhecidos.

- As mensagens serão enviadas à barra de status e o **Gerenciador de biblioteca** feed da **Saída** janela. Por exemplo:

```
Restore operation started...
Restoring libraries for project LibManSample
Restoring library jquery@3.3.1... (LibManSample)
wwwroot/lib/jquery/jquery.min.js written to destination (LibManSample)
wwwroot/lib/jquery/jquery.js written to destination (LibManSample)
wwwroot/lib/jquery/jquery.map written to destination (LibManSample)
Restore operation completed
1 libraries restored in 2.32 seconds
```

## Excluir arquivos de biblioteca

Para executar o *limpa* operação, que exclui os arquivos de biblioteca restaurados anteriormente no Visual Studio:

- Clique com botão direito do *libman.json* arquivo no **Gerenciador de soluções**.
- Selecione o **bibliotecas de cliente limpa** opção.

Para evitar a remoção não intencional de arquivos não seja de biblioteca, a operação de limpeza não exclui diretórios inteiros. Ela remove somente os arquivos que foram incluídos na restauração anterior.

Enquanto a operação de limpeza é executado:

- O ícone TSC na barra de status do Visual Studio será animado e lerá *operação de bibliotecas de cliente iniciada*. Clicando no ícone abre uma dica de ferramenta listando as tarefas em segundo plano conhecidos.
- As mensagens são enviadas para a barra de status e o **Gerenciador de biblioteca** feed da **Saída** janela. Por exemplo:

```
Clean libraries operation started...
Clean libraries operation completed
2 libraries were successfully deleted in 1.91 secs
```

A operação de limpeza exclui somente os arquivos do projeto. Arquivos de biblioteca permanecem no cache para uma recuperação mais rápida em operações de restauração futuras. Para gerenciar arquivos de biblioteca armazenados no cache do computador local, use o [LibMan CLI](#).

## Desinstalar os arquivos de biblioteca

Para desinstalar os arquivos de biblioteca:

- Abra *libman.json*.
- Posicione o cursor dentro correspondente `libraries` literal de objeto.
- Clique no ícone de lâmpada que aparece na margem esquerda e selecione **Uninstall <nome\_da\_biblioteca> @<library\_version>**:



Como alternativa, você pode editar e salvar o manifesto LibMan manualmente (*libman.json*). O [operação de restauração](#) é executado quando o arquivo é salvo. Arquivos de biblioteca que não estão definidos na *libman.json* são removidas do projeto.

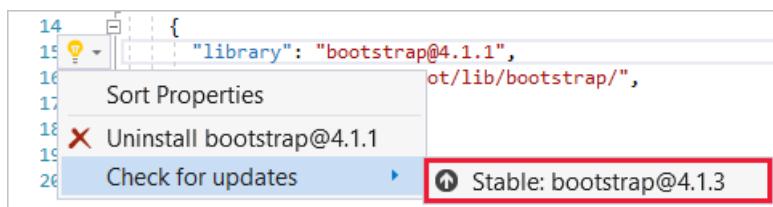
## Atualizar a versão da biblioteca

Para verificar se há uma versão atualizada de biblioteca:

- Abra *libman.json*.
- Posicione o cursor dentro correspondente `libraries` literal de objeto.
- Clique no ícone de lâmpada que aparece na margem esquerda. Passe o mouse sobre **verificar se há atualizações**.

LibMan verifica se há uma versão mais recente do que a versão instalada da biblioteca. Podem ocorrer os seguintes resultados:

- Um **nenhuma atualização encontrada** mensagem será exibida se a versão mais recente já está instalada.
- A versão estável mais recente é exibida se não estiver instalada.



- Se houver uma versão de pré-lançamento mais recente do que a versão instalada, a versão de pré-lançamento é exibida.

Para fazer o downgrade para uma versão mais antiga da biblioteca, edite manualmente o *libman.json* arquivo. Quando o arquivo é salvo, o LibMan [operação de restauração](#):

- Remove arquivos redundantes da versão anterior.
- Adiciona arquivos novos e atualizados da nova versão.

## Recursos adicionais

- [Use a interface de linha de comando LibMan \(CLI\) com o ASP.NET Core](#)
- [Repositório do GitHub do LibMan](#)

# Gerenciar pacotes do lado do cliente com Bower no ASP.NET Core

13/11/2018 • 10 minutes to read • [Edit Online](#)

Por [Rick Anderson](#), [Noel arroz](#), e [Scott Addie](#)

## IMPORTANT

Enquanto o Bower é mantido, seus mantenedores recomendam usando uma solução diferente. [Gerenciador de biblioteca \(LibMan de forma abreviada\)](#) é a ferramenta de aquisição de biblioteca do lado do cliente novo do Visual Studio (Visual Studio 15,8 ou posterior). Para obter mais informações, consulte [Aquisição de biblioteca do lado do cliente no ASP.NET Core com LibMan](#). Bower tem suporte no Visual Studio versão 15,5.

Yarn com Webpack é uma alternativa popular para a qual [instruções de migração](#) estão disponíveis.

[Bower](#) chama a próprio "Um Gerenciador de pacotes para a web". Dentro do ecossistema do .NET, ele preenche essa lacuna deixado pela incapacidade do NuGet para entregar os arquivos de conteúdo estático. Para projetos do ASP.NET Core, esses arquivos estáticos são inerentes às bibliotecas do lado do cliente, como [jQuery](#) e [Bootstrap](#). Para bibliotecas do .NET, você usar [NuGet](#) Gerenciador de pacotes.

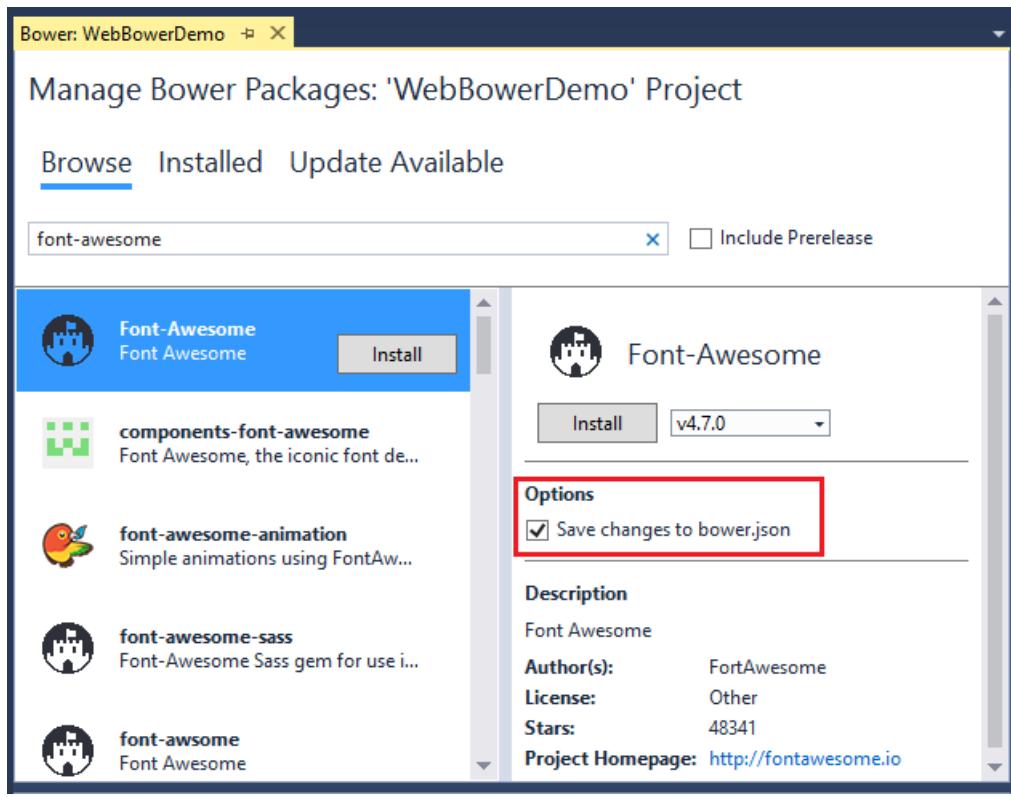
Processo de compilação de novos projetos criados com os modelos de projeto do ASP.NET Core configurar lado do cliente. [jQuery](#) e [Bootstrap](#) estiverem instalados, e Bower é suportado.

Pacotes do lado do cliente são listados na `bower.json` arquivo. Os modelos de projeto do ASP.NET Core configura `bower.json` com o jQuery, validação do jQuery e Bootstrap.

Neste tutorial, vamos adicionar suporte para [Font Awesome](#). Pacotes do bower podem ser instalados com o **gerenciar pacotes do Bower** interface do usuário ou manualmente na `bower.json` arquivo.

## Instalação por meio de pacotes do Bower gerenciar da interface do usuário

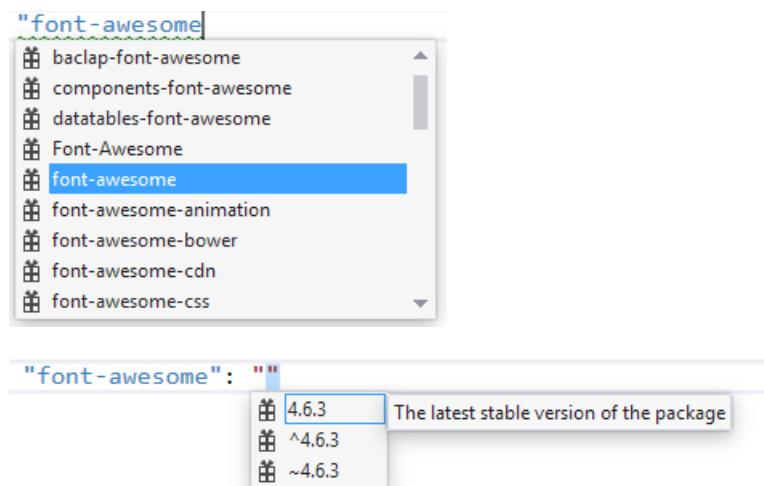
- Criar um novo aplicativo Web do ASP.NET Core com o **aplicativo Web do ASP.NET Core (.NET Core)** modelo. Selecione **aplicativo Web e nenhuma autenticação**.
- Clique com botão direito no projeto no Gerenciador de soluções e selecione **gerenciar pacotes do Bower** (como alternativa, no menu principal, **Project > gerenciar pacotes Bower**).
- No **Bower: <nome do projeto>** janela, clique na guia "Procurar" e, em seguida, filtre a lista de pacotes inserindo `font-awesome` na caixa de pesquisa:



- Confirme se o "Salvar alterações bower JSON" caixa de seleção está marcada. Selecione uma versão na lista suspensa e clique no **instalar** botão. O **saída** janela mostra os detalhes da instalação.

### Instalação manual no bower. JSON

Abra o *bower.json* arquivo e adicione "font-incrível" para as dependências. O IntelliSense mostra os pacotes disponíveis. Quando um pacote é selecionado, as versões disponíveis são exibidas. As imagens abaixo são mais antigas e não corresponder ao que você vê.



Usos para bower [controle de versão semântico](#) para organizar as dependências. Controle de versão semântico, também conhecido como SemVer, identifica os pacotes com o esquema de numeração <principal>.<secundária>. <patch>. IntelliSense simplifica o controle de versão semântico, mostrando apenas algumas opções comuns. O item superior na lista do IntelliSense (4.6.3 no exemplo acima) é considerado a versão estável mais recente do pacote. O símbolo de acento circunflexo (^) corresponde à versão principal mais recente e o til (~) corresponde a versão secundária mais recente.

Salvar a *bower.json* arquivo. Visual Studio inspeciona as *bower.json* arquivo para que as alterações. Ao salvar, o *bower install* comando é executado. Consulte a janela de saída **npm/Bower** modo de exibição para o comando executado.

Abra o *.bowerrc* do arquivo sob *bower.json*. O *directory* estiver definida como *wwwroot/lib* que indica o local

do Bower instalará os ativos do pacote.

```
{  
  "directory": "wwwroot/lib"  
}
```

Você pode usar a caixa de pesquisa no Gerenciador de soluções para localizar e exibir o pacote font awesome.

Abra o `Views\Shared_layout.cshtml` arquivo e adicione o arquivo CSS font awesome no ambiente auxiliar de marca para `Development`. No Gerenciador de soluções, arraste e solte `fonte awesome.css` dentro de `<environment names="Development">` elemento.

```
<environment names="Development">  
  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />  
  <link rel="stylesheet" href="~/css/site.css" />  
  <link href="~/lib/font-awesome/css/font-awesome.css" rel="stylesheet" />  
</environment>
```

Um aplicativo de produção, você adicionaria `fonte awesome.min.css` para o auxiliar de marca de ambiente para `Staging,Production`.

Substitua o conteúdo do `Views\Home\About.cshtml` arquivo Razor com a seguinte marcação:

```
@{  
  ViewData["Title"] = "About";  
}  
  
<div class="list-group">  
  <a class="list-group-item" href="#"></i>&nbsp; Home</a>  
  <a class="list-group-item" href="#"></i>&nbsp; Library</a>  
  <a class="list-group-item" href="#"></i>&nbsp;  
Applications</a>  
  <a class="list-group-item" href="#"></i>&nbsp; Settings</a>  
</div>
```

Execute o aplicativo e navegue até a exibição About sobre para verificar se o pacote font awesome funciona.

## Explorando o processo de compilação do lado do cliente

A maioria dos modelos de projeto do ASP.NET Core já estão configurados para usar o Bower. Este passo a passo de próximo começa com um projeto vazio do ASP.NET Core e adiciona cada parte manualmente, portanto, você pode ter uma ideia de como o Bower é usado em um projeto. Você pode ver o que acontece com a estrutura do projeto e o tempo de execução de saída que cada alteração de configuração é feita.

As etapas gerais para usar o processo de compilação do lado do cliente com o Bower são:

- Defina pacotes usados em seu projeto.
- Pacotes de referência de suas páginas da web.

### Definir pacotes

Depois que você listar pacotes na `bower.json` arquivo, o Visual Studio irá baixá-los. O exemplo a seguir usa o Bower para carregar o jQuery e Bootstrap para o `wwwroot` pasta.

- Criar um novo aplicativo Web do ASP.NET Core com o **aplicativo Web do ASP.NET Core (.NET Core)** modelo. Selecione o **vazio** modelo de projeto e clique em **Okey**.
- No Gerenciador de soluções, clique com botão direito no projeto > **Adicionar Novo Item** e selecione

**arquivo de configuração Bower.** Observação: Um `.bowerrc` arquivo também é adicionado.

- Abra `bower.json` e adicionar o `jquery` e `bootstrap` para o `dependencies` seção. Resultante `bower.json` arquivo se parecerá com o exemplo a seguir. As versões serão alteradas ao longo do tempo e podem não corresponder a imagem a seguir.

```
{  
  "name": "asp.net",  
  "private": true,  
  "dependencies": {  
    "jquery": "3.1.1",  
    "bootstrap": "3.3.7"  
  }  
}
```

- Salvar a `bower.json` arquivo.

Verifique se o projeto inclui o `bootstrap` e `jQuery` diretórios `wwwroot/lib`. Bower usa o `.bowerrc` arquivo para instalar os ativos na `wwwroot/lib`.

Observação: A interface do usuário "Gerenciar pacotes do Bower" fornece uma alternativa à edição do arquivo manual.

## Habilitar arquivos estáticos

- Adicionar o `Microsoft.AspNetCore.StaticFiles` pacote NuGet ao projeto.
- Habilitar arquivos estáticos a serem atendidos com o [middleware de arquivo estático](#). Adicione uma chamada para `UseStaticFiles` para o `Configure` método `Startup`.

```
using Microsoft.AspNetCore.Builder;  
using Microsoft.AspNetCore.Hosting;  
using Microsoft.AspNetCore.Http;  
  
public class Startup  
{  
    public void Configure(IApplicationBuilder app)  
    {  
        app.UseStaticFiles();  
  
        app.Run(async (context) =>  
        {  
            await context.Response.WriteAsync("Hello World!");  
        });  
    }  
}
```

## Pacotes de referência

Nesta seção, você criará uma página HTML para verificar se ele pode acessar os pacotes implantados.

- Adicionar uma nova página HTML chamada `index.html` para o `wwwroot` pasta. Observação: Você deve adicionar o arquivo HTML para o `wwwroot` pasta. Por padrão, o conteúdo estático não pode ser servido fora `wwwroot`. Ver [arquivos estáticos](#) para obter mais informações.

Substitua o conteúdo do `index.html` com a seguinte marcação:

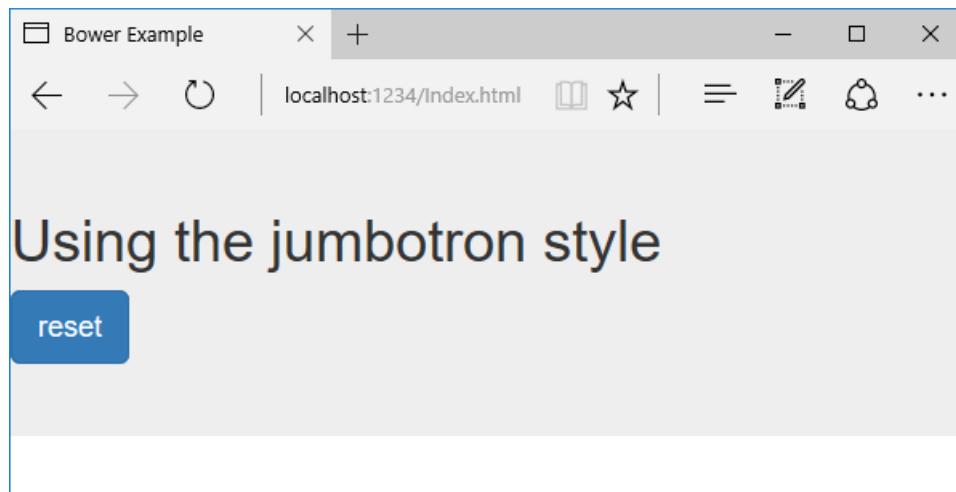
```

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Bower Example</title>
    <link href="lib/bootstrap/dist/css/bootstrap.css" rel="stylesheet" />
</head>
<body>
    <div class="jumbotron">
        <h1>Using the jumbotron style</h1>
        <p>
            <a class="btn btn-primary btn-lg" role="button">Stateful button</a>
        </p>
    </div>
    <script src="lib/jquery/dist/jquery.js"></script>
    <script src="lib/bootstrap/dist/js/bootstrap.js"></script>
    <script>
        $(".btn").click(function () {
            $(this).text('loading')
                .delay(1000)
                .queue(function () {
                    $(this).text('reset');
                    $(this).dequeue();
                });
        });
    </script>
</body>

</html>

```

- Execute o aplicativo e navegue até `http://localhost:<port>/Index.html`. Como alternativa, com *index.HTML* aberto, pressione `Ctrl+Shift+W`. Verifique se que o estilo de jumbotron é aplicado, o código jQuery responde quando o botão é clicado e que o Bootstrap botão muda de estado.



# Less, Sass e fonte Awesome no ASP.NET Core

22/06/2018 • 22 minutes to read • [Edit Online](#)

Por [Steve Smith](#)

Os usuários de aplicativos da web têm expectativas cada vez mais altas quando se trata de estilo e a experiência geral. Aplicativos web modernos frequentemente aproveitam avançadas ferramentas e estruturas para definir e gerenciar sua aparência de uma maneira consistente. Estruturas como [inicialização](#) pode ir um longo caminho para definir um conjunto comum de opções de layout para sites da web e estilos. No entanto, a maioria dos sites não trivial também se beneficiar de ser capaz de definir e manter estilos e arquivos de folhas de estilo em cascata com eficiência, bem como acesso fácil a imagem não ícones que ajudam a tornar a interface do site mais intuitiva. É onde linguagens e ferramentas que dão suporte a [menos](#) e [Sass](#), e bibliotecas, como [fonte Awesome](#), entrar.

## Idiomas de pré-processador de CSS

Idiomas que são compilados em outros idiomas, para melhorar a experiência de trabalhar com o idioma base são chamados de pré-processador. Há dois pré-processadores populares de CSS: menor e Sass. Esses pré-processadores adicionar recursos a CSS, como suporte para variáveis e regras aninhadas que melhorar a facilidade de manutenção de folhas de estilo grandes e complexas. CSS como uma linguagem é muito básica, sem suporte a até mesmo algo simples, como variáveis e isso tende a fazer arquivos CSS repetitivas e inchada. Adicionando recursos de idioma real de programação via pré-processadores pode ajudar a reduzir a duplicação e fornecer melhor organização de regras de estilo. Visual Studio fornece suporte interno para ambos os menor e Sass, bem como as extensões que podem melhorar ainda mais a experiência de desenvolvimento ao trabalhar com esses idiomas.

Como um exemplo rápido de como pré-processadores podem melhorar a leitura e a manutenção de informações de estilo, considere este CSS:

```
.header {  
    color: black;  
    font-weight: bold;  
    font-size: 18px;  
    font-family: Helvetica, Arial, sans-serif;  
}  
  
.small-header {  
    color: black;  
    font-weight: bold;  
    font-size: 14px;  
    font-family: Helvetica, Arial, sans-serif;  
}
```

Usando Less, isso pode ser reescrito para eliminar a duplicação, usando um *mixin* (chamada assim porque ele permite que você "misture" propriedades de uma classe ou conjunto de regras em outra):

```

.header {
    color: black;
    font-weight: bold;
    font-size: 18px;
    font-family: Helvetica, Arial, sans-serif;
}

.small-header {
    .header;
    font-size: 14px;
}

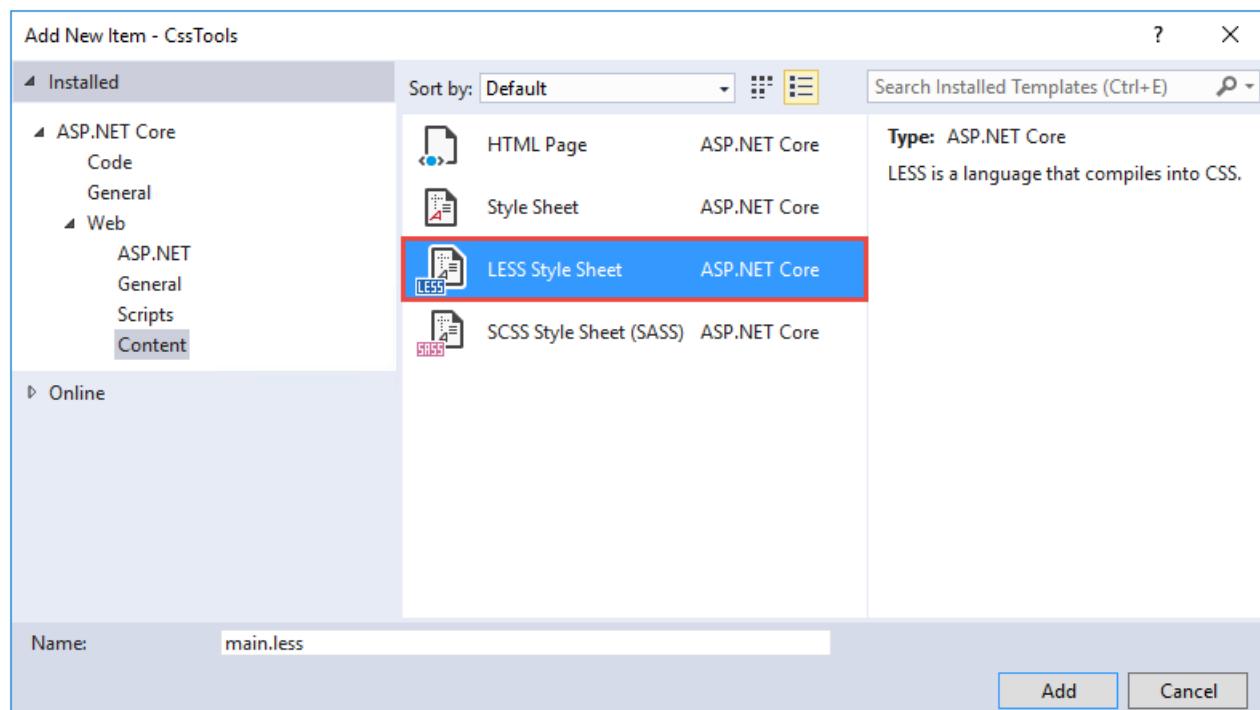
```

## Less

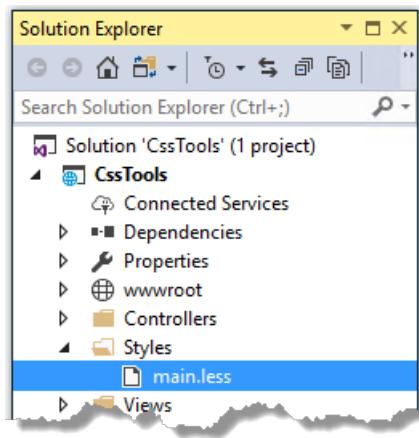
O pré-processador CSS less é executado usando o Node.js. Para instalar less, use o Gerenciador de pacotes de nó (npm) em um prompt de comando (-g significa "global"):

```
npm install -g less
```

Se você estiver usando o Visual Studio, você pode começar com less adicionando um ou mais arquivos less ao seu projeto e, em seguida, configurando Gulp (ou Grunt) para processá-los em tempo de compilação. Adicione uma pasta *estilos* ao seu projeto e, em seguida, adicione um novo arquivo less chamado *main.less* nesta pasta.



Depois de adicionado, a estrutura de pastas deve ser algo assim:



Agora você pode adicionar alguns estilos básicos para o arquivo, que será compilado em CSS e implantado na pasta wwwroot por vez.

Modificar *main.less* para incluir o conteúdo a seguir, que cria uma paleta de cores simples de uma única cor base.

```
@base: #663333;
@background: spin(@base, 180);
@lighter: lighten(spin(@base, 5), 10%);
@lighter2: lighten(spin(@base, 10), 20%);
@darker: darken(spin(@base, -5), 10%);
@darker2: darken(spin(@base, -10), 20%);

body {
    background-color:@background;
}
.baseColor {color:@base}
.bgLight {color:@lighter}
.bgLight2 {color:@lighter2}
.bgDark {color:@darker}
.bgDark2 {color:@darker2}
```

`@base` e o outro `@`-prefixed itens são variáveis. Cada um deles representa uma cor. Exceto para `@base`, eles são definidos usando funções de cor: mais claro, mais escuro e rotação. Mais claro e escureça fazer muito bem o que você esperaria; rotação ajusta o matiz da cor por um número de graus (ao redor do círculo de cores). O processador de menos é inteligente ignore variáveis que não são usados, por isso, para demonstrar como funcionam essas variáveis, precisamos usá-los em algum lugar. As classes `.baseColor`, etc. demonstrará os valores calculados de cada uma das variáveis no arquivo CSS que é produzida.

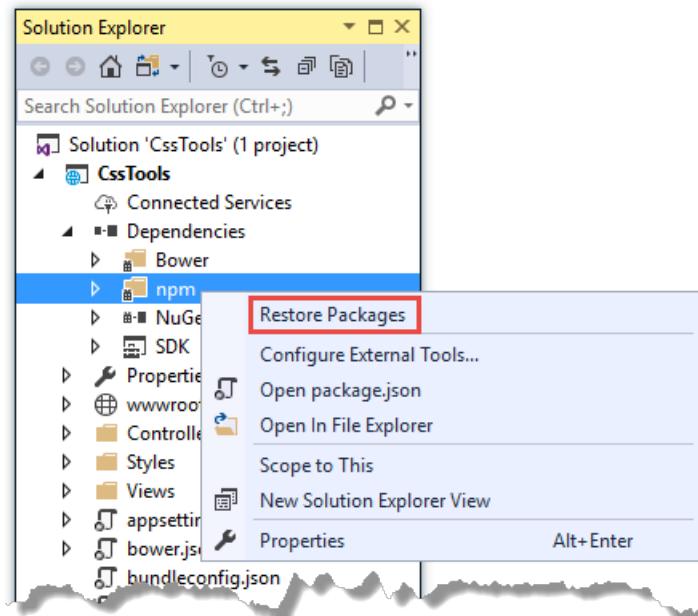
## Introdução

Criar um **arquivo de configuração npm** (*Package.json*) na pasta do projeto e editá-lo para fazer referência a `gulp` e `gulp-less`:

```
{
  "version": "1.0.0",
  "name": "asp.net",
  "private": true,
  "devDependencies": {
    "gulp": "3.9.1",
    "gulp-less": "3.3.0"
  }
}
```

Instalar as dependências em um prompt de comando na pasta do projeto ou no Visual Studio **Solution Explorer** (**dependências > npm > restaurar os pacotes**).

```
npm install
```



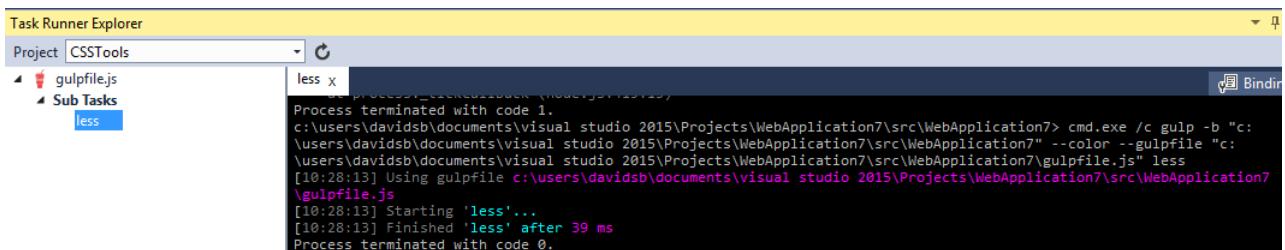
Na pasta do projeto, criar um **Gulp arquivo de configuração** (`gulpfile.js`) para definir o processo automatizado. Adicione uma variável na parte superior do arquivo para representar less e uma tarefa para execução less:

```
var gulp = require("gulp"),
    fs = require("fs"),
    less = require("gulp-less");

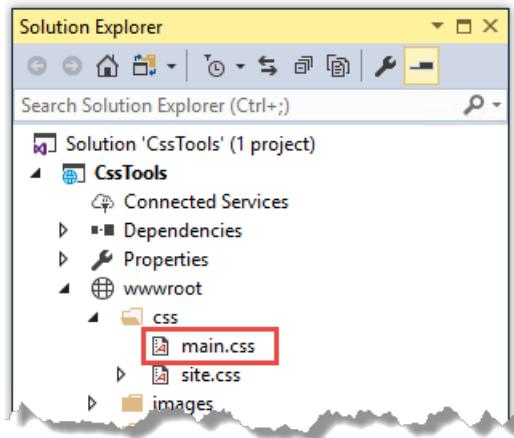
gulp.task("less", function () {
    return gulp.src('Styles/main.less')
        .pipe(less())
        .pipe(gulp.dest('wwwroot/css'));
});
```

Abra o **Explorador do Executador de tarefas** (exibição > outras janelas > **Explorador do Executador de tarefas**). Entre as tarefas, você verá uma nova tarefa denominada `less`. Você talvez precise atualizar a janela.

Execute o `less` tarefa e você verá uma saída semelhante ao que é mostrado aqui:



O `wwwroot/css` pasta agora contém um novo arquivo, `Main`:



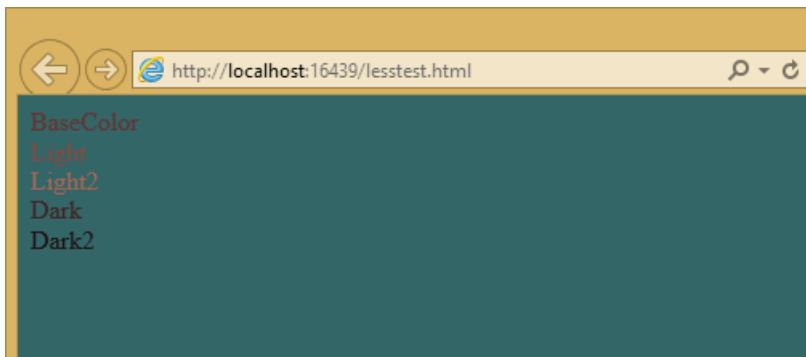
Abra *Main* e você verá algo parecido com o seguinte:

```
body {  
    background-color: #336666;  
}  
.baseColor {  
    color: #663333;  
}  
.bgLight {  
    color: #884a44;  
}  
.bgLight2 {  
    color: #aa6355;  
}  
.bgDark {  
    color: #442225;  
}  
.bgDark2 {  
    color: #221114;  
}
```

Adicionar uma página HTML simples para o *wwwroot* pasta e referência *Main* para ver a paleta de cores em ação.

```
<!DOCTYPE html>  
<html>  
<head>  
    <meta charset="utf-8" />  
    <link href="css/main.css" rel="stylesheet" />  
    <title></title>  
</head>  
<body>  
    <div>  
        <div class="baseColor">BaseColor</div>  
        <div class="bgLight">Light</div>  
        <div class="bgLight2">Light2</div>  
        <div class="bgDark">Dark</div>  
        <div class="bgDark2">Dark2</div>  
    </div>  
</body>  
</html>
```

Você pode ver que o grau de 180 girar na `@base` usada para produzir `@background` resultou no disco de cores opostas cor de `@base`:



Less também oferece suporte para regras aninhadas, bem como as consultas de mídia aninhada. Por exemplo, definindo hierarquias aninhadas como menus podem resultar em regras de CSS detalhadas como estes:

```
nav {  
    height: 40px;  
    width: 100%;  
}  
nav li {  
    height: 38px;  
    width: 100px;  
}  
nav li a:link {  
    color: #000;  
    text-decoration: none;  
}  
nav li a:visited {  
    text-decoration: none;  
    color: #CC3333;  
}  
nav li a:hover {  
    text-decoration: underline;  
    font-weight: bold;  
}  
nav li a:active {  
    text-decoration: underline;  
}
```

O ideal é todas as regras de estilo relacionados serão colocadas juntos dentro do arquivo CSS, mas na prática, não há nada impor essa regra exceto convenção e talvez os comentários do bloco.

Definir essas mesmas regras usando less tem esta aparência:

```
nav {  
    height: 40px;  
    width: 100%;  
    li {  
        height: 38px;  
        width: 100px;  
        a {  
            color: #000;  
            &:link { text-decoration:none}  
            &:visited { color: #CC3333; text-decoration:none}  
            &:hover { text-decoration:underline; font-weight:bold}  
            &:active {text-decoration:underline}  
        }  
    }  
}
```

Observe que, nesse caso, todos os elementos subordinados do `nav` estão contidos dentro de seu escopo. Não há nenhuma repetição dos elementos pai (`nav`, `li`, `a`), e a contagem de linha de total caiu também (embora algumas das é um resultado de colocar valores nas linhas da mesmas no segundo exemplo). Ele pode ser muito

útil, organizacional, para ver todas as regras para um determinado elemento de interface do usuário em um escopo explicitamente associado, nesse caso definido do restante do arquivo de chaves.

O `&` sintaxe é um recurso seletor less, com `&` que representa o pai de seletor atual. Portanto, dentro do `{...}` bloco, `&` representa um `a` marca e, portanto, `&:link` é equivalente a `a:link`.

Consultas de mídia, extremamente úteis na criação de designs de resposta também podem contribuir muito para repetição e a complexidade em CSS. Less permite que as consultas de mídia a ser aninhada dentro de classes, para que a definição de classe inteira não precisa ser repetido em diferentes nível superior `@media` elementos. Por exemplo, aqui está o CSS para um menu de resposta:

```
.navigation {  
    margin-top: 30%;  
    width: 100%;  
}  
@media screen and (min-width: 40em) {  
    .navigation {  
        margin: 0;  
    }  
}  
@media screen and (min-width: 62em) {  
    .navigation {  
        width: 960px;  
        margin: 0;  
    }  
}
```

Isso pode ser melhor definido em less como:

```
.navigation {  
    margin-top: 30%;  
    width: 100%;  
    @media screen and (min-width: 40em) {  
        margin: 0;  
    }  
    @media screen and (min-width: 62em) {  
        width: 960px;  
        margin: 0;  
    }  
}
```

Outro recurso de less que já vimos é seu suporte para operações matemáticas, permitindo que os atributos de estilo a ser construído de variáveis predefinidas. Isso facilita a atualização estilos relacionados muito mais fácil, já que a variável de base pode ser modificada e todos os valores dependentes alterar automaticamente.

Arquivos CSS, especialmente para grandes sites (e especialmente se as consultas de mídia estão sendo usadas), tendem a ter muito grandes ao longo do tempo, tornando a trabalhar com elas complicada. Arquivos less podem ser definidos separadamente, obtidas usando `@import` diretivas. Less também pode ser usado para importar arquivos CSS individuais, se desejado.

*Mixins* pode aceitar parâmetros e less oferece suporte à lógica condicional na forma de protege mesclado, que fornecem uma maneira declarativa para definir quando determinados mixins entra em vigor. Um uso comum para protege mesclado ajustar as cores com base em como a luz ou escuro a cor da fonte. Dado um mesclado que aceita um parâmetro para a cor, um protetor mesclado pode ser usado para modificar o mesclado com base na cor:

```

.box (@color) when (lightness(@color) >= 50%) {
    background-color: #000;
}
.box (@color) when (lightness(@color) < 50%) {
    background-color: #FFF;
}
.box (@color) {
    color: @color;
}

.feature {
    .box (@base);
}

```

Considerando nossa atual `@base` valor `#663333`, esse script less produzirá o seguinte CSS:

```

.feature {
    background-color: #FFF;
    color: #663333;
}

```

Less fornece uma série de recursos adicionais, mas isso deve dar uma ideia da energia desse idioma de pré-processamento.

## Sass

Sass é semelhante ao menos, fornecendo suporte para muitos dos mesmos recursos, mas com sintaxe ligeiramente diferente. Ele é criado usando o Ruby, em vez de JavaScript, e portanto tem requisitos de instalação diferentes. O idioma Sass original não usa chaves ou ponto e vírgula, mas em vez disso, definida escopo usando o recuo e espaços em branco. Na versão 3 dos Sass, uma nova sintaxe foi introduzida, **SCSS** ("Sassy CSS"). SCSS é semelhante ao CSS que ignora os níveis de recuo e espaços em branco e, em vez disso, usa o ponto e vírgula e chaves.

Para instalar Sass, normalmente você deve primeiro instalar Ruby (pré-instalado macOS) e, em seguida, execute:

```
gem install sass
```

No entanto, se você estiver executando o Visual Studio, você pode começar com Sass em grande parte da mesma maneira como você faria com less. Abra *Package.json* e adicionar o pacote "gulp sass" `devDependencies`:

```

"devDependencies": {
    "gulp": "3.9.1",
    "gulp-less": "3.3.0",
    "gulp-sass": "3.1.0"
}

```

Em seguida, modifique *gulpfile.js* para adicionar uma variável de sass e uma tarefa para compilar os arquivos Sass e colocar os resultados na pasta wwwroot:

```

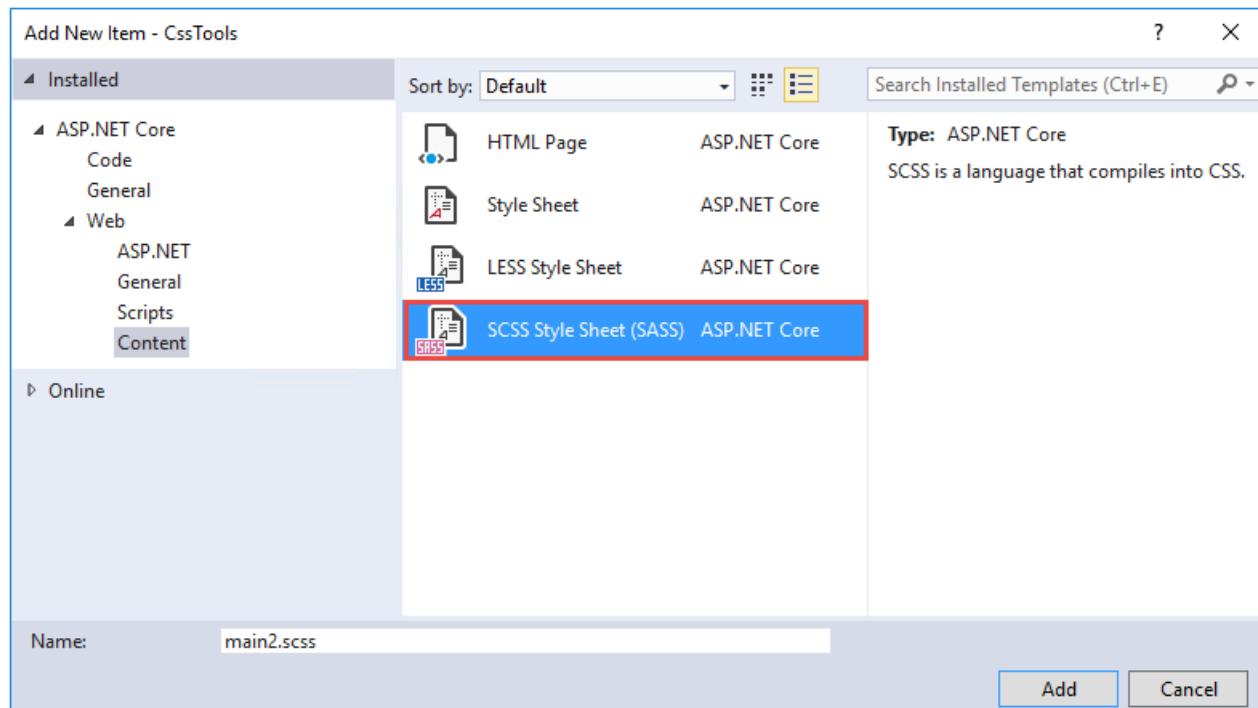
var gulp = require("gulp"),
  fs = require("fs"),
  less = require("gulp-less"),
  sass = require("gulp-sass");

// other content removed

gulp.task("sass", function () {
  return gulp.src('Styles/main2.scss')
    .pipe(sass())
    .pipe(gulp.dest('wwwroot/css'));
});

```

Agora você pode adicionar o arquivo Sass *main2.scss* para o *estilos* pasta na raiz do projeto:



Abra *main2.scss* e adicione o seguinte:

```

$base: #CC0000;
body {
  background-color: $base;
}

```

Salve todos os arquivos. Agora quando você atualiza **Explorador do Executador de tarefas**, você verá um `sass` tarefa. Executá-lo e examinar o `/wwwroot/css` pasta. Agora há uma *main2.css* arquivo com esse conteúdo:

```

body {
  background-color: #CC0000;
}

```

Sass oferece suporte a aninhamento praticamente o mesmo que less não, fornecer benefícios semelhantes. Os arquivos podem ser divididos por função e incluídos usando a `@import` diretiva:

```

@import 'anotherfile';

```

Sass oferece suporte a mixins, usando o `@mixin` palavra-chave para defini-los e `@include` para incluí-los, como

neste exemplo de [sass lang.com](http://sass-lang.com):

```
@mixin border-radius($radius) {  
  -webkit-border-radius: $radius;  
  -moz-border-radius: $radius;  
  -ms-border-radius: $radius;  
  border-radius: $radius;  
}  
  
.box { @include border-radius(10px); }
```

Além mixins, Sass também oferece suporte ao conceito de herança, permitindo que uma classe estender o outro. Ele é conceitualmente semelhante a um mesclado, mas resulta em menos código CSS. Ela é realizada usando o `@extend` palavra-chave. Para testar mixins, adicione o seguinte ao seu *main2.scss* arquivo:

```
@mixin alert {  
  border: 1px solid black;  
  padding: 5px;  
  color: #333333;  
}  
  
.success {  
  @include alert;  
  border-color: green;  
}  
  
.error {  
  @include alert;  
  color: red;  
  border-color: red;  
  font-weight:bold;  
}
```

Examine a saída em *main2.css* depois de executar o `sass` tarefa no **Explorador do Executador de tarefas**:

```
.success {  
  border: 1px solid black;  
  padding: 5px;  
  color: #333333;  
  border-color: green;  
}  
  
.error {  
  border: 1px solid black;  
  padding: 5px;  
  color: #333333;  
  color: red;  
  border-color: red;  
  font-weight: bold;  
}
```

Observe que todas as propriedades comuns do alerta mesclado são repetidas em cada classe. O mesclado foi um bom trabalho de ajudar a eliminar a duplicação no tempo de desenvolvimento, mas ela ainda está criando um CSS com muita duplicação, resultando em maior do que arquivos CSS necessários - um possível problema de desempenho.

Agora, substitua o alerta mesclado com um `.alert` classe e altere `@include` para `@extend` (Lembre-se estender `.alert`, não `alert`):

```
.alert {  
    border: 1px solid black;  
    padding: 5px;  
    color: #333333;  
}  
  
.success {  
    @extend .alert;  
    border-color: green;  
}  
  
.error {  
    @extend .alert;  
    color: red;  
    border-color: red;  
    font-weight:bold;  
}
```

Execute Sass mais uma vez e examine o CSS resultante:

```
.alert, .success, .error {  
    border: 1px solid black;  
    padding: 5px;  
    color: #333333;  
}  
  
.success {  
    border-color: green;  
}  
  
.error {  
    color: red;  
    border-color: red;  
    font-weight: bold;  
}
```

Agora as propriedades são definidas apenas como quantas vezes forem necessárias, e melhor CSS é gerado.

Sass também inclui funções e operações de lógica condicional, semelhantes ao less. Na verdade, os recursos de dois idiomas são muito semelhantes.

## Less ou Sass?

Ainda não há consenso se geralmente é melhor usar less ou Sass (ou até mesmo se preferir o Sass original ou a sintaxe SCSS mais recente em Sass). Provavelmente, a decisão mais importante é **usar uma dessas ferramentas**, em vez de apenas codificação manual em seus arquivos CSS. Depois que você fez essa decisão, ambos Less e Sass são boas opções.

## Font Awesome

Além de pré-processadores CSS, outro excelente recurso para aplicativos web modernos de estilo é incrível de fonte. Lista de fonte é um kit de ferramentas fornece mais de 500 ícones de SVG que podem ser usados livremente em seus aplicativos web. Ela foi originalmente projetada para trabalhar com inicialização, mas ele não tem nenhuma dependência no framework ou em qualquer biblioteca de JavaScript.

A maneira mais fácil começar com o incríveis fonte é adicionar uma referência a ele, usando seu local de rede (CDN) do fornecimento de conteúdo público:

```
<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/font-awesome/4.3.0/css/font-awesome.min.css">
```

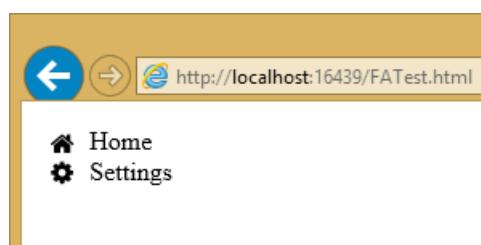
Você também pode adicioná-lo ao seu projeto do Visual Studio adicionando-as "dependências" em *bower.json*:

```
{  
  "name": "ASP.NET",  
  "private": true,  
  "dependencies": {  
    "bootstrap": "3.0.0",  
    "jquery": "1.10.2",  
    "jquery-validation": "1.11.1",  
    "jquery-validation-unobtrusive": "3.2.2",  
    "hammer.js": "2.0.4",  
    "bootstrap-touch-carousel": "0.8.0",  
    "Font-Awesome": "4.3.0"  
  }  
}
```

Depois que você tem uma referência para o incrível fonte em uma página, você pode adicionar ícones para o seu aplicativo aplicando fonte Awesome classes, normalmente é prefixados com "fa-", para os elementos embutidos HTML (como `<span>` ou `<i>`). Por exemplo, você pode adicionar ícones de listas simples e menus usando código como este:

```
<!DOCTYPE html>  
<html>  
<head>  
  <meta charset="utf-8" />  
  <title></title>  
  <link href="lib/font-awesome/css/font-awesome.css" rel="stylesheet" />  
</head>  
<body>  
  <ul class="fa-ul">  
    <li><i class="fa fa-li fa-home"></i> Home</li>  
    <li><i class="fa fa-li fa-cog"></i> Settings</li>  
  </ul>  
</body>  
</html>
```

Isso produz o seguinte no navegador - Observe o ícone ao lado de cada item:



Você pode exibir uma lista completa de ícones disponíveis:

<http://fontawesome.io/icons/>

## Resumo

Aplicativos web modernos exigem designs cada vez mais responsivos e fluidos que são normais, intuitivos e fáceis de usar em uma variedade de dispositivos. Gerenciar a complexidade das folhas de estilo CSS necessárias para atingir essas metas é melhor usando um tipo de pré-processador less ou Sass. Além disso, kits de ferramentas como a Awesome fonte rapidamente fornecem ícones conhecidos a menus de navegação textual e experiência de botões, melhorando o usuário do seu aplicativo.

# Agrupar e minificar ativos estáticos no ASP.NET Core

22/11/2018 • 18 minutes to read • [Edit Online](#)

Por [Scott Addie](#) e [Alcatrão de David](#)

Este artigo explica os benefícios da aplicação de agrupamento e minificação, incluindo como esses recursos podem ser usados com aplicativos web ASP.NET Core.

## O que é o agrupamento e minificação

Agrupamento e minificação são duas otimizações de desempenho distintos, que você pode aplicar em um aplicativo web. Usados juntos, agrupamento e minificação melhoram o desempenho reduzindo o número de solicitações do servidor e reduzindo o tamanho dos ativos mais solicitados estáticos.

Agrupamento e minificação basicamente melhoram o tempo de carregamento de solicitação de página primeiro. Depois que uma página da web foi solicitado, o navegador armazena em cache os recursos estáticos (JavaScript, CSS e imagens). Consequentemente, agrupamento e minificação não melhoram o desempenho ao solicitar a mesma página ou páginas, no mesmo site que está solicitando os mesmos recursos. Se a expira cabeçalho não está definido corretamente nos ativos e se não for usado o agrupamento e minificação, heurística de atualização do navegador marca os ativos obsoletos depois de alguns dias. Além disso, o navegador requer uma solicitação de validação para cada ativo. Nesse caso, o agrupamento e minificação fornecem uma melhoria de desempenho, mesmo após a primeira solicitação de página.

### Agrupamento

O agrupamento combina vários arquivos em um único arquivo. Agrupamento reduz o número de solicitações de servidor que são necessários para renderizar um ativo da web, como uma página da web. Você pode criar qualquer número de pacotes individuais especificamente para o CSS, JavaScript, etc. Menos arquivos significa menos solicitações HTTP do navegador para o servidor ou do serviço de fornecimento de seu aplicativo. Isso resulta em melhor desempenho de carregamento de página primeiro.

### Minimização

Minificação remove caracteres desnecessários de código sem alterar a funcionalidade. O resultado é uma redução de tamanho significativo nos ativos mais solicitados (como CSS, imagens e arquivos JavaScript). Os efeitos colaterais de minimização incluem encurtar os nomes de variável para um caractere e remover comentários e espaço em branco desnecessário.

Considere a seguinte função de JavaScript:

```
AddAltToImg = function (imageTagAndImageID, imageContext) {
    ///<signature>
    ///<summary> Adds an alt tab to the image
    // </summary>
    //<param name="imgElement" type="String">The image selector.</param>
    //<param name="ContextForImage" type="String">The image context.</param>
    //</signature>
    var imageElement = $(imageTagAndImageID, imageContext);
    imageElement.attr('alt', imageElement.attr('id').replace(/ID/, ''));
}
```

Minificação reduz a função para o seguinte:

```
AddAltToImg=function(t,a){var r=$(t,a);r.attr("alt",r.attr("id").replace(/\ID/,("")))};
```

Além de remover os comentários e espaço em branco desnecessário, os seguintes nomes de parâmetro e variável foram renomeados da seguinte maneira:

ORIGINAL	RENOMEADO
imageTagAndImageID	t
imageContext	a
imageElement	r

## Impacto de agrupamento e minificação

A tabela a seguir descreve as diferenças entre carregar ativos individualmente e usando o agrupamento e minificação:

AÇÃO	COM B/M	SEM B/M	ALTERAÇÃO
Solicitações de arquivos	7	18	157%
KB transferido	156	264.68	70%
Tempo de carregamento (ms)	885	2360	167%

Navegadores são bastante detalhados em relação a cabeçalhos de solicitação HTTP. O total de bytes enviados métrica viu uma redução significativa ao agrupamento. O tempo de carregamento mostra uma melhoria significativa, no entanto, este exemplo foi executado localmente. Maior ganhos de desempenho são obtidos ao usar o agrupamento e minificação com ativos transferidos por uma rede.

## Escolher uma estratégia de agrupamento e minificação

Os modelos de projeto do MVC e páginas Razor oferecem uma solução de out-of-the-box para agrupamento e minificação consiste em um arquivo de configuração JSON. Ferramentas de terceiros, tais como o [Gulp](#) e [Grunt](#) executores de tarefas, executar as mesmas tarefas com um pouco mais complexidade. Uma ferramenta de terceiros é uma excelente opção quando o fluxo de trabalho de desenvolvimento requer processamento além do agrupamento e minificação—como otimização linting e imagem. Usando o agrupamento e minificação tempo de design, os arquivos reduzidos são criados antes da implantação do aplicativo. Empacotando e minimizando antes da implantação tem a vantagem de carga do servidor reduzido. No entanto, é importante reconhecer que esse tempo de design de agrupamento e minificação aumenta a complexidade de build e só funciona com arquivos estáticos.

## Configurar o agrupamento e minificação

No ASP.NET Core 2.0 ou anterior, os modelos de projeto do MVC e páginas do Razor fornecem uma `bundleconfig.json` arquivo de configuração que define as opções para cada pacote:

No ASP.NET Core 2.1 ou posterior, adicione um novo arquivo JSON, denominado `bundleconfig.json`, para a raiz do projeto MVC ou páginas do Razor. Inclua o JSON a seguir no arquivo como um ponto de partida:

```

[
  {
    "outputFileName": "wwwroot/css/site.min.css",
    "inputFiles": [
      "wwwroot/css/site.css"
    ]
  },
  {
    "outputFileName": "wwwroot/js/site.min.js",
    "inputFiles": [
      "wwwroot/js/site.js"
    ],
    "minify": {
      "enabled": true,
      "renameLocals": true
    },
    "sourceMap": false
  }
]

```

O `bundleconfig.json` arquivo define as opções para cada pacote. No exemplo anterior, uma configuração de pacote único é definida para o JavaScript personalizado (`wwwroot/js/site.js`) e a folha de estilos (`wwwroot/css/site.css`) arquivos.

Opções de configuração incluem:

- `outputFileName` : O nome do arquivo de pacote de saída. Pode conter um caminho relativo do `bundleconfig.json` arquivo. **Necessário**
- `inputFiles` : Uma matriz de arquivos para agrupar. Esses são os caminhos relativos para o arquivo de configuração. **opcional**, \* um valor vazio resulta em um arquivo de saída vazia. [recurso de curinga](#) padrões são suportados.
- `minify` : As opções de minimização para o tipo de saída. **opcional, padrão**: `minify: { enabled: true }`
  - Opções de configuração estão disponíveis por tipo de arquivo de saída.
    - [Minificador CSS](#)
    - [Minificador de JavaScript](#)
    - [Minificador de HTML](#)
- `includeInProject` : O sinalizador que indica se deseja adicionar os arquivos gerados para o arquivo de projeto. **opcional, padrão – false**
- `sourceMap` : O sinalizador que indica se é necessário gerar um mapa de código-fonte para o arquivo agrupado. **opcional, padrão – false**
- `sourceMapRootPath` : O caminho raiz para armazenar o arquivo de mapa de código-fonte gerado.

## Compilação em tempo de execução de agrupamento e minificação

O [BuildBundlerMinifier](#) pacote NuGet permite que a execução de agrupamento e minificação no momento da compilação. O pacote injeta [destinos do MSBuild](#) quais executar na compilação e de hora limpa. O `bundleconfig.json` arquivo é analisado pelo processo de compilação para produzir os arquivos de saída com base na configuração definida.

### NOTE

BuildBundlerMinifier pertence a um projeto voltado à comunidade no GitHub para o qual a Microsoft fornece sem suporte. Problemas devem ser arquivados [aqui](#).

- [Visual Studio](#)

- [CLI do .NET Core](#)

Adicione a *BuildBundlerMinifier* pacote ao seu projeto.

Compile o projeto. O exemplo a seguir é exibida na janela de saída:

```
1>----- Build started: Project: BuildBundlerMinifierApp, Configuration: Debug Any CPU -----
1>
1>Bundler: Begin processing bundleconfig.json
1>  Minified wwwroot/css/site.min.css
1>  Minified wwwroot/js/site.min.js
1>Bundler: Done processing bundleconfig.json
1>BuildBundlerMinifierApp -> C:\BuildBundlerMinifierApp\bin\Debug\netcoreapp2.0\BuildBundlerMinifierApp.dll
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

Limpe o projeto. O exemplo a seguir é exibida na janela de saída:

```
1>----- Clean started: Project: BuildBundlerMinifierApp, Configuration: Debug Any CPU -----
1>
1>Bundler: Cleaning output from bundleconfig.json
1>Bundler: Done cleaning output file from bundleconfig.json
===== Clean: 1 succeeded, 0 failed, 0 skipped =====
```

## Execução ad hoc de agrupamento e minificação

É possível executar as tarefas de empacotamento e minimização em uma base ad hoc, sem compilar o projeto.

Adicione a [BundlerMinifier.Core](#) pacote NuGet ao seu projeto:

```
<DotNetCliToolReference Include="BundlerMinifier.Core" Version="2.6.362" />
```

### NOTE

BundlerMinifier.Core pertence a um projeto voltado à comunidade no GitHub para o qual a Microsoft fornece sem suporte. Problemas devem ser arquivados [aqui](#).

Este pacote estende a CLI do .NET Core para incluir a *pacote dotnet* ferramenta. O comando a seguir pode ser executado na janela do Console de Gerenciador de pacote (PMC) ou em um shell de comando:

```
dotnet bundle
```

### IMPORTANT

Gerenciador de pacotes NuGet adiciona as dependências para o arquivo \*.csproj como `<PackageReference />` nós. O `dotnet bundle` comando está registrado com a CLI do .NET Core somente quando um `<DotNetCliToolReference />` nó é usado. Modifique o arquivo \*.csproj adequadamente.

## Adicionar arquivos ao fluxo de trabalho

Considere um exemplo no qual adicional *Custom.css* arquivo for adicionado a seguir:

```
.about, [role=main], [role=complementary] {  
    margin-top: 60px;  
}  
  
footer {  
    margin-top: 10px;  
}
```

Para minificar *Custom. CSS* e agrupá-la com *CSS* em um *site* arquivo, adicione o caminho relativo para *bundleconfig.json*:

```
[  
{  
    "outputFileName": "wwwroot/css/site.min.css",  
    "inputFiles": [  
        "wwwroot/css/site.css",  
        "wwwroot/css/custom.css"  
    ]  
},  
{  
    "outputFileName": "wwwroot/js/site.min.js",  
    "inputFiles": [  
        "wwwroot/js/site.js"  
    ],  
    "minify": {  
        "enabled": true,  
        "renameLocals": true  
    },  
    "sourceMap": false  
}  
]
```

#### NOTE

Como alternativa, é possível usar o seguinte padrão de recurso de curinga:

```
"inputFiles": [ "wwwroot/**/*(*.css|!(*.min.css))" ]
```

Esse padrão de recurso de curinga corresponde a todos os arquivos CSS e exclui o padrão de arquivo reduzido.

Compile o aplicativo. Abra *site* e observe o conteúdo da *Custom. CSS* é acrescentado ao final do arquivo.

## Agrupamento e minificação com base no ambiente

Como prática recomendada, os arquivos agrupados e minificados do seu aplicativo devem ser usados em um ambiente de produção. Durante o desenvolvimento, os arquivos originais fazem para depuração mais fácil do aplicativo.

Especificar quais arquivos serão incluídos em suas páginas usando o [auxiliar de marca de ambiente](#) em modos de exibição. O auxiliar de marca de ambiente renderiza apenas seu conteúdo ao executar em particular [ambientes](#).

O seguinte `environment` marca renderiza os arquivos CSS não processados durante a execução no `Development` ambiente:

```
<environment include="Development">
  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
  <link rel="stylesheet" href("~/css/site.css" />
</environment>
```

```
<environment names="Development">
  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
  <link rel="stylesheet" href "~/css/site.css" />
</environment>
```

O seguinte `environment` marca renderiza os arquivos CSS agrupados e minificados quando em execução em um ambiente diferente de `Development`. Por exemplo, em execução no `Production` ou `Staging` dispara o processamento dessas folhas de estilo:

```
<environment exclude="Development">
  <link rel="stylesheet" href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/css/bootstrap.min.css"
        asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"
        asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-test-
        value="absolute" />
  <link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true" />
</environment>
```

```
<environment names="Staging,Production">
  <link rel="stylesheet" href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/css/bootstrap.min.css"
        asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"
        asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-test-
        value="absolute" />
  <link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true" />
</environment>
```

## Consumir bundleconfig.json do Gulp

Há casos em que o fluxo de trabalho agrupamento e minificação do aplicativo requer processamento adicional. Exemplos incluem processamento de ativos da CDN, extração de cache e otimização de imagem. Para atender a esses requisitos, você pode converter o fluxo de trabalho de agrupamento e minificação para usar o Gulp.

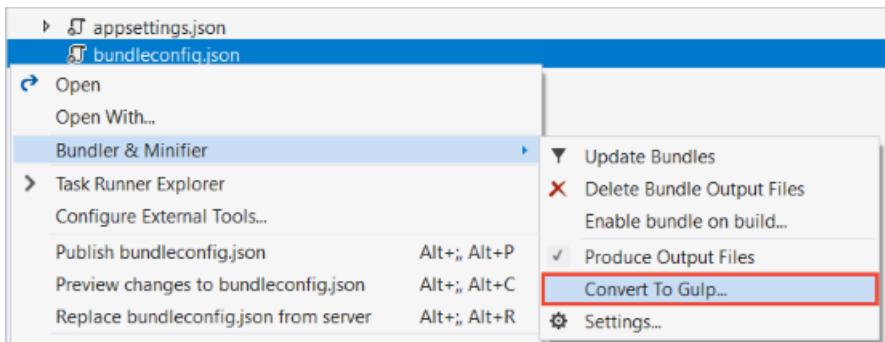
### Usar a extensão Bundler & Minifier

O Visual Studio [Bundler & Minifier](#) extensão lida com a conversão para o Gulp.

#### NOTE

A extensão Bundler & Minifier pertence a um projeto voltado à comunidade no GitHub para o qual a Microsoft fornece sem suporte. Problemas devem ser arquivados [aqui](#).

Clique com botão direito do `bundleconfig.json` arquivo no Gerenciador de soluções e selecione **Bundler & Minifier > converter Gulp...**:



O *gulpfile.js* e *Package. JSON* arquivos são adicionados ao projeto. O suporte a [npm](#) os pacotes listados na *Package. JSON* do arquivo `devDependencies` seção estão instalados.

Execute o seguinte comando na janela do PMC para instalar a CLI do Gulp como uma dependência global:

```
npm i -g gulp-cli
```

O *gulpfile.js* leituras de arquivos a *bundleconfig.json* arquivo para as entradas, saídas e as configurações.

```
'use strict';

var gulp = require('gulp'),
    concat = require('gulp-concat'),
    cssmin = require('gulp-cssmin'),
    htmlmin = require('gulp-htmlmin'),
    uglify = require('gulp-uglify'),
    merge = require('merge-stream'),
    del = require('del'),
    bundleconfig = require('./bundleconfig.json');

// Code omitted for brevity
```

## Converter manualmente

Se o Visual Studio e/ou a extensão Bundler & Minifier não estiverem disponível, converta manualmente.

Adicionar um *Package. JSON* arquivo pelo seguinte `devDependencies`, para a raiz do projeto:

```
"devDependencies": {
  "del": "^3.0.0",
  "gulp": "^4.0.0",
  "gulp-concat": "^2.6.1",
  "gulp-cssmin": "^0.2.0",
  "gulp-htmlmin": "^3.0.0",
  "gulp-uglify": "^3.0.0",
  "merge-stream": "^1.0.1"
}
```

Instalar as dependências, executando o comando a seguir no mesmo nível que *Package. JSON*:

```
npm i
```

Instale a CLI do Gulp como uma dependência global:

```
npm i -g gulp-cli
```

Cópia de *gulpfile.js* abaixo o arquivo para a raiz do projeto:

```

'use strict';

var gulp = require('gulp'),
concat = require('gulp-concat'),
cssmin = require('gulp-cssmin'),
htmlmin = require('gulp-htmlmin'),
uglify = require('gulp-uglify'),
merge = require('merge-stream'),
del = require('del'),
bundleconfig = require('./bundleconfig.json');

const regex = {
  css: /\.css$/,
  html: /\.(html|htm)$/,
  js: /\.js$/
};

gulp.task('min:js',
() => merge(getBundles(regex.js).map(bundle => {
  return gulp.src(bundle.inputFiles, { base: '.' })
    .pipe(concat(bundle.outputFileName))
    .pipe(uglify())
    .pipe(gulp.dest('.'));
})));

gulp.task('min:css',
() => merge(getBundles(regex.css).map(bundle => {
  return gulp.src(bundle.inputFiles, { base: '.' })
    .pipe(concat(bundle.outputFileName))
    .pipe(cssmin())
    .pipe(gulp.dest('.'));
})));

gulp.task('min:html',
() => merge(getBundles(regex.html).map(bundle => {
  return gulp.src(bundle.inputFiles, { base: '.' })
    .pipe(concat(bundle.outputFileName))
    .pipe(htmlmin({ collapseWhitespace: true, minifyCSS: true, minifyJS: true }))
    .pipe(gulp.dest('.'));
})));

gulp.task('min', gulp.series(['min:js', 'min:css', 'min:html']));

gulp.task('clean', () => {
  return del(bundleconfig.map(bundle => bundle.outputFileName));
});

gulp.task('watch', () => {
  getBundles(regex.js).forEach(
    bundle => gulp.watch(bundle.inputFiles, gulp.series(["min:js"])));
  getBundles(regex.css).forEach(
    bundle => gulp.watch(bundle.inputFiles, gulp.series(["min:css"])));
  getBundles(regex.html).forEach(
    bundle => gulp.watch(bundle.inputFiles, gulp.series(['min:html'])));
});

const getBundles = (regexPattern) => {
  return bundleconfig.filter(bundle => {
    return regexPattern.test(bundle.outputFileName);
  });
};

gulp.task('default', gulp.series(['min']));

```

## Executar tarefas de Gulp

Para disparar a tarefa do Gulp minificação antes que o projeto é compilado no Visual Studio, adicione o seguinte destino do MSBuild para o arquivo \*. csproj:

```
<Target Name="MyPreCompileTarget" BeforeTargets="Build">
  <Exec Command="gulp min" />
</Target>
```

Neste exemplo, as tarefas definidas dentro de `MyPreCompileTarget` execução antes do predefinidos de destino `Build` destino. Saída semelhante à seguinte é exibida na janela de saída do Visual Studio:

```
1>----- Build started: Project: BuildBundlerMinifierApp, Configuration: Debug Any CPU -----
1>BuildBundlerMinifierApp -> C:\BuildBundlerMinifierApp\bin\Debug\netcoreapp2.0\BuildBundlerMinifierApp.dll
1>[14:17:49] Using gulpfile C:\BuildBundlerMinifierApp\gulpfile.js
1>[14:17:49] Starting 'min:js'...
1>[14:17:49] Starting 'min:css'...
1>[14:17:49] Starting 'min:html'...
1>[14:17:49] Finished 'min:js' after 83 ms
1>[14:17:49] Finished 'min:css' after 88 ms
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

Como alternativa, o Gerenciador de executor de tarefas do Visual Studio pode ser usado para associar as tarefas de Gulp a eventos específicos do Visual Studio. Ver [execução de tarefas padrão](#) para obter instruções sobre como fazer isso.

## Recursos adicionais

- [Usar o Gulp](#)
- [Usar o Grunt](#)
- [Usar vários ambientes](#)
- [Auxiliares de marcação](#)

# Link do navegador no ASP.NET Core

23/08/2018 • 8 minutes to read • [Edit Online](#)

Por [Nicolò Carandini](#), [Mike Wasson](#), e [Tom Dykstra](#)

Link do navegador é um recurso no Visual Studio que cria um canal de comunicação entre o ambiente de desenvolvimento e um ou mais navegadores da web. Você pode usar o Link do navegador para atualizar seu aplicativo web em vários navegadores ao mesmo tempo, que é útil para testes entre navegadores.

## Configuração do Link de navegador

Ao converter um projeto do ASP.NET Core 2.0 para ASP.NET Core 2.1 e fazer a transição para o [metapacote do Microsoft](#), instale o [browserlink](#) do pacote para Funcionalidade BrowserLink. Usam os modelos de projeto do ASP.NET Core 2.1 a [Microsoft.AspNetCore.App](#) metapacote por padrão.

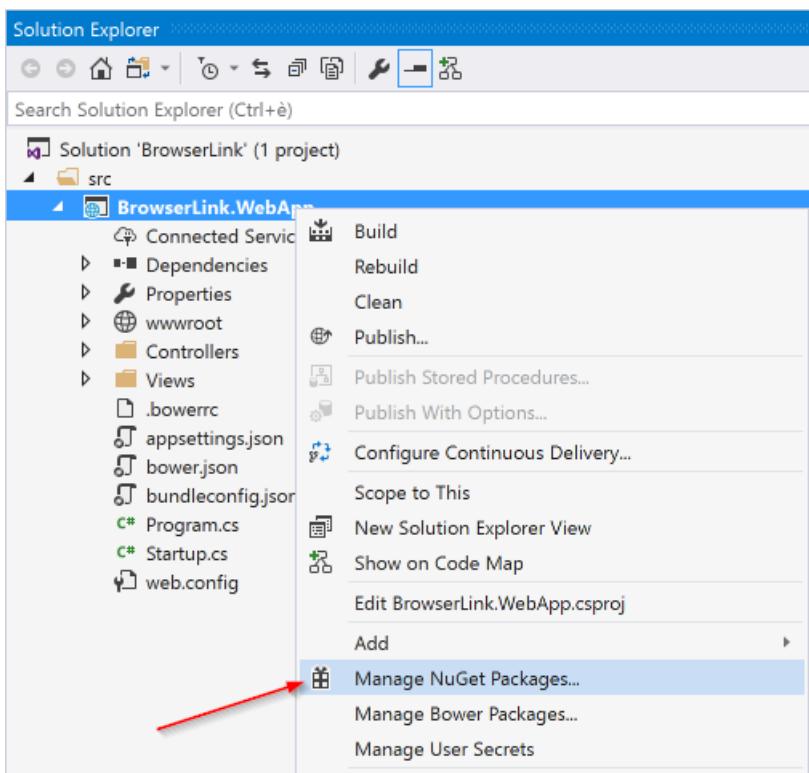
O ASP.NET Core 2.0 **aplicativo Web, vazia, e API da Web** uso de modelos de projeto a [Microsoft.AspNetCore.All](#) metapacote , que contém uma referência de pacote para [browserlink](#). Portanto, usando o [Microsoft.AspNetCore.All](#) metapacote não requer nenhuma ação adicional para disponibilizar o Link do navegador para uso.

O ASP.NET Core 1.x **aplicativo Web** modelo de projeto possui uma referência de pacote para o [browserlink](#) pacote. O **vazio** ou **API da Web** exigem que você adicione uma referência de pacote para projetos de modelo [Microsoft.VisualStudio.Web.BrowserLink](#) .

Como esse é um recurso do Visual Studio, a maneira mais fácil para adicionar o pacote para um **vazio** ou **API da Web** projeto modelo é abrir o **Package Manager Console (Modo de exibição > Other Windows > Package Manager Console)** e execute o seguinte comando:

```
install-package Microsoft.VisualStudio.Web.BrowserLink
```

Como alternativa, você pode usar **Gerenciador de pacotes NuGet**. Clique com botão direito no nome do projeto no **Gerenciador de soluções** e escolha **Manage NuGet Packages**:



Localizar e instalar o pacote:

The screenshot shows the NuGet Package Manager interface for the 'NuGet: BrowserLink.WebApp' package. The search bar contains 'Microsoft.VisualStudio.Web.BrowserLink.Loader'. The results show a single item: 'Microsoft.VisualStudio.Web.BrowserLink.Loader' by Microsoft, version 14.1.0. The description states it's a middleware for creating a communication channel between the development environment and one or more web browsers. On the right, the 'Install' button for the latest stable version (14.1.0) is highlighted with a red arrow.

## Configuração

No método `Startup.Configure`:

```
app.UseBrowserLink();
```

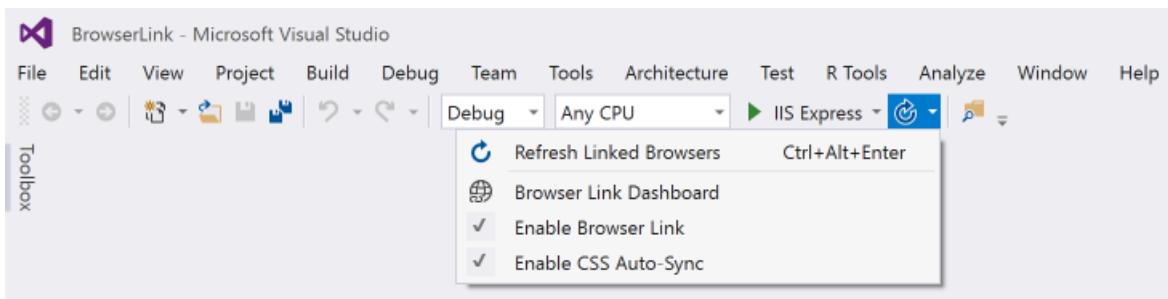
Normalmente, o código está dentro de um `if` bloco que apenas permite que o Link do navegador no ambiente de desenvolvimento, como mostrado aqui:

```
if (env.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
    app.UseBrowserLink();
}
```

Para obter mais informações, veja [Usar vários ambientes](#).

## Como usar o Link do navegador

Quando um projeto do ASP.NET Core está aberto, o Visual Studio mostra o controle de barra de ferramentas do Link do navegador ao lado do controle de barra de ferramentas do **Destino de Depuração**:



O controle de barra de ferramentas do Link do navegador, você pode:

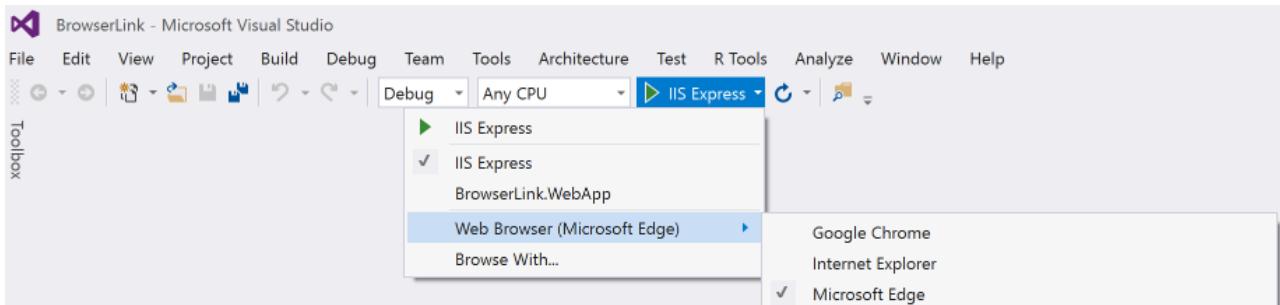
- Atualizar o aplicativo Web em vários navegadores de uma vez.
- Abrir o **Painel de link do navegador**.
- Habilitar ou desabilitar **Link do navegador**. Observação: O Link do navegador está desabilitado por padrão no Visual Studio 2017 (15.3).
- Habilitar ou desabilitar [sincronização automática de CSS](#).

#### NOTE

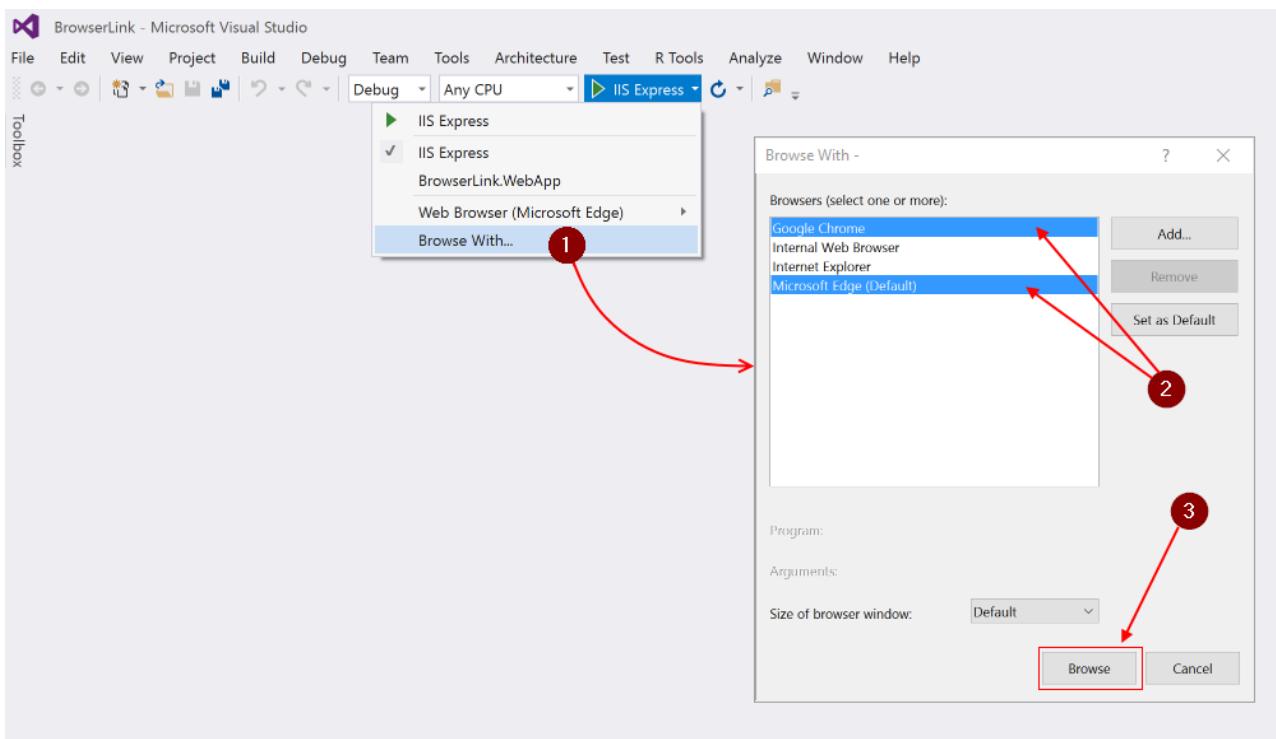
Alguns plug-ins do Visual Studio, mais notavelmente *Web extensão Pack 2015 e 2017 de pacote de extensão do Web*, oferecem funcionalidade estendida para o Link do navegador, mas alguns dos recursos adicionais não funcionam com o ASP. Projetos do .NET Core.

## Atualizar o aplicativo web em vários navegadores ao mesmo tempo

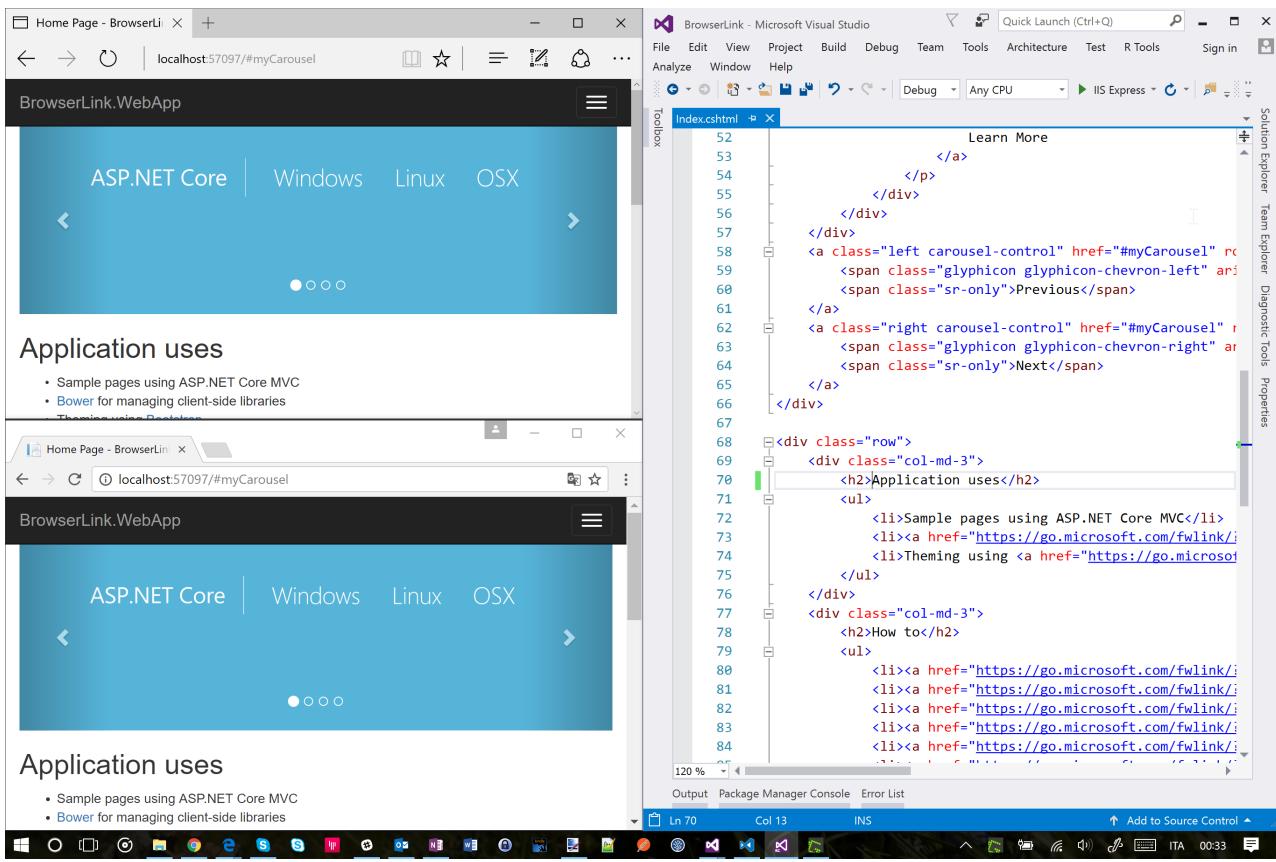
Para escolher um único navegador Web para abrir ao iniciar o projeto, use o menu suspenso do controle de barra de ferramentas do **Destino de depuração**:



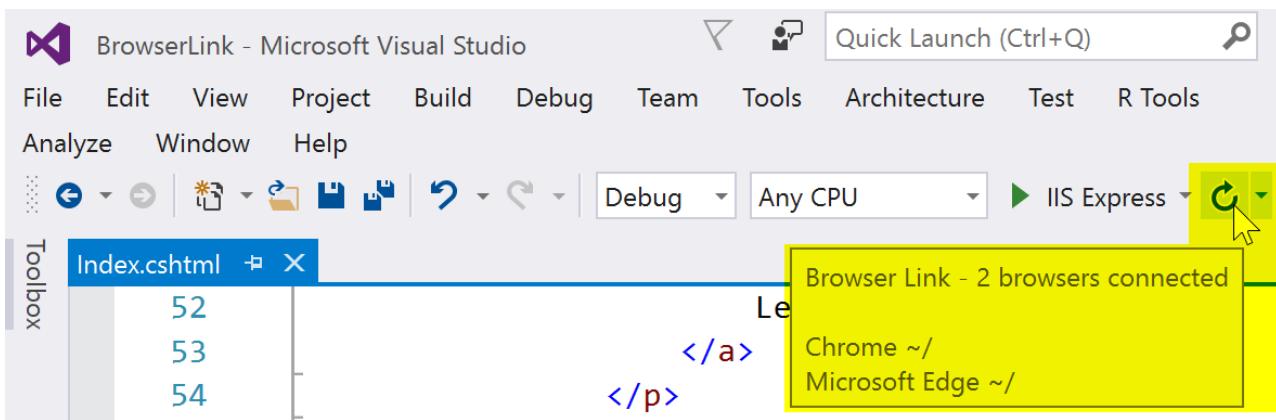
Para abrir vários navegadores, ao mesmo tempo, escolha **procurar com...** da mesma lista suspensa. Mantenha pressionada a tecla CTRL para selecionar os navegadores que você deseja e, em seguida, clique em **procurar**:



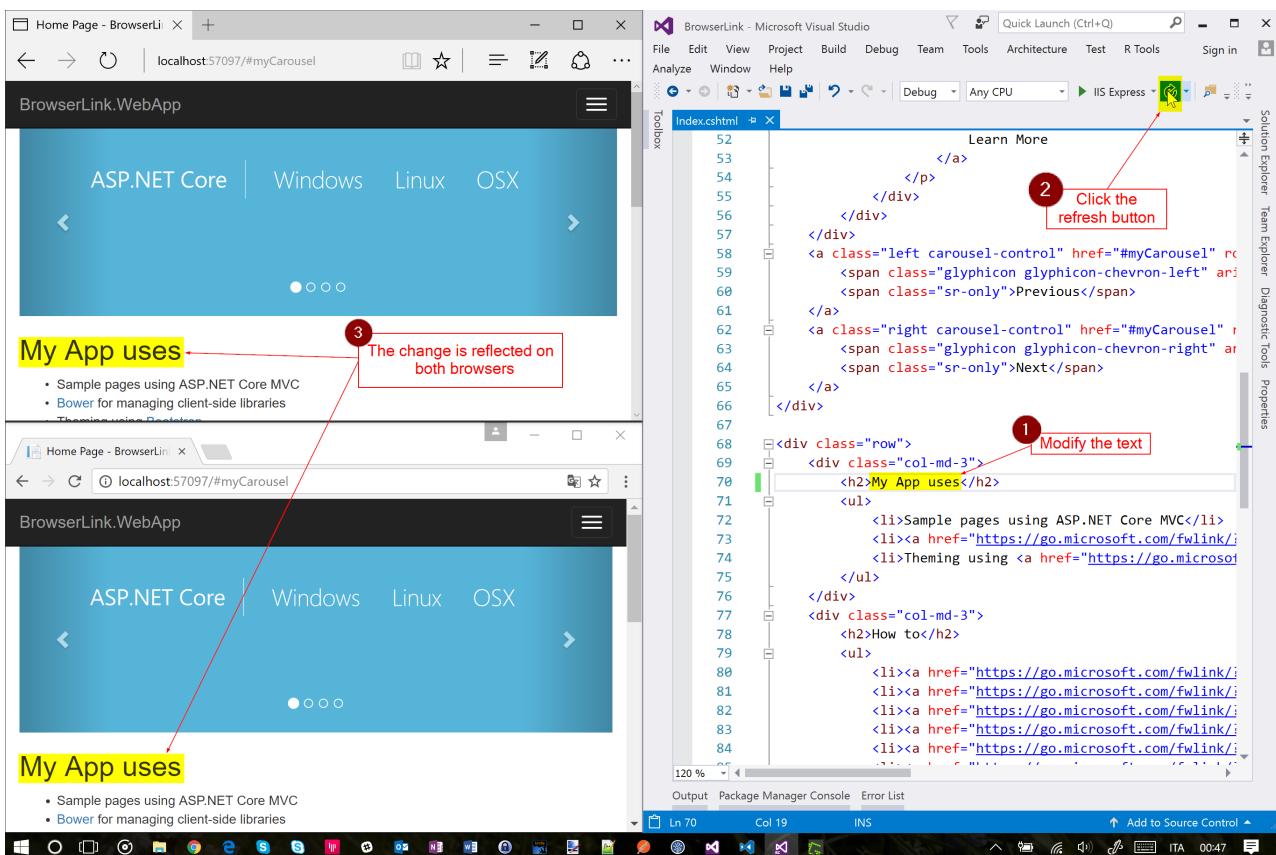
Aqui está uma captura de tela mostrando o Visual Studio com o modo de exibição de índice aberto e dois navegadores abertas:



Passe o mouse sobre o controle de barra de ferramentas do Link do navegador para ver os navegadores que estão conectados ao projeto:



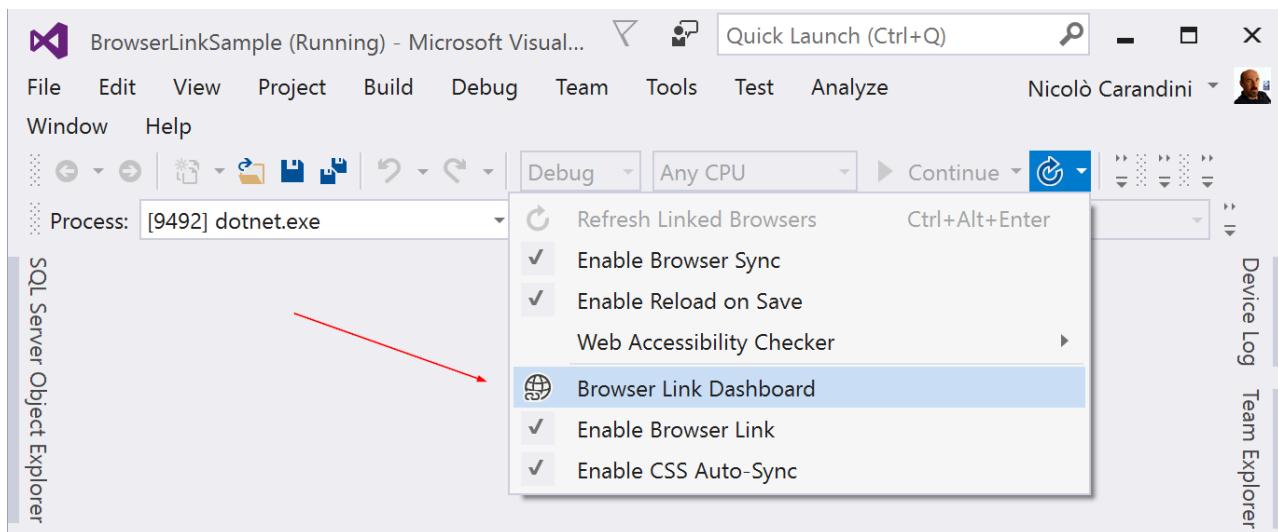
Altere o modo de exibição de índice e todos os navegadores conectados são atualizados quando você clicar no botão de atualização de Link do navegador:



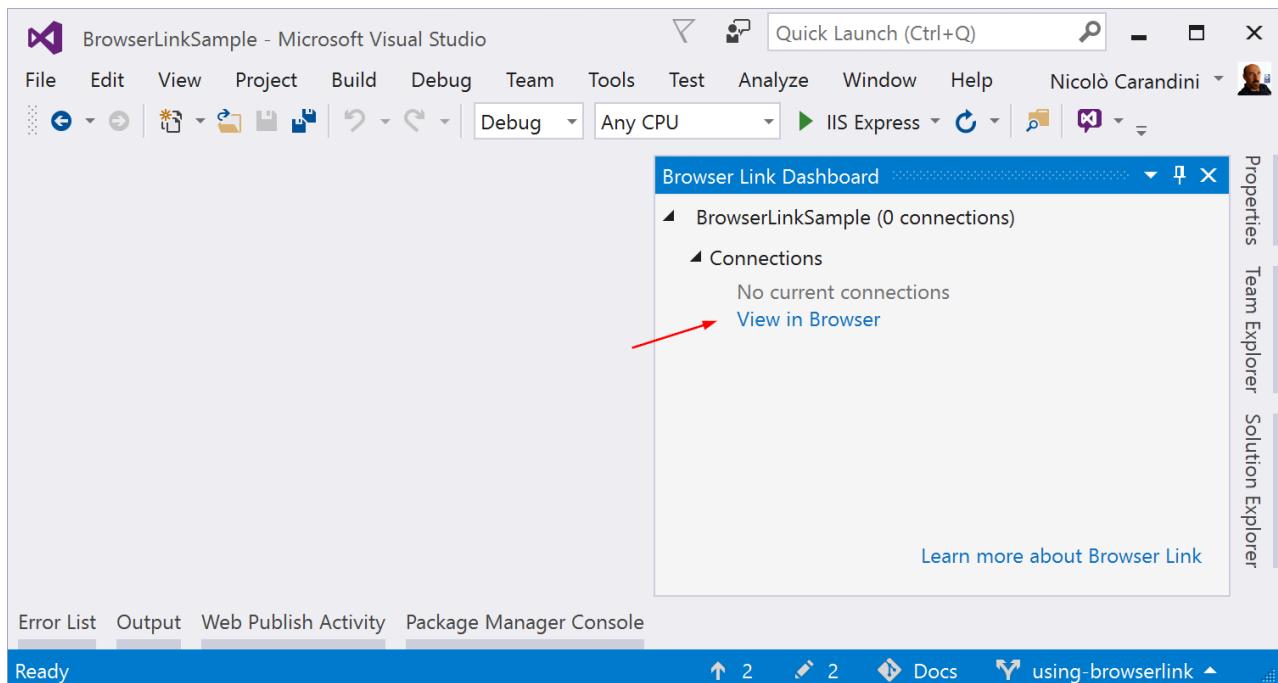
Link do navegador também funciona com navegadores que você inicie de fora do Visual Studio e navegue até a URL do aplicativo.

## O painel de Link do navegador

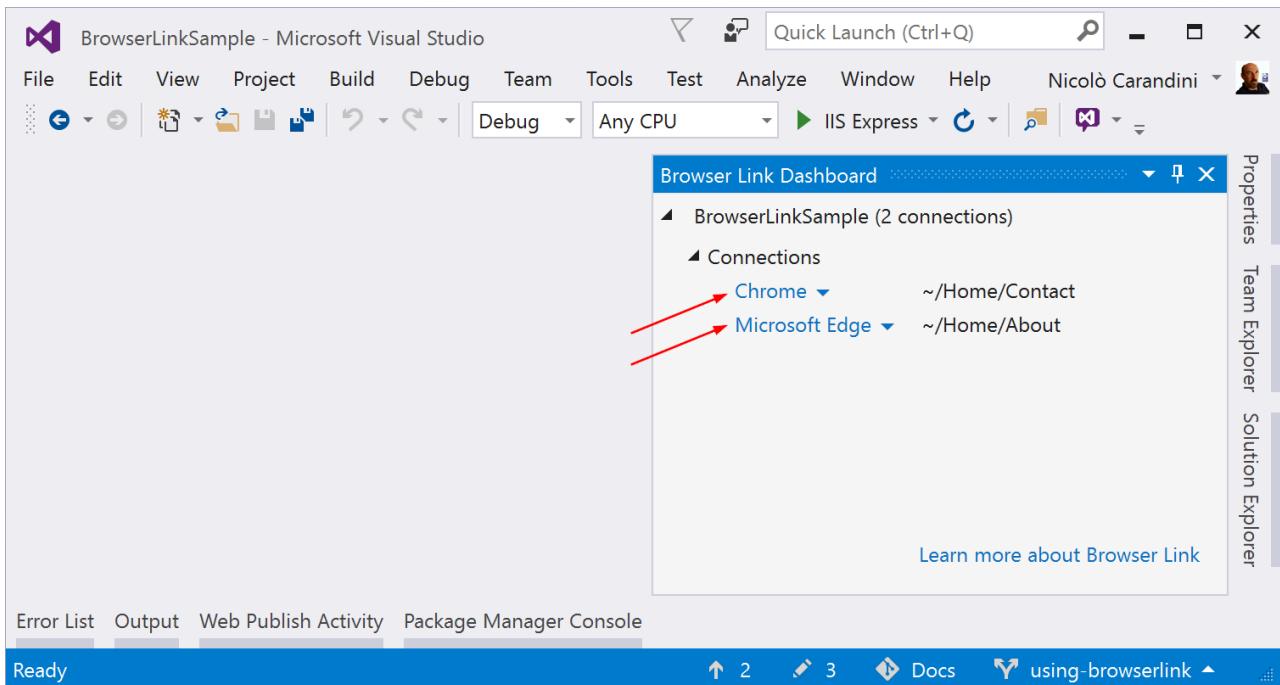
Abra o painel de Link do navegador de menu para gerenciar a conexão com o navegador abertas suspenso Link do navegador:



Se nenhum navegador estiver conectado, você pode iniciar uma sessão de depuração não, selecionando Se nenhum navegador estiver conectado, você poderá iniciar uma sessão de não depuração selecionando o link *exibir no navegador*:



Caso contrário, os navegadores conectados são mostrados com o caminho para a página que mostra cada navegador:



Se desejar, você pode clicar em um nome de navegador listados para atualizar esse único navegador.

### Habilitar ou desabilitar o Link do navegador

Quando você habilita novamente o Link do navegador depois de desabilitá-lo, você deve atualizar os navegadores para reconectar-se-los.

### Habilitar ou desabilitar a sincronização automática de CSS

Quando a sincronização automática de CSS está habilitada, os navegadores conectados serão atualizados automaticamente quando você fizer qualquer alteração em arquivos CSS.

## Como ele funciona

Link do navegador usa o SignalR para criar um canal de comunicação entre o Visual Studio e o navegador. Quando o Link do navegador está habilitado, o Visual Studio atua como um servidor de SignalR que vários clientes (navegadores) podem se conectar ao. Link do navegador também registra um componente de middleware no pipeline de solicitação do ASP.NET Core. Esse componente injeta especial `<script>` referências em cada solicitação de página do servidor. Você pode ver as referências de script selecionando **Exibir código-fonte** no navegador e rolar até o final do `<body>` conteúdo de marca:

```
<!-- Visual Studio Browser Link -->
<script type="application/json" id="__browserLink_initializationData">
    {"requestId":"a717d5a07c1741949a7cefd6fa2bad08","requestMappingFromServer":false}
</script>
<script type="text/javascript" src="http://localhost:54139/b6e36e429d034f578ebcccd6a79bf19bf/browserLink"
async="async"></script>
<!-- End Browser Link -->
</body>
```

Os arquivos de origem não são modificados. O componente de middleware injeta as referências de script dinamicamente.

Como o código do navegador é todo JavaScript, ele funciona em todos os navegadores. SignalR dá suporte a sem a necessidade de um plug-in de navegador.

# Introdução aos Componentes Razor

04/02/2019 • 13 minutes to read • [Edit Online](#)

Por [Steve Sanderson](#), [Daniel Roth](#) e [Luke Latham](#)

## NOTE

O Razor Components do ASP.NET Core é compatível com ASP.NET Core 3.0 ou posterior.

Blazor é uma estrutura da Web sem suporte e experimental que não deve ser usada para cargas de trabalho de produção no momento.

Os *Componentes Razor do ASP.NET Core* executam o lado do servidor no ASP.NET Core, enquanto o *Blazor* (Componentes Razor que executam o lado do cliente) é uma estrutura da Web experimental do .NET que usa C#/Razor e HTML executados no navegador com o WebAssembly. O Blazor fornece todos os benefícios de uma estrutura de interface do usuário da Web do lado do cliente usando o .NET no cliente.

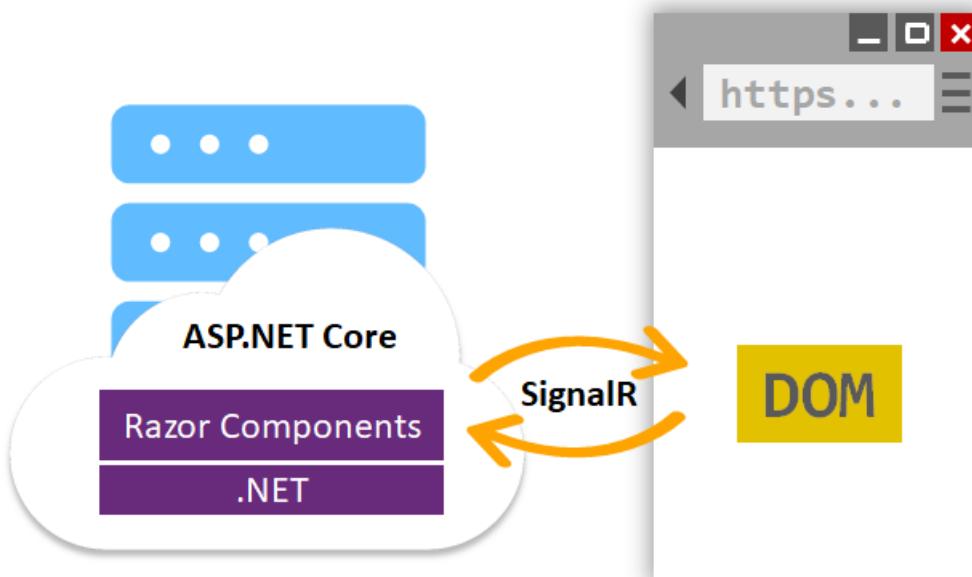
O desenvolvimento para a Web foi aprimorado de diversas maneiras ao longo dos anos, mas a criação de aplicativos Web modernos ainda apresenta desafios. O uso do .NET no navegador oferece muitas vantagens que podem ajudar a tornar o desenvolvimento para Web mais fácil e mais produtivo:

- **Estabilidade e consistência:** o .NET fornece estruturas de programação padronizadas entre plataformas, que são estáveis, completas e fáceis de usar.
- **Linguagens modernas inovadoras:** as linguagens do .NET estão melhorando constantemente com novos recursos de linguagem inovadores.
- **Ferramentas líderes do setor:** a família de produtos do Visual Studio fornece uma experiência fantástica de desenvolvimento no .NET entre plataformas, em Windows, Linux e macOS.
- **Velocidade e escalabilidade:** o .NET tem um forte histórico de desempenho, confiabilidade e segurança para o desenvolvimento de aplicativos. O uso do .NET como uma solução de pilha completa facilita a criação de aplicativos rápidos, confiáveis e seguros.
- **Desenvolvimento de pilha completa que aproveita as habilidades existentes:** os desenvolvedores C#/Razor usam as habilidades em C#/Razor que eles já têm para escrever o código do lado do cliente e compartilhar a lógica do lado do servidor e do lado do cliente entre os aplicativos.
- **Amplo suporte a navegadores:** os Componentes Razor renderizam a interface do usuário como uma marcação e um JavaScript comuns. O Blazor é executado no .NET, no navegador, usando padrões abertos da Web sem nenhum plug-in e nenhuma transpilação de código. O Blazor funciona em todos os navegadores da Web modernos, incluindo os navegadores móveis.

## Modelos de hospedagem

### Modelo de hospedagem do lado do servidor

Como os Componentes Razor desvinculam a lógica de renderização de um componente da maneira de aplicar as atualizações da interface do usuário, há flexibilidade na maneira de hospedar os Componentes Razor. Os Componentes Razor do ASP.NET Core no .NET Core 3.0 adicionam suporte para a hospedagem dos Componentes Razor no servidor, em um aplicativo ASP.NET Core, em que todas as atualizações da interface do usuário são realizadas por uma conexão SignalR. O tempo de execução realiza o envio de eventos da interface do usuário do navegador para o servidor e, depois de executar os componentes, aplica as atualizações na interface do usuário retornadas pelo servidor ao navegador. A mesma conexão também é usada para realizar as chamadas de interoperabilidade do JavaScript.



Para obter mais informações, consulte [Razor Components hosting models](#).

### Modelo de hospedagem do lado do cliente

A execução do código do .NET em navegadores da Web é possibilitada por uma tecnologia relativamente nova, o [WebAssembly](#) (abreviado como *wasm*). O WebAssembly é um padrão aberto da Web compatível com navegadores da Web sem plug-ins. O WebAssembly é um formato de código de bytes compacto, otimizado para download rápido e máxima velocidade de execução.

O código do WebAssembly pode acessar a funcionalidade completa do navegador por meio da interoperabilidade do JavaScript. Ao mesmo tempo, o código do WebAssembly é executado na mesma área restrita confiável que o JavaScript para impedir ações mal-intencionadas no computador cliente.

Quando um aplicativo Blazor é criado e executado em um navegador:

1. os arquivos C# e os arquivos Razor são compilados em assemblies do .NET.
2. os assemblies e o tempo de execução do .NET são baixados no navegador.
3. o Blazor usa o JavaScript para inicializar o tempo de execução do .NET e configura o tempo de execução para carregar as referências de assembly necessárias. A manipulação do DOM (Modelo de Objeto do Documento) e as chamadas à API do navegador são realizadas pelo tempo de execução do Blazor por meio da interoperabilidade do JavaScript.

Para dar suporte a navegadores mais antigos que não são compatíveis com o WebAssembly, você pode usar o [modelo de hospedagem do lado do servidor](#) dos Componentes Razor do ASP.NET Core.

Para obter mais informações, consulte [Razor Components hosting models](#).

## Componentes

Os aplicativos são criados com *componentes*. Um componente é uma parte da interface do usuário, como uma página, uma caixa de diálogo ou um formulário de entrada de dados. Os componentes podem ser aninhados, reutilizados e compartilhados entre os projetos.

Um *componente* é uma classe do .NET. A classe também pode ser escrita diretamente, como uma classe C# (\*.cs) ou, com mais frequência, na forma de uma página de marcação Razor (\*.cshtml).

O [Razor](#) é uma sintaxe para a combinação da marcação HTML com o código C#. O Razor foi projetado visando a produtividade do desenvolvedor, permitindo que ele mude entre a marcação e o C# no mesmo arquivo com o

suporte do [IntelliSense](#). A marcação a seguir é um exemplo de um componente básico de caixa de diálogo personalizada em um arquivo Razor (*DialogComponent.cshtml*):

```
<div>
    <h2>@Title</h2>
    @BodyContent
    <button onclick=@OnOK>OK</button>
</div>

@functions {
    public string Title { get; set; }
    public RenderFragment BodyContent { get; set; }
    public Action OnOK { get; set; }
}
```

Quando esse componente é usado em outro lugar no aplicativo, o IntelliSense acelera o desenvolvimento com o preenchimento de sintaxe e de parâmetro.

Os componentes podem:

- ser aninhados.
- ser criados com o Razor (\*.cshtml) ou com o código C# (\*.cs).
- ser compartilhados por meio de bibliotecas de classes.
- receber um teste de unidade sem precisar de um navegador DOM.

## Infraestrutura

Os Componentes Razor oferecem os principais recursos que a maioria dos aplicativos exige, como:

- Layouts
- Roteamento
- Injeção de dependência

Todos esses recursos são opcionais. Quando um desses recursos não é usado em um aplicativo, a implementação é eliminada do aplicativo quando ele é publicado pelo [Vinculador de IL \(linguagem intermediária\)](#).

## Compartilhamento de código e o .NET Standard

Os aplicativos podem referenciar e usar as bibliotecas [.NET Standard](#) existentes. O .NET Standard é uma especificação formal das APIs do .NET que são comuns entre as implementações do .NET. Há suporte para o .NET standard 2.0 ou superior. As APIs que não são aplicáveis em um navegador da Web (por exemplo, para acessar o sistema de arquivos, abrir um soquete, threading e outros recursos) geram a [PlatformNotSupportedException](#). As bibliotecas de classes do .NET Standard podem ser compartilhadas entre o código do servidor e em aplicativos baseados em navegador.

## Interoperabilidade do JavaScript

Para aplicativos que exigem bibliotecas JavaScript e APIs do navegador de terceiros, o WebAssembly foi projetado para interoperar com o JavaScript. Os Componentes Razor são capazes de usar qualquer biblioteca ou API que o JavaScript possa usar. O código C# pode chamar o código JavaScript, e o código JavaScript pode chamar o código C#. Para obter mais informações, confira [Interoperabilidade do JavaScript](#).

## Otimização

Para aplicativos do lado do cliente, o tamanho do conteúdo é crítico. O Blazor otimiza o tamanho do conteúdo para reduzir os tempos de download. Por exemplo, as partes não usadas dos assemblies do .NET são removidas

durante o processo de build, as respostas HTTP são compactadas e o tempo de execução do .NET e os assemblies são armazenados em cache no navegador.

Os Componentes Razor fornecem um tamanho de conteúdo ainda menor do que o Blazor mantendo os assemblies do .NET, o assembly do aplicativo e o lado do servidor do tempo de execução. Os aplicativos dos Componentes Razor fornecem somente marcação, scripts e folhas de estilos aos clientes.

## Implantação

Use o Blazor para criar um aplicativo do lado do cliente autônomo puro ou um aplicativo do ASP.NET Core de pilha completa que contenha os aplicativos cliente e os aplicativos para servidor:

- Em um **aplicativo autônomo do lado do cliente**, o aplicativo Blazor é compilado em uma pasta *dist* que contém apenas arquivos estáticos. Os arquivos podem ser hospedados no Serviço de Aplicativo do Azure, nas páginas do GitHub, no IIS (configurado como um servidor de arquivos estático), nos servidores do Nodejs e em vários outros servidores e serviços. O .NET não é necessário no servidor de produção.
- Em um **aplicativo do ASP.NET Core de pilha completa**, o código pode ser compartilhado entre os aplicativos cliente e os aplicativos para servidor. O aplicativo dos Componentes Razor do ASP.NET Core resultante, que atende à interface do usuário do lado do cliente e a outros pontos de extremidade de API do lado do servidor, pode ser criado e implantado em qualquer host de nuvem ou local compatível com o ASP.NET Core.

## Sugerir um recurso ou enviar um relatório de bug

Para fazer sugestões e enviar relatórios de bug, [abra um problema](#). Para obter ajuda geral e respostas da comunidade, participe da conversa no [Gitter](#).

## Recursos adicionais

- [WebAssembly](#)
- [Guia do C#](#)
- [Razor](#)
- [HTML](#)

# Hospedar e implantar Componentes Razor

05/02/2019 • 25 minutes to read • [Edit Online](#)

Por [Luke Latham](#), [Rainer Stropek](#) e [Daniel Roth](#)

## NOTE

O Razor Components do ASP.NET Core é compatível com ASP.NET Core 3.0 ou posterior.

Blazor é uma estrutura da Web sem suporte e experimental que não deve ser usada para cargas de trabalho de produção no momento.

## Publique o aplicativo

Os aplicativos são publicados para implantação na configuração de versão com o comando `dotnet publish`. Um IDE pode manipular a execução do comando `dotnet publish` automaticamente usando recursos de publicação internos, para que não seja necessário executar o comando manualmente usando um prompt de comando, dependendo das ferramentas de desenvolvimento em uso.

```
dotnet publish -c Release
```

O `dotnet publish` dispara uma [restauração](#) das dependências do projeto e [compila](#) o projeto antes de criar os ativos para implantação. Como parte do processo de build, os assemblies e métodos não usados são removidos para reduzir o tamanho de download do aplicativo e os tempos de carregamento. A implantação é criada na pasta `/bin/Release/<target-framework>/publish`.

Os ativos na pasta `publish` são implantados no servidor Web. A implantação pode ser um processo manual ou automatizado, dependendo das ferramentas de desenvolvimento em uso.

## Valores de configuração do host

Os aplicativos dos Componentes Razor que usam o [modelo de hospedagem do lado do servidor](#) podem aceitar os [valores de configuração do host da Web](#).

Os aplicativos do Blazor que usam o [modelo de hospedagem do lado do cliente](#) podem aceitar os seguintes valores de configuração do host como argumentos de linha de comando no tempo de execução no ambiente de desenvolvimento.

### Raiz do conteúdo

O argumento `--contentroot` define o caminho absoluto para o diretório que contém os arquivos de conteúdo do aplicativo.

- Passe o argumento ao executar o aplicativo localmente em um prompt de comando. No diretório do aplicativo, execute:

```
dotnet run --contentroot=/<content-root>
```

- Adicione uma entrada ao arquivo `launchSettings.json` do aplicativo no perfil do **IIS Express**. Essa configuração é obtida ao executar o aplicativo com o Depurador do Visual Studio e ao executar o aplicativo em um prompt de comando com `dotnet run`.

```
"commandLineArgs": "--contentroot=/<content-root>"
```

- No Visual Studio, especifique o argumento em **Propriedades > Depurar > Argumentos de aplicativo**. A configuração do argumento na página de propriedades do Visual Studio adiciona o argumento ao arquivo *launchSettings.json*.

```
--contentroot=/<content-root>
```

## Caminho base

O argumento `--pathbase` define o caminho base do aplicativo para um aplicativo executado localmente com um caminho virtual não raiz (a tag `href` `<base>` é definida como um caminho diferente de `/` para preparo e produção). Para obter mais informações, confira a seção [Caminho base do aplicativo](#).

### IMPORTANT

Ao contrário do caminho fornecido ao `href` da tag `<base>`, não inclua uma barra à direita (`/`) ao passar o valor do argumento `--pathbase`. Se o caminho base do aplicativo for fornecido na tag `<base>` como `<base href="/CoolApp/" />` (inclui uma barra à direita), passe o valor do argumento de linha de comando como `--pathbase=/CoolApp` (nenhuma barra à direita).

- Passe o argumento ao executar o aplicativo localmente em um prompt de comando. No diretório do aplicativo, execute:

```
dotnet run --pathbase=/<virtual-path>
```

- Adicione uma entrada ao arquivo *launchSettings.json* do aplicativo no perfil do **IIS Express**. Essa configuração é obtida ao executar o aplicativo com o Depurador do Visual Studio e ao executar o aplicativo em um prompt de comando com `dotnet run`.

```
"commandLineArgs": "--pathbase=/<virtual-path>"
```

- No Visual Studio, especifique o argumento em **Propriedades > Depurar > Argumentos de aplicativo**. A configuração do argumento na página de propriedades do Visual Studio adiciona o argumento ao arquivo *launchSettings.json*.

```
--pathbase=/<virtual-path>
```

## URLs

O argumento `--urls` indica os endereços IP ou os endereços de host com portas e protocolos para escutar solicitações.

- Passe o argumento ao executar o aplicativo localmente em um prompt de comando. No diretório do aplicativo, execute:

```
dotnet run --urls=http://127.0.0.1:0
```

- Adicione uma entrada ao arquivo *launchSettings.json* do aplicativo no perfil do **IIS Express**. Essa configuração é obtida ao executar o aplicativo com o Depurador do Visual Studio e ao executar o aplicativo em um prompt de comando com `dotnet run`.

```
"commandLineArgs": "--urls=http://127.0.0.1:0"
```

- No Visual Studio, especifique o argumento em **Propriedades > Depurar > Argumentos de aplicativo**. A configuração do argumento na página de propriedades do Visual Studio adiciona o argumento ao arquivo `launchSettings.json`.

```
--urls=http://127.0.0.1:0
```

## Implantar um aplicativo do Blazor do lado do cliente

Com o [modelo de hospedagem do lado do cliente](#):

- O aplicativo do Blazor, suas dependências e o tempo de execução do .NET são baixados no navegador.
- O aplicativo é executado diretamente no thread da interface do usuário do navegador. Há suporte para todas as estratégias a seguir:
  - O aplicativo do Blazor é atendido por um aplicativo ASP.NET Core. Abordado na seção [Implantação hospedada do Blazor do lado do cliente com o ASP.NET Core](#).
  - O aplicativo do Blazor é colocado em um serviço ou um servidor Web de hospedagem estático, em que o .NET não é usado para atender ao aplicativo do Blazor. Abordado na seção [Implantação autônoma do Blazor do lado do cliente](#).

### Configurar o vinculador

O Blazor executa a vinculação de IL (linguagem intermediária) em cada build para remover a IL desnecessária dos assemblies de saída. Você pode controlar a vinculação de assembly no build. Para obter mais informações, consulte [Configurar o Vinculador para o Blazor](#).

### Reescrever as URLs para obter o roteamento correto

O roteamento de solicitações para componentes de página em um aplicativo do lado do cliente não é tão simples quanto o roteamento de solicitações para um aplicativo hospedado do lado do servidor. Considere um aplicativo do lado do cliente com duas páginas:

- **Main.cshtml** – É carregado na raiz do aplicativo e contém um link para a página Sobre (`href="About"`).
- **About.cshtml** – Página Sobre.

Quando o documento padrão do aplicativo é solicitado usando a barra de endereços do navegador (por exemplo, `https://www.contoso.com/`):

1. O navegador faz uma solicitação.
2. A página padrão é retornada, que é geralmente `index.html`.
3. A `index.html` inicia o aplicativo.
4. O roteador do Blazor é carregado e a página Principal do Razor (`Main.cshtml`) é exibida.

Na página Principal, é possível carregar a página Sobre selecionando o link para ela. A seleção do link para a página Sobre funciona no cliente porque o roteador do Blazor impede que o navegador faça uma solicitação na Internet para `www.contoso.com` de `About` e atende à própria página Sobre. Todas as solicitações de páginas internas *no aplicativo do lado do cliente* funcionam da mesma maneira: Não são disparadas solicitações baseadas em navegador para os recursos hospedados no servidor na Internet. O roteador trata das solicitações internamente.

Se uma solicitação for feita usando a barra de endereços do navegador para `www.contoso.com/About`, a solicitação falhará. Esse recurso não existe no host do aplicativo na Internet, portanto, uma resposta *404 Não Encontrado* é retornada.

Como os navegadores fazem solicitações aos hosts baseados na Internet de páginas do lado do cliente, os

servidores Web e os serviços de hospedagem precisam reescrever todas as solicitações de recursos que não estão fisicamente no servidor para a página `index.html`. Quando a `index.html` for retornada, o roteador do lado do cliente do aplicativo assumirá o controle e responderá com o recurso correto.

## Caminho base do aplicativo

O caminho base do aplicativo é o caminho raiz do aplicativo virtual no servidor. Por exemplo, um aplicativo que reside no servidor Contoso em uma pasta virtual em `/CoolApp/` é acessado em `https://www.contoso.com/CoolApp` e tem o caminho base virtual `/CoolApp/`. Quando o caminho base do aplicativo é definido como `CoolApp/`, o aplicativo fica ciente de onde ele reside virtualmente no servidor. O aplicativo pode usar o caminho base para construir URLs relativas à raiz do aplicativo de um componente que não esteja no diretório raiz. Isso permite que os componentes que existem em diferentes níveis da estrutura de diretório criem links para outros recursos em locais em todo o aplicativo. O caminho base do aplicativo também é usado para interceptar cliques em hiperlink em que o destino `href` do link está dentro do espaço do URI do caminho base do aplicativo. O roteador do Blazor manipula a navegação interna.

Em muitos cenários de hospedagem, o caminho virtual do servidor para o aplicativo é a raiz do aplicativo. Nesses casos, o caminho base do aplicativo é uma barra invertida (`<base href="/" />`), que é a configuração padrão de um aplicativo. Em outros cenários de hospedagem, como Páginas do GitHub e diretórios virtuais ou subaplicativos do IIS, o caminho base do aplicativo precisa ser definido como o caminho virtual do servidor para o aplicativo. Para definir o caminho base do aplicativo, adicione ou atualize a tag `<base>` na `index.html` encontrada nos elementos da tag `<head>`. Defina o valor de atributo `href` como `<virtual-path>/` (a barra à direita é necessária), em que `<virtual-path>/` é o caminho raiz do aplicativo virtual completo no servidor do aplicativo. No exemplo anterior, o caminho virtual é definido como `CoolApp/`: `<base href="CoolApp/" />`.

No caso de um aplicativo com um caminho virtual não raiz configurado (por exemplo, `<base href="CoolApp/" />`), o aplicativo não consegue localizar seus recursos *quando é executado localmente*. Para superar esse problema durante o desenvolvimento e os testes locais, você pode fornecer um argumento *base de caminho* que corresponde ao valor de `href` da tag `<base>` no tempo de execução.

Para passar o argumento base de caminho com o caminho raiz (`/`) ao executar o aplicativo localmente, execute o seguinte comando no diretório do aplicativo:

```
dotnet run --pathbase=/CoolApp
```

O aplicativo responde localmente em `http://localhost:port/CoolApp`.

Para obter mais informações, confira a seção de [valor de configuração do host base de caminho](#).

## IMPORTANT

Se um aplicativo usa o [modelo de hospedagem do lado do cliente](#) (com base no modelo de projeto **Blazor**) e é hospedado como um subaplicativo do IIS em um aplicativo do ASP.NET Core, é importante desabilitar o manipulador de módulo do ASP.NET Core herdado ou verificar se a seção `<handlers>` do aplicativo raiz (pai) no arquivo `web.config` não é herdada pelo subaplicativo.

Remova o manipulador do arquivo `web.config` publicado do aplicativo adicionando uma seção `<handlers>` ao arquivo:

```
<handlers>
  <remove name="aspNetCore" />
</handlers>
```

Como alternativa, desabilite a herança da seção `<system.webServer>` do aplicativo raiz (pai) usando um elemento

```
<location> com inheritInChildApplications definido como false:
```

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <location path=". " inheritInChildApplications="false">
    <system.webServer>
      <handlers>
        <add name="aspNetCore" ... />
      </handlers>
      <aspNetCore ... />
    </system.webServer>
  </location>
</configuration>
```

A ação de remover o manipulador ou desabilitar a herança é realizada além da ação da configurar o caminho base do aplicativo, conforme descrito nesta seção. Defina o caminho base do aplicativo no arquivo `index.html` do aplicativo do alias do IIS usado ao configurar o subaplicativo no IIS.

## Implantação hospedada do Blazor do lado do cliente com o ASP.NET Core

Uma *implantação hospedada* atende ao aplicativo do Blazor do lado do cliente para navegadores de um [aplicativo do ASP.NET Core](#) que é executado em um servidor.

O aplicativo do Blazor é incluído com o aplicativo do ASP.NET Core na saída publicada para que ambos sejam implantados juntos. É necessário um servidor Web capaz de hospedar um aplicativo do ASP.NET Core. Para uma implantação hospedada, o Visual Studio inclui o modelo de projeto **Blazor (hospedado no ASP.NET Core)** (modelo `blazorhosted` ao usar o comando `dotnet new`).

Para obter mais informações sobre a implantação e a hospedagem de aplicativo do ASP.NET Core, confira [Hospedar e implantar o ASP.NET Core](#).

Para obter informações de como implantar o Serviço de Aplicativo do Azure, confira os tópicos a seguir:

### [Publicar um aplicativo ASP.NET Core no Azure com o Visual Studio](#)

Aprenda como publicar um aplicativo ASP.NET Core no Serviço de Aplicativo do Azure usando o Visual Studio.

## Implantação autônoma do Blazor do lado do cliente

Uma *implantação autônoma* atende ao aplicativo do Blazor do lado do cliente como um conjunto de arquivos estáticos que são solicitados diretamente pelos clientes. Não é usado é um servidor Web para atender ao aplicativo do Blazor.

### [Hospedagem autônoma do Blazor do lado do cliente com o IIS](#)

O IIS é um servidor de arquivos estático com capacidade para aplicativos do Blazor. Para configurar o IIS para hospedar o Blazor, confira [Build a Static Website on IIS](#) (Criar um site estático no IIS).

Os ativos publicados são criados na pasta `\bin\Release\<target-framework>\publish`. Hospede o conteúdo da

pasta *publish* no servidor Web ou no serviço de hospedagem.

## web.config

Quando um projeto Blazor é publicado, um arquivo *web.config* é criado com a seguinte configuração do IIS:

- Os tipos MIME são definidos para as seguintes extensões de arquivo:
  - \*.dll: application/octet-stream
  - \*json: application/json
  - \*.wasm: application/wasm
  - \*.woff: application/font-woff
  - \*.woff2: application/font-woff
- A compactação HTTP está habilitada para os seguintes tipos MIME:
  - application/octet-stream
  - application/wasm
- As regras do Módulo de Reescrita de URL são estabelecidas:
  - Atender ao subdiretório em que residem os ativos estáticos do aplicativo (<*assembly\_name*>|*dist*|<*path\_requested*>).
  - Criar o roteamento de fallback do SPA, de modo que as solicitações de ativos que não sejam arquivos sejam redirecionadas ao documento padrão do aplicativo na pasta de ativos estáticos dele (<*assembly\_name*>|*dist*|index.html).

## Instalar o Módulo de Reescrita de URL

O [Módulo de Reescrita de URL](#) é necessário para reescrever URLs. O módulo não está instalado por padrão e não está disponível para instalação como um recurso do serviço de função do servidor Web (IIS). O módulo precisa ser baixado do site do IIS. Use o Web Platform Installer para instalar o módulo:

1. Localmente, navegue até a [página de downloads do Módulo de Reescrita de URL](#). Para obter a versão em inglês, selecione **WebPI** para baixar o instalador do WebPI. Para outros idiomas, selecione a arquitetura apropriada para o servidor (x86/x64) para baixar o instalador.
2. Copie o instalador para o servidor. Execute o instalador. Selecione o botão **Instalar** e aceite os termos de licença. Uma reinicialização do servidor não será necessária após a conclusão da instalação.

## Configurar o site

Defina o **Caminho físico** do site como a pasta do aplicativo. A pasta contém:

- O arquivo *web.config* que o IIS usa para configurar o site, incluindo as regras de redirecionamento e os tipos de conteúdo do arquivo necessários.
- A pasta de ativos estática do aplicativo.

## Solução de problemas

Se um *500 Erro Interno do Servidor* for recebido e o Gerenciador do IIS gerar erros ao tentar acessar a configuração do site, confirme se o Módulo de Reescrita de URL está instalado. Quando o módulo não estiver instalado, o arquivo *web.config* não poderá ser analisado pelo IIS. Isso impede que o Gerenciador do IIS carregue a configuração do site e que o site atenda aos arquivos estáticos do Blazor.

Para obter mais informações de como solucionar problemas de implantações no IIS, confira [Solucionar problemas do ASP.NET Core no IIS](#).

## Hospedagem autônoma do Blazor do lado do cliente com o Nginx

O arquivo *nginx.conf* a seguir é simplificado para mostrar como configurar o Nginx para enviar o arquivo *Index.html* sempre que ele não puder encontrar um arquivo correspondente no disco.

```
events { }
http {
    server {
        listen 80;

        location / {
            root /usr/share/nginx/html;
            try_files $uri $uri/ /Index.html =404;
        }
    }
}
```

Para obter mais informações sobre a configuração do servidor Web Nginx de produção, confira [Creating NGINX Plus and NGINX Configuration Files](#) (Criando arquivos de configuração do NGINX Plus e do NGINX).

#### Hospedagem autônoma do Blazor do lado do cliente no Docker

Para hospedar o Blazor no Docker usando o Nginx, configure o Dockerfile para usar a imagem do Nginx baseada no Alpine. Atualize o Dockerfile para copiar o arquivo `nginx.config` no contêiner.

Adicione uma linha ao Dockerfile, conforme é mostrado no exemplo a seguir:

```
FROM nginx:alpine
COPY ./bin/Release/netstandard2.0/publish /usr/share/nginx/html/
COPY nginx.conf /etc/nginx/nginx.conf
```

#### Hospedagem autônoma do Blazor do lado do cliente com as Páginas do GitHub

Para lidar com as reescritas de URL, adicione um arquivo `404.html` com um script que manipule o redirecionamento de solicitação para a página `index.html`. Para obter uma implementação de exemplo fornecida pela comunidade, confira [Single Page Apps for GitHub Pages](#) (Aplicativos de página única das Páginas do GitHub) ([rafrex/spa-github-pages no GitHub](#)). Um exemplo usando a abordagem da comunidade pode ser visto em [blazor-demo/blazor-demo.github.io no GitHub](#) (site dinâmico).

Ao usar um site de projeto em vez de um site de empresa, adicione ou atualize a tag `<base>` no `index.html`. Defina o valor do atributo `<repository-name>/` do `href`, em que `<repository-name>/` é o nome do repositório do GitHub.

## Implantar um aplicativo dos Componentes Razor do lado do servidor

Com o [modelo de hospedagem do lado do servidor](#), os Componentes Razor são executados no servidor de dentro de um aplicativo ASP.NET Core. As atualizações da interface do usuário, a manipulação de eventos e as chamadas de JavaScript são realizadas por uma conexão SignalR.

O aplicativo é incluído com o aplicativo do ASP.NET Core na saída publicada para que os dois aplicativos sejam implantados juntos. É necessário um servidor Web capaz de hospedar um aplicativo do ASP.NET Core. Para uma implantação do lado do servidor, o Visual Studio inclui o modelo de projeto **Blazor (do lado do servidor no ASP.NET Core)** (modelo `blazorserver` ao usar o comando `dotnet new`).

Quando o aplicativo do ASP.NET Core é publicado, o aplicativo dos Componentes Razor é incluído na saída publicada para que ambos possam ser implantados juntos. Para obter mais informações sobre a implantação e a hospedagem de aplicativo do ASP.NET Core, confira [Hospedar e implantar o ASP.NET Core](#).

Para obter informações de como implantar o Serviço de Aplicativo do Azure, confira os tópicos a seguir:

#### Publicar um aplicativo ASP.NET Core no Azure com o Visual Studio

Aprenda como publicar um aplicativo ASP.NET Core no Serviço de Aplicativo do Azure usando o Visual Studio.

# Configurar o Vinculador para o Blazor

06/02/2019 • 2 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

## NOTE

O Razor Components do ASP.NET Core é compatível com ASP.NET Core 3.0 ou posterior.

Blazor é uma estrutura da Web sem suporte e experimental que não deve ser usada para cargas de trabalho de produção no momento.

O Blazor executa a vinculação de [IL \(linguagem intermediária\)](#) durante cada build do Modo de Versão para remover IL desnecessária dos assemblies de saída.

Você pode controlar a vinculação do assembly com uma das seguintes abordagens:

- Desabilite a vinculação globalmente com uma propriedade MSBuild.
- Controle a vinculação por assembly usando um arquivo de configuração.

## Desabilitar a vinculação com uma propriedade MSBuild

A vinculação é habilitada por padrão no Modo de Versão quando um aplicativo é criado, o que inclui publicação.

Para desabilitar a vinculação para todos os assemblies, defina a propriedade `<BlazorLinkOnBuild>` MSBuild como `false` no arquivo de projeto:

```
<PropertyGroup>
  <BlazorLinkOnBuild>false</BlazorLinkOnBuild>
</PropertyGroup>
```

## Controlar a vinculação com um arquivo de configuração

A vinculação pode ser controlada por assembly fornecendo um arquivo de configuração XML e especificando o arquivo como um item MSBuild no arquivo de projeto.

Veja a seguir um exemplo de arquivo de configuração (*Linker.xml*):

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
  This file specifies which parts of the BCL or Blazor packages must not be
  stripped by the IL Linker even if they aren't referenced by user code.
-->
<linker>
  <assembly fullname="mscorlib">
    <!--
      Preserve the methods in WasmRuntime because its methods are called by
      JavaScript client-side code to implement timers.
      Fixes: https://github.com/aspnet/Blazor/issues/239
    -->
    <type fullname="System.Threading.WasmRuntime" />
  </assembly>
  <assembly fullname="System.Core">
    <!--
      System.Linq.Expressions* is required by Json.NET and any
      expression.Compile caller. The assembly isn't stripped.
    -->
    <type fullname="System.Linq.Expressions*" />
  </assembly>
  <!--
    In this example, the app's entry point assembly is listed. The assembly
    isn't stripped by the IL Linker.
  -->
  <assembly fullname="MyCoolBlazorApp" />
</linker>
```

Para saber mais sobre o formato de arquivo para o arquivo de configuração, veja [Vinculador de IL: sintaxe do descriptor de xml](#).

Especifique o arquivo de configuração no arquivo de projeto com o item `BlazorLinkerDescriptor`:

```
<ItemGroup>
  <BlazorLinkerDescriptor Include="Linker.xml" />
</ItemGroup>
```

# Usar o modelo de projeto Angular com o ASP.NET Core

15/10/2018 • 20 minutes to read • [Edit Online](#)

## NOTE

Esta documentação não é sobre o modelo de projeto do Angular incluído no ASP.NET Core 2.0. Ela é sobre o modelo Angular mais recente, para o qual você pode atualizar manualmente. O modelo está incluído no ASP.NET Core 2.1 por padrão.

O modelo de projeto do Angular atualizado fornece um ponto inicial conveniente para aplicativos do ASP.NET Core usando o Angular e a CLI do Angular para implementar uma IU (interface do usuário) avançada do lado do cliente.

O modelo é equivalente à criação de um projeto do ASP.NET Core para atuar como um back-end de API e um projeto de CLI do Angular para atuar como uma interface do usuário. O modelo oferece a conveniência de hospedagem de ambos os tipos de projeto em um projeto de aplicativo único. Consequentemente, o projeto de aplicativo pode ser criado e publicado como uma única unidade.

## Criar um novo aplicativo

Se usar o ASP.NET Core 2.0, verifique se você [instalou o modelo de projeto do Angular atualizado](#).

Se você tiver o ASP.NET Core 2.1 instalado, não será necessário instalar o modelo de projeto Angular.

Crie um novo projeto de um prompt de comando usando o comando `dotnet new angular` em um diretório vazio. Por exemplo, os seguintes comandos criam o aplicativo em um diretório `my-new-app` e mudam para esse diretório:

```
dotnet new angular -o my-new-app  
cd my-new-app
```

Execute o aplicativo do Visual Studio ou da CLI do .NET Core:

- [Visual Studio](#)
- [CLI do .NET Core](#)

Abra o arquivo `.csproj` gerado e execute o aplicativo normalmente de lá.

O processo de build restaura dependências npm na primeira execução, o que pode levar vários minutos. Builds subsequentes são muito mais rápidos.

O modelo de projeto cria um aplicativo ASP.NET Core e um aplicativo do Angular. O aplicativo ASP.NET Core destina-se a ser usado para acesso a dados, autorização e outras questões do lado do servidor. O aplicativo do Angular, que reside no subdiretório `ClientApp`, destina-se a ser usado para todas as questões de interface do usuário.

## Adicione páginas, imagens, estilos, módulos, etc.

O diretório `ClientApp` contém um aplicativo padrão da CLI do Angular. Veja a [documentação oficial do Angular](#)

para obter mais informações.

Há pequenas diferenças entre o aplicativo do Angular criado por este modelo e um criado pela CLI do Angular propriamente dita (por meio de `ng new`); no entanto, os recursos do aplicativo são inalterados. O aplicativo criado pelo modelo contém um layout com base em [Bootstrap](#) e um exemplo básico de roteamento.

## Executar comandos ng

Em um prompt de comando, mude para o subdiretório *ClientApp*:

```
cd ClientApp
```

Se você tiver a ferramenta `ng` instalada globalmente, você poderá executar qualquer um dos seus comandos. Por exemplo, você poderá executar `ng lint`, `ng test` ou qualquer um dos outros [comandos da CLI do Angular](#). No entanto, não é necessário executar `ng serve`, porque o seu aplicativo ASP.NET Core dá conta de servir tanto a parte do lado do servidor quanto a do lado do cliente do seu aplicativo. Internamente, ele usa `ng serve` em desenvolvimento.

Se você não tiver a ferramenta `ng` instalada, execute `npm run ng` em vez dela. Por exemplo, você pode executar `npm run ng lint` ou `npm run ng test`.

## Instalar pacotes npm

Para instalar pacotes npm de terceiros, use um prompt de comando no subdiretório *ClientApp*. Por exemplo:

```
cd ClientApp
npm install --save <package_name>
```

## Publicar e implantar

No desenvolvimento, o aplicativo é executado de um modo otimizado para conveniência do desenvolvedor. Por exemplo, pacotes JavaScript incluem mapas de origem (de modo que durante a depuração, você pode ver o código TypeScript original). O aplicativo observa alterações em arquivos TypeScript, HTML e CSS no disco e recompila e recarrega automaticamente quando as detecta.

Em produção, atende a uma versão de seu aplicativo que é otimizada para desempenho. Isso é configurado para ocorrer automaticamente. Quando você publica, a configuração de build emite um build minificado em compilação AoT (Ahead Of Time) do código do lado do cliente. Diferentemente do build de desenvolvimento, o build de produção não requer que o Node.js esteja instalado no servidor (a menos que você tenha habilitado a [pré-renderização do lado do servidor](#)).

Você pode usar os [métodos padrão de implantação e hospedagem do ASP.NET Core](#).

## Executar "ng serve" independentemente

O projeto está configurado para iniciar sua própria instância do servidor da CLI do Angular em segundo plano quando o aplicativo ASP.NET Core é iniciado no modo de desenvolvimento. Isso é conveniente, porque você não precisa executar um servidor separado manualmente.

Há uma desvantagem nessa configuração padrão. Cada vez que você modificar seu código C# e o aplicativo ASP.NET Core precisar ser reiniciado, o servidor da CLI do Angular será reiniciado também. São necessários aproximadamente 10 segundos para iniciar um backup. Se você estiver fazendo edições frequentes de código C# e não quiser esperar a CLI do Angular reiniciar, execute o servidor da CLI do Angular externamente, independentemente do processo do ASP.NET Core. Para fazer isso:

1. Em um prompt de comando, vá para o subdiretório *ClientApp* e inicie o Development Server da CLI do Angular:

```
cd ClientApp  
npm start
```

#### IMPORTANT

Use `npm start` para iniciar o Development Server da CLI do Angular, não `ng serve`, de modo que a configuração no `package.json` seja respeitada. Para passar parâmetros adicionais para o servidor da CLI do Angular, adicione-os à linha `scripts` relevante em seu arquivo `package.json`.

2. Modifique o aplicativo ASP.NET Core para, em vez de iniciar uma instância da CLI do Angular própria, usar a externa. Na classe *Startup*, substitua a invocação `spa.UseAngularCliServer` pelo seguinte:

```
spa.UseProxyToSpaDevelopmentServer("http://localhost:4200");
```

Quando você iniciar seu aplicativo ASP.NET Core, ele não inicializará um servidor da CLI do Angular. Em vez disso, a instância que você iniciou manualmente é usada. Isso permite a ele iniciar e reiniciar mais rapidamente. Ele não está mais aguardando a CLI do Angular recompilar o aplicativo cliente a cada vez.

## Renderização do lado do servidor

Como um recurso de desempenho, você pode escolher pré-renderizar seu aplicativo do Angular no servidor, bem como executá-lo no cliente. Isso significa que os navegadores recebem uma marcação HTML que representa a interface do usuário inicial do aplicativo, para exibirem mesmo antes de baixar e executar os pacotes de JavaScript. A maior parte da implementação disso vem de um recurso do Angular chamado [Angular Universal](#).

#### TIP

Habilitar a SSR (renderização do lado do servidor) apresenta uma série de complicações adicionais, durante o desenvolvimento e a implantação. Leia [desvantagens da SSR](#) para determinar se a SSR é uma boa opção para as suas necessidades.

Para habilitar a SSR, você precisa fazer um número de adições ao seu projeto.

Na classe *Startup*, *após* a linha que configura `spa.Options.SourcePath` e *antes* da chamada para `UseAngularCliServer` OU `UseProxyToSpaDevelopmentServer`, adicione o seguinte:

```

app.UseSpa(spa =>
{
    spa.Options.SourcePath = "ClientApp";

    spa.UseSpaPrerendering(options =>
    {
        options.BootModulePath = $"{spa.Options.SourcePath}/dist-server/main.bundle.js";
        options.BootModuleBuilder = env.IsDevelopment()
            ? new AngularCliBuilder(npmScript: "build:ssr")
            : null;
        options.ExcludeUrls = new[] { "/sockjs-node" };
    });

    if (env.IsDevelopment())
    {
        spa.UseAngularCliServer(npmScript: "start");
    }
});

```

No modo de desenvolvimento, esse código tentará criar o pacote SSR executando o script `build:ssr`, que é definido em `ClientApp\package.json`. Isso cria um aplicativo do Angular chamado `ssr`, que ainda não está definido.

No final da matriz `apps` em `ClientApp\angular-cli.json`, defina um aplicativo adicional com o nome `ssr`. Use as seguintes opções:

```
{
    "name": "ssr",
    "root": "src",
    "outDir": "dist-server",
    "assets": [
        "assets"
    ],
    "main": "main.server.ts",
    "tsconfig": "tsconfig.server.json",
    "prefix": "app",
    "scripts": [],
    "environmentSource": "environments/environment.ts",
    "environments": {
        "dev": "environments/environment.ts",
        "prod": "environments/environment.prod.ts"
    },
    "platform": "server"
}
```

Essa nova configuração de aplicativo habilitada para SSR requer dois arquivos adicionais: `tsconfig.server.json` e `main.server.ts`. O arquivo `tsconfig.server.json` especifica opções de compilação de TypeScript. O arquivo `main.server.ts` serve como o ponto de entrada de código durante a SSR.

Adicione um novo arquivo chamado `tsconfig.server.json` dentro de `ClientApp/src` (junto com o `tsconfig.app.json` existente), contendo o seguinte:

```
{  
  "extends": "../tsconfig.json",  
  "compilerOptions": {  
    "baseUrl": "./",  
    "module": "commonjs"  
  },  
  "angularCompilerOptions": {  
    "entryModule": "app/app.server.module#AppServerModule"  
  }  
}
```

Esse arquivo configura o compilador AoT do Angular para procurar por um módulo chamado `app.server.module`. Adicione isso ao criar um novo arquivo em `ClientApp/src/app/app.server.module.ts` (junto com o `app.module.ts` existente) que contém o seguinte:

```
import { NgModule } from '@angular/core';  
import { ServerModule } from '@angular/platform-server';  
import { ModuleMapLoaderModule } from '@nguniversal/module-map-ngfactory-loader';  
import { AppComponent } from './app.component';  
import { AppModule } from './app.module';  
  
@NgModule({  
  imports: [AppModule, ServerModule, ModuleMapLoaderModule],  
  bootstrap: [AppComponent]  
})  
export class AppServerModule { }
```

Esse módulo herda de seu `app.module` do lado do cliente e define quais módulos extra do Angular estão disponíveis durante a SSR.

Lembre-se de que a nova entrada `ssr` em `.angular-cli.json` referenciou de um arquivo de ponto de entrada chamado `main.server.ts`. Você ainda não adicionou esse arquivo e agora é hora de fazê-lo. Crie um novo arquivo em `ClientApp/src/main.server.ts` (junto com o `main.ts` existente), contendo o seguinte:

```

import 'zone.js/dist/zone-node';
import 'reflect-metadata';
import { renderModule, renderModuleFactory } from '@angular/platform-server';
import { APP_BASE_HREF } from '@angular/common';
import { enableProdMode } from '@angular/core';
import { provideModuleMap } from '@nguniversal/module-map-ngfactory-loader';
import { createServerRenderer } from 'aspnet-prerendering';
export { AppServerModule } from './app/app.server.module';

enableProdMode();

export default createServerRenderer(params => {
  const { AppServerModule, AppServerModuleNgFactory, LAZY_MODULE_MAP } = (module as any).exports;

  const options = {
    document: params.data.originalHtml,
    url: params.url,
    extraProviders: [
      provideModuleMap(LAZY_MODULE_MAP),
      { provide: APP_BASE_HREF, useValue: params.baseUrl },
      { provide: 'BASE_URL', useValue: params.origin + params.baseUrl }
    ]
  };
}

const renderPromise = AppServerModuleNgFactory
  ? /* AoT */ renderModuleFactory(AppServerModuleNgFactory, options)
  : /* dev */ renderModule(AppServerModule, options);

return renderPromise.then(html => ({ html }));
});

```

O código do arquivo é o que o ASP.NET Core executa para cada solicitação quando ele executa o middleware `UseSpaPrerendering` que você adicionou à classe `Startup`. Ele trata do recebimento de `params` do código .NET (por exemplo, a URL que está sendo solicitada) e de fazer chamadas a APIs de SSR do Angular para obter o HTML resultante.

A rigor, isso é suficiente para habilitar a SSR no modo de desenvolvimento. É essencial para fazer uma alteração final para que seu aplicativo funcione corretamente quando publicado. No arquivo `.csproj` principal do seu aplicativo, defina o valor da propriedade `BuildServerSideRenderer` para `true`:

```

<!-- Set this to true if you enable server-side prerendering -->
<BuildServerSideRenderer>true</BuildServerSideRenderer>

```

Isso configura o processo de build para executar `build:ssr` durante a publicação e implantar os arquivos SSR no servidor. Se você não habilitar isso, a SSR falhará em produção.

Quando o aplicativo é executado no modo de desenvolvimento ou de produção, o código do Angular é previamente renderizado como HTML no servidor. O código do lado do cliente é executado normalmente.

### **Passar dados de código .NET para código do TypeScript**

Durante a SSR, convém passar dados por solicitação, do aplicativo ASP.NET Core para o aplicativo do Angular. Por exemplo, você pode transmitir informações de cookie ou algo lido de um banco de dados. Para fazer isso, edite a classe `Startup`. No retorno de chamada para `UseSpaPrerendering`, defina um valor para `options.SupplyData` como o seguinte:

```
options.SupplyData = (context, data) =>
{
  // Creates a new value called isHttpsRequest that's passed to TypeScript code
  data["isHttpsRequest"] = context.Request.IsHttps;
};
```

O retorno de chamada `SupplyData` permite que você passe dados serializáveis por JSON arbitrários e por solicitação (por exemplo, cadeias de caracteres, booleanos ou números). O código de `main.server.ts` recebe isso como `params.data`. Por exemplo, o exemplo de código anterior passa um valor booleano como `params.data.isHttpsRequest` para o retorno de chamada `createServerRenderer`. Você pode passar isso para outras partes do seu aplicativo de alguma forma compatível com o Angular. Por exemplo, veja como `main.server.ts` passa o valor `BASE_URL` para qualquer componente cujo construtor é declarado para recebê-lo.

## Desvantagens da SSR

Nem todos os aplicativos se beneficiam da SSR. O principal benefício é o desempenho percebido. Os visitantes que chegam ao aplicativo por uma conexão de rede lenta ou em dispositivos móveis lentos veem a interface do usuário inicial rapidamente, mesmo que leve algum tempo para buscar ou para analisar os pacotes de JavaScript. No entanto, muitos SPAs são usados principalmente em redes de empresa internas e rápidas, em computadores rápidos nos quais o aplicativo aparece quase instantaneamente.

Ao mesmo tempo, há desvantagens significativas em habilitar a SSR. Ele adiciona complexidade ao seu processo de desenvolvimento. O código deve ser executado em dois ambientes diferentes: no lado do cliente e no do servidor (em um ambiente Node.js invocado do ASP.NET Core). Estes são alguns pontos a considerar:

- A SSR requer uma instalação de Node.js nos servidores de produção. Isso ocorre automaticamente para alguns cenários de implantação como Serviços de Aplicativos do Azure, mas não para outros como o Azure Service Fabric.
- Habilitar o sinalizador de build `BuildServerSideRenderer` faz com que o diretório `node_modules` seja publicado. Esta pasta contém mais de 20.000 arquivos, o que aumenta o tempo de implantação.
- Para executar o código em um ambiente Node.js, ele não pode depender da existência de APIs de JavaScript específicas a um navegador, tais como `window` ou `localStorage`. Se seu código (ou alguma biblioteca de terceiros à qual você faz referência) tentar usar essas APIs, você obterá um erro durante a SSR. Por exemplo, não use jQuery porque faz referência a APIs específicas a um navegador em vários locais. Para evitar erros, você deve evitar a SSR ou então evitar APIs ou bibliotecas específicas a um navegador. Você pode encapsular todas as chamadas para essas APIs em verificações para garantir que elas não sejam invocadas durante a SSR. Por exemplo, use uma verificação como a seguinte no código JavaScript ou TypeScript:

```
if (typeof window !== 'undefined') {
  // Call browser-specific APIs here
}
```

# Usar o modelo de projeto do React com o ASP.NET Core

02/02/2019 • 8 minutes to read • [Edit Online](#)

## NOTE

Esta documentação não é sobre o modelo de projeto do React incluído no ASP.NET Core 2.0. Ela é sobre o modelo React mais recente, para o qual você pode atualizar manualmente. O modelo está incluído no ASP.NET Core 2.1 por padrão.

O modelo de projeto do React atualizado fornece um ponto inicial conveniente para aplicativos do ASP.NET Core usando convenções do React e de CRA ([criar-aplicativo-do-React](#)) para implementar uma IU (interface do usuário) avançada do lado do cliente.

O modelo é equivalente à criação de dois projetos: um projeto do ASP.NET Core, para atuar como um back-end de API, e um projeto do React CRA padrão, para atuar como uma interface do usuário, mas com a praticidade de hospedar ambos em um único projeto de aplicativo que pode ser criado e publicado como uma única unidade.

## Criar um novo aplicativo

Se usar o ASP.NET Core 2.0, verifique se você [instalou o modelo de projeto do React atualizado](#).

Se você tiver o ASP.NET Core 2.1 instalado, não será necessário instalar o modelo de projeto React.

Crie um novo projeto de um prompt de comando usando o comando `dotnet new react` em um diretório vazio. Por exemplo, os seguintes comandos criam o aplicativo em um diretório `my-new-app` e mudam para esse diretório:

```
dotnet new react -o my-new-app  
cd my-new-app
```

Execute o aplicativo do Visual Studio ou da CLI do .NET Core:

- [Visual Studio](#)
- [CLI do .NET Core](#)

Abra o arquivo `.csproj` gerado e execute o aplicativo normalmente de lá.

O processo de build restaura dependências npm na primeira execução, o que pode levar vários minutos. Builds subsequentes são muito mais rápidos.

O modelo de projeto cria um aplicativo ASP.NET Core e um aplicativo do React. O aplicativo ASP.NET Core destina-se a ser usado para acesso a dados, autorização e outras questões do lado do servidor. O aplicativo do React, que reside no subdiretório `ClientApp`, destina-se a ser usado para todas as questões de interface do usuário.

## Adicione páginas, imagens, estilos, módulos, etc.

O diretório `ClientApp` é um aplicativo do React CRA padrão. Veja a [documentação oficial do CRA](#) para obter mais informações.

Há pequenas diferenças entre o aplicativo do React criado por este modelo e um criado pelo CRA propriamente dito; no entanto, os recursos do aplicativo são inalterados. O aplicativo criado pelo modelo contém um layout com

base em [Bootstrap](#) e um exemplo básico de roteamento.

## Instalar pacotes npm

Para instalar pacotes npm de terceiros, use um prompt de comando no subdiretório *ClientApp*. Por exemplo:

```
cd ClientApp  
npm install --save <package_name>
```

## Publicar e implantar

No desenvolvimento, o aplicativo é executado de um modo otimizado para conveniência do desenvolvedor. Por exemplo, pacotes JavaScript incluem mapas de origem (de modo que durante a depuração, você pode ver o código-fonte original). O aplicativo observa alterações em arquivos JavaScript, HTML e CSS no disco e recompila e recarrega automaticamente quando as detecta.

Em produção, atende a uma versão de seu aplicativo que é otimizada para desempenho. Isso é configurado para ocorrer automaticamente. Quando você publica, a configuração de build emite um build minificado e transpilado do seu código do lado do cliente. Diferentemente do build de desenvolvimento, o build de produção não requer que o Node.js esteja instalado no servidor.

Você pode usar os [métodos padrão de implantação e hospedagem do ASP.NET Core](#).

## Executar o servidor CRA independentemente

O projeto está configurado para iniciar sua própria instância do Development Server do CRA em segundo plano quando o aplicativo ASP.NET Core é iniciado no modo de desenvolvimento. Isso é conveniente, porque significa que você não precisa executar um servidor separado manualmente.

Há uma desvantagem nessa configuração padrão. Cada vez que você modificar seu código C# e o aplicativo ASP.NET Core precisar ser reiniciado, o servidor CRA será reiniciado também. São necessários alguns segundos para iniciar um backup. Se você estiver fazendo edições frequentes de código C# e não quiser esperar o servidor CRA reiniciar, execute o servidor CRA externamente, independentemente do processo do ASP.NET Core. Para fazer isso:

1. Adicionar um *.env* do arquivo para o *ClientApp* subdiretório com a seguinte configuração:

```
BROWSER=none
```

Isso impedirá o navegador da web seja aberto ao iniciar o servidor CRA externamente.

2. Em um prompt de comando, vá para o subdiretório *ClientApp* e inicie o Development Server do CRA:

```
cd ClientApp  
npm start
```

3. Modifique o aplicativo ASP.NET Core para, em vez de iniciar uma instância do servidor CRA própria, usar a externa. Na classe *Startup*, substitua a invocação `spa.UseReactDevelopmentServer` pelo seguinte:

```
spa.UseProxyToSpaDevelopmentServer("http://localhost:3000");
```

Quando você iniciar seu aplicativo ASP.NET Core, ele não inicializará um servidor CRA. Em vez disso, a instância que você iniciou manualmente é usada. Isso permite a ele iniciar e reiniciar mais rapidamente. Ele não aguarda

mais que o aplicativo do React seja recompilado a cada vez.

# Usar o modelo de projeto do React com Redux com o ASP.NET Core

16/01/2019 • 2 minutes to read • [Edit Online](#)

## NOTE

Esta documentação não pertencem ao modelo de projeto React com Redux incluído no ASP.NET Core 2.0. Ele se refere ao modelo de reagir com Redux mais recente que você pode atualizar manualmente. O modelo está disponível no ASP.NET Core 2.1 ou posterior.

O modelo de projeto do React com Redux atualizado fornece um ponto inicial conveniente para aplicativos do ASP.NET Core usando convenções do React, do Redux e de CRA ([create-react-app](#)) para implementar uma IU (interface do usuário) avançada do lado do cliente.

Com exceção do comando de criação de projeto, todas as informações sobre o modelo React com Redux são as mesmas que aquelas sobre o modelo React. Para criar esse tipo de projeto, execute `dotnet new reactredux` em vez de `dotnet new react`. Para obter mais informações sobre a funcionalidade comum para ambos os modelos baseados em reagir, confira a [Documentação do modelo React](#).

Para obter informações sobre como configurar um aplicativo de subpropriedades React com Redux no IIS, consulte [ReactRedux modelo 2.1: Não é possível usar o SPA no IIS \(aspnet/modelagem #555\)](#).

# Usar JavaScriptServices para criar aplicativos de única página no ASP.NET Core

09/12/2018 • 21 minutes to read • [Edit Online](#)

Por [Scott Addie](#) e [Fiyaz Hasan](#)

Um aplicativo de página única (SPA) é um tipo popular de aplicativo web devido à sua experiência de usuário avançada inherente. A integração do lado do cliente estruturas de SPA ou bibliotecas, como [Angular](#) ou [reagir](#), com estruturas do lado do servidor, como ASP.NET Core pode ser difícil. [JavaScriptServices](#) foi desenvolvido para reduzir a fricção no processo de integração. Ele permite que a operação contínua entre o cliente diferentes e pilhas de tecnologia do servidor.

## O que é JavaScriptServices

JavaScriptServices é uma coleção de tecnologias do lado do cliente para o ASP.NET Core. Sua meta é posicionar o ASP.NET Core como plataforma de servidor preferencial dos desenvolvedores para a criação de SPAs.

JavaScriptServices consiste em três pacotes do NuGet distintos:

- [Microsoft.AspNetCore.NodeServices](#) (NodeServices)
- [Microsoft.AspNetCore.SpaServices](#) (SpaServices)
- [Microsoft.AspNetCore.SpaTemplates](#) (SpaTemplates)

Esses pacotes são úteis se você:

- Executar o JavaScript no servidor
- Usar uma estrutura de SPA ou biblioteca
- Criar ativos do lado do cliente com o Webpack

O foco deste artigo é colocado sobre como usar o pacote SpaServices.

## O que é SpaServices

SpaServices foi criado para posicionar o ASP.NET Core como plataforma de servidor preferencial dos desenvolvedores para a criação de SPAs. SpaServices não é necessário para desenvolver os SPAs com o ASP.NET Core, e ele não bloqueie você em uma estrutura de cliente específico.

SpaServices fornece infraestrutura úteis, como:

- [Pré-processamento do lado do servidor](#)
- [Middleware de desenvolvimento webpack](#)
- [Substituição do módulo quente](#)
- [Auxiliares de roteamentos](#)

Coletivamente, esses componentes de infraestrutura aprimoraram o fluxo de trabalho de desenvolvimento e a experiência de tempo de execução. Os componentes podem ser adotados individualmente.

## Pré-requisitos para usar SpaServices

Para trabalhar com SpaServices, instale o seguinte:

- [Node.js](#) (versão 6 ou posterior) com npm
  - Para verificar se esses componentes estão instalados e podem ser encontrados, execute o seguinte na linha de comando:

```
node -v && npm -v
```

Observação: Se você estiver implantando em um site do Azure, você não precisará fazer nada aqui — Node.js está instalado e disponível em ambientes de servidor.

- [SDK 2.0 ou posterior do .NET Core](#)

- Se você estiver no Windows usando o Visual Studio 2017, o SDK está instalado, selecionando o **desenvolvimento de plataforma cruzada do .NET Core** carga de trabalho.

- [Microsoft.AspNetCore.SpaServices](#) pacote do NuGet

## Pré-processamento do lado do servidor

Um aplicativo universal de (também conhecido como isomórficos) é um aplicativo de JavaScript pode ser executada tanto no servidor e cliente. Angular, React e outras estruturas populares fornecem uma plataforma universal para esse estilo de desenvolvimento do aplicativo. A ideia é renderizado primeiro os componentes do framework no servidor por meio do Node.js e delegar ainda mais a execução para o cliente.

ASP.NET Core [auxiliares de marca](#) fornecidos pelo SpaServices simplificar a implementação de pré-processamento do lado do servidor, chamando as funções de JavaScript no servidor.

### Pré-requisitos

Instale o seguinte:

- [ASP.NET pré-processamento](#) pacote npm:

```
npm i -S aspnet-prerendering
```

### Configuração

Os auxiliares de marca são feitos podem ser descobertos por meio do registro do namespace do projeto *viewimports.cshtml* arquivo:

```
@using SpaServicesSampleApp  
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers"  
@addTagHelper *, Microsoft.AspNetCore.SpaServices"
```

Esses auxiliares de marcação abstraem as complexidades de se comunicar diretamente com as APIs de baixo nível, utilizando uma sintaxe semelhante ao HTML dentro a exibição do Razor:

```
<app asp-prerender-module="ClientApp/dist/main-server">Loading...</app>
```

#### O `asp-prerender-module` auxiliar de marca

O `asp-prerender-module` auxiliar de marca, usado no exemplo de código anterior executa *ClientApp/dist/main-server.js* no servidor por meio do Node.js. Para clareza, *main-Server.js* arquivo é um artefato da tarefa em que a transcompilação TypeScript e JavaScript a [Webpack](#) processo de compilação. Webpack define um alias de ponto de entrada de `main-server`; e, passagem de grafo de dependência para este alias começa em de *ClientApp/inicialização-server.ts* arquivo:

```
entry: { 'main-server': './ClientApp/boot-server.ts' },
```

No exemplo a seguir Angular, o `ClientApp/initialização-server.ts` arquivo utiliza a `createServerRenderer` função e `RenderResult` tipo do `aspnet-prerendering` pacote npm para configurar a renderização do servidor por meio do Node.js. A marcação HTML destinada a renderização do lado do servidor é passada para uma chamada de função de resolução, que é encapsulada em um JavaScript fortemente tipada `Promise` objeto. O `Promise` significância do objeto é que ele fornece assincronamente a marcação HTML para a página para injeção no elemento de espaço reservado do DOM.

```
import { createServerRenderer, RenderResult } from 'aspnet-prerendering';

export default createServerRenderer(params => {
    const providers = [
        { provide: INITIAL_CONFIG, useValue: { document: '<app></app>', url: params.url } },
        { provide: 'ORIGIN_URL', useValue: params.origin }
    ];

    return platformDynamicServer(providers).bootstrapModule(AppModule).then(moduleRef => {
        const appRef = moduleRef.injector.get(ApplicationRef);
        const state = moduleRef.injector.get(PlatformState);
        const zone = moduleRef.injector.get(NgZone);

        return new Promise<RenderResult>((resolve, reject) => {
            zone.onError.subscribe(errorInfo => reject(errorInfo));
            appRef.isStable.first(isStable => isStable).subscribe(() => {
                // Because 'onStable' fires before 'onError', we have to delay slightly before
                // completing the request in case there's an error to report
                setImmediate(() => {
                    resolve({
                        html: state.renderToString()
                    });
                    moduleRef.destroy();
                });
            });
        });
    });
});
```

## O `asp-prerender-data` auxiliar de marca

Quando combinado com o `asp-prerender-module` auxiliar de marca, o `asp-prerender-data` auxiliar de marca pode ser usado para passar informações contextuais da exibição do Razor para o JavaScript do lado do servidor. Por exemplo, a marcação a seguir passa os dados do usuário para o `main-server` módulo:

```
<app asp-prerender-module="ClientApp/dist/main-server"
    asp-prerender-data='new {
        UserName = "John Doe"
    }'>Loading...</app>
```

Recebido `UserName` argumento for serializado usando o serializador JSON interno e é armazenado no `params.data` objeto. No exemplo a seguir Angular, os dados são usados para construir uma saudação personalizada dentro de um `h1` elemento:

```

import { createServerRenderer, RenderResult } from 'aspnet-prerendering';

export default createServerRenderer(params => {
    const providers = [
        { provide: INITIAL_CONFIG, useValue: { document: '<app></app>', url: params.url } },
        { provide: 'ORIGIN_URL', useValue: params.origin }
    ];

    return platformDynamicServer(providers).bootstrapModule(AppModule).then(moduleRef => {
        const appRef = moduleRef.injector.get(ApplicationRef);
        const state = moduleRef.injector.get(PlatformState);
        const zone = moduleRef.injector.get(NgZone);

        return new Promise<RenderResult>((resolve, reject) => {
            const result = `<h1>Hello, ${params.data.userName}</h1>`;

            zone.onError.subscribe(errorInfo => reject(errorInfo));
            appRef.isStable.first(isStable => isStable).subscribe(() => {
                // Because 'onStable' fires before 'onError', we have to delay slightly before
                // completing the request in case there's an error to report
                setImmediate(() => {
                    resolve({
                        html: result
                    });
                    moduleRef.destroy();
                });
            });
        });
    });
});

```

Observação: Os nomes de propriedade passados na auxiliares de marca são representados com **PascalCase** notação. Compare isso com JavaScript, onde os mesmos nomes de propriedade são representados com **camelCase**. A configuração de serialização JSON padrão é responsável por essa diferença.

Para expandir o exemplo de código anterior, dados podem ser passados do servidor para o modo de exibição por hydrating a `globals` propriedade fornecida para o `resolve` função:

```

import { createServerRenderer, RenderResult } from 'aspnet-prerendering';

export default createServerRenderer(params => {
    const providers = [
        { provide: INITIAL_CONFIG, useValue: { document: '<app></app>', url: params.url } },
        { provide: 'ORIGIN_URL', useValue: params.origin }
    ];

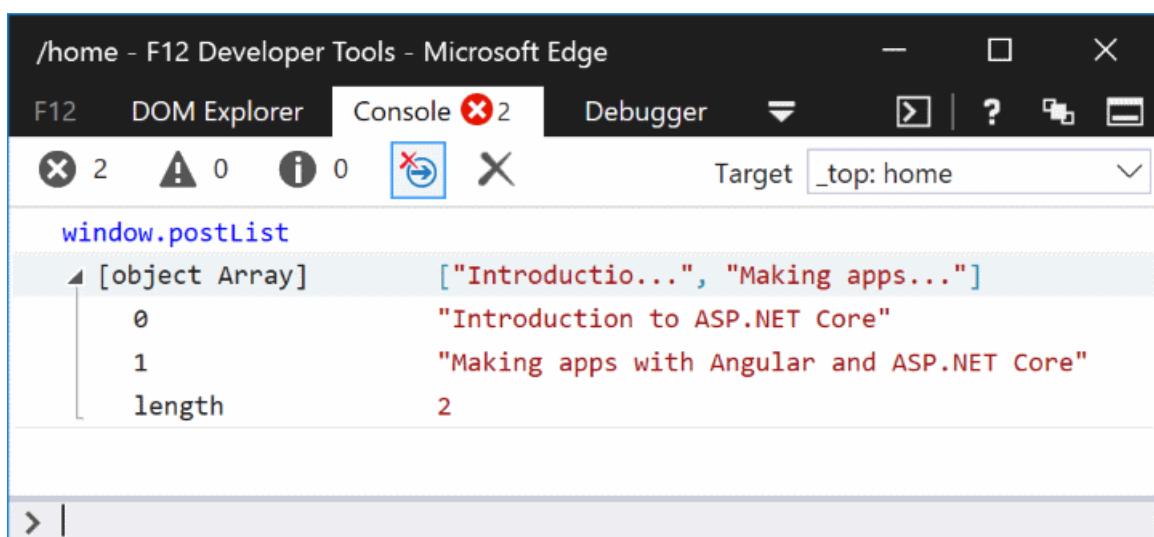
    return platformDynamicServer(providers).bootstrapModule(AppModule).then(moduleRef => {
        const appRef = moduleRef.injector.get(ApplicationRef);
        const state = moduleRef.injector.get(PlatformState);
        const zone = moduleRef.injector.get(NgZone);

        return new Promise<RenderResult>((resolve, reject) => {
            const result = `<h1>Hello, ${params.data.userName}</h1>`;

            zone.onError.subscribe(errorInfo => reject(errorInfo));
            appRef.isStable.first(isStable => isStable).subscribe(() => {
                // Because 'onStable' fires before 'onError', we have to delay slightly before
                // completing the request in case there's an error to report
                setImmediate(() => {
                    resolve({
                        html: result,
                        globals: {
                            postList: [
                                'Introduction to ASP.NET Core',
                                'Making apps with Angular and ASP.NET Core'
                            ]
                        }
                    });
                    moduleRef.destroy();
                });
            });
        });
    });
});

```

O `window.postList` matriz definida dentro de `globals` objeto está anexado para o navegador global `window` objeto. Essa elevação de variáveis em escopo global elimina a duplicação de esforço, particularmente, pois pertence ao carregar os mesmos dados, uma vez no servidor e novamente no cliente.



## Middleware de desenvolvimento webpack

[Middleware de desenvolvimento webpack](#) introduz um fluxo de trabalho de desenvolvimento simplificado no qual o Webpack cria recursos sob demanda. O middleware automaticamente compila e atende recursos do lado do cliente quando uma página é recarregada no navegador. A abordagem alternativa é invocar manualmente

Webpack por meio do script de compilação do projeto npm quando uma dependência de terceiros ou o código personalizado é alterado. Script de construção de um npm *Package.JSON* arquivo é mostrado no exemplo a seguir:

```
"build": "npm run build:vendor && npm run build:custom",
```

## Pré-requisitos

Instale o seguinte:

- [ASP.NET webpack](#) pacote npm:

```
npm i -D aspnet-webpack
```

## Configuração

Webpack Dev Middleware é registrado no pipeline de solicitação HTTP por meio de código a seguir na *Startup.cs* do arquivo `Configure` método:

```
if (env.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
    app.UseWebpackDevMiddleware();
}

else
{
    app.UseExceptionHandler("/Home/Error");
}

// Call UseWebpackDevMiddleware before UseStaticFiles
app.UseStaticFiles();
```

O `UseWebpackDevMiddleware` método de extensão deve ser chamado antes [Registrando a hospedagem de arquivos estáticos](#) por meio de `UseStaticFiles` método de extensão. Por motivos de segurança, registre o middleware somente quando o aplicativo é executado no modo de desenvolvimento.

O *webpack.config.js* do arquivo `output.publicPath` propriedade informa o middleware para observar o `dist` pasta para que as alterações:

```
module.exports = (env) => {
    output: {
        filename: '[name].js',
        publicPath: '/dist/' // Webpack dev middleware, if enabled, handles requests for this URL prefix
    },
}
```

## Substituição do módulo quente

Pense do Webpack [módulo de substituição a quente](#) recurso (HMR) como uma evolução do [Webpack Dev Middleware](#). HMR apresenta todos os mesmos benefícios, mas ela simplifica ainda mais o fluxo de trabalho de desenvolvimento, atualizando automaticamente o conteúdo da página depois de compilar as alterações. Não confunda isso com uma atualização do navegador, que poderia interferir com o estado atual de na memória e a sessão de depuração do SPA. Há um link em tempo real entre o serviço de Middleware de desenvolvimento Webpack e o navegador, o que significa que alterações são enviadas para o navegador.

## Pré-requisitos

Instale o seguinte:

- middleware hot webpack pacote npm:

```
npm i -D webpack-hot-middleware
```

## Configuração

O componente HMR deve ser registrado no pipeline de solicitação HTTP do MVC no `configure` método:

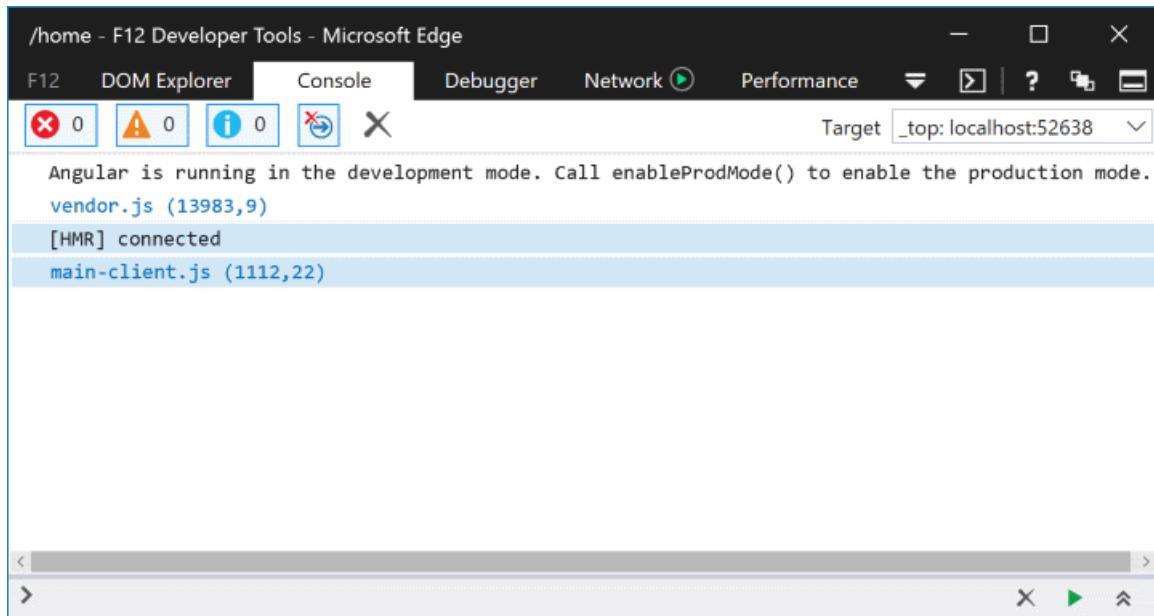
```
app.UseWebpackDevMiddleware(new WebpackDevMiddlewareOptions {
    HotModuleReplacement = true
});
```

Como ocorria com [Webpack Dev Middleware](#), o `UseWebpackDevMiddleware` método de extensão deve ser chamado antes do `UseStaticFiles` método de extensão. Por motivos de segurança, registre o middleware somente quando o aplicativo é executado no modo de desenvolvimento.

O `webpack.config.js` arquivo deve definir um `plugins` de matriz, mesmo se ela for deixada em branco:

```
module.exports = (env) => {
    plugins: [new CheckerPlugin()]
```

Depois de carregar o aplicativo no navegador, a guia Console de ferramentas de desenvolvedor fornece confirmação de ativação de HMR:



## Auxiliares de roteamentos

A maioria dos SPAs com base no ASP.NET Core, você desejará roteamento do lado do cliente além do roteamento do lado do servidor. Os sistemas de roteamento do SPA e o MVC podem trabalhar de forma independente, sem interferência. Há, no entanto, os desafios de uma borda caso apresentando: identificando as respostas HTTP 404.

Considere o cenário em que uma rota sem extensão de `/some/page` é usado. Suponha que a solicitação não-correspondência de padrão uma rota do lado do servidor, mas seu padrão corresponde a uma rota do lado do cliente. Agora, considere uma solicitação de entrada para `/images/user-512.png`, que geralmente espera encontrar um arquivo de imagem no servidor. Se esse caminho de recurso solicitado não corresponder a qualquer rota do lado do servidor ou um arquivo estático, é improvável que o aplicativo do lado do cliente seria lidar com isso, você geralmente deseja retornar um código de status HTTP 404.

## Pré-requisitos

Instale o seguinte:

- O pacote npm de roteamento do lado do cliente. Usando Angular como um exemplo:

```
npm i -S @angular/router
```

## Configuração

Um método de extensão denominado `MapSpaFallbackRoute` é usado no `Configure` método:

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");

    routes.MapSpaFallbackRoute(
        name: "spa-fallback",
        defaults: new { controller = "Home", action = "Index" });
});
```

Dica: As rotas são avaliadas na ordem em que eles foram configurados. Consequentemente, o `default` rota no exemplo de código anterior é usada primeiro para correspondência de padrões.

## Criar um novo projeto

JavaScriptServices fornece modelos de aplicativos pré-configurados. SpaServices é usado nesses modelos, em conjunto com diferentes estruturas e bibliotecas, como Angular, React e Redux.

Esses modelos podem ser instalados por meio da CLI do .NET Core, executando o seguinte comando:

```
dotnet new --install Microsoft.AspNetCore.SpaTemplates::*
```

Uma lista de modelos SPA disponíveis é exibida:

MODELOS	NOME CURTO	IDIOMA	MARCAS
MVC ASP.NET Core com Angular	angular	[C#]	Web/MVC/SPA
MVC ASP.NET Core com React.js	react	[C#]	Web/MVC/SPA
MVC ASP.NET Core com React.js e Redux	reactredux	[C#]	Web/MVC/SPA

Para criar um novo projeto usando um dos modelos do SPA, inclua o **nome curto** do modelo na `dotnet novo` comando. O comando a seguir cria um aplicativo Angular com ASP.NET Core MVC configurado para o lado do servidor:

```
dotnet new angular
```

## Definir o modo de configuração de tempo de execução

Existem dois modos de configuração de tempo de execução principal:

- **Desenvolvimento:**

- Inclui mapas de código-fonte para facilitar a depuração.
- Não Otimize o código do lado do cliente para o desempenho.

- **Produção:**

- Exclui os mapas de origem.
- Otimiza o código do lado do cliente por meio do agrupamento e minificação.

O ASP.NET Core usa uma variável de ambiente denominada `ASPNETCORE_ENVIRONMENT` para armazenar o modo de configuração. Ver [defina o ambiente](#) para obter mais informações.

### Executando com a CLI do .NET Core

Restaure os pacotes de npm e NuGet necessários, executando o seguinte comando na raiz do projeto:

```
dotnet restore && npm i
```

Compilar e executar o aplicativo:

```
dotnet run
```

O aplicativo é iniciado no localhost de acordo com o [modo de configuração de tempo de execução](#). Navegando até `http://localhost:5000` no navegador exibe a página de aterrissagem.

### Em execução com o Visual Studio 2017

Abra o `.csproj` arquivo gerado pelo [dotnet novo](#) comando. Os pacotes NuGet e npm necessários são restaurados automaticamente ao projeto aberto. Esse processo de restauração pode demorar alguns minutos e o aplicativo está pronto para ser executado quando ele for concluído. Clique no botão de execução verde ou pressione `Ctrl + F5`, e o navegador abre a página de aterrissagem do aplicativo. O aplicativo é executado no localhost de acordo com o [modo de configuração de tempo de execução](#).

## Testar o aplicativo

Modelos SpaServices são pré-configuradas para executar testes do lado do cliente usando [Karma](#) e [Jasmine](#). Jasmine é uma estrutura de teste para JavaScript, de unidade popular enquanto Karma é um executor de teste para que esses testes. Karma está configurado para funcionar com o [Webpack Dev Middleware](#), de modo que o desenvolvedor não é necessário parar e executar o teste sempre que forem feitas alterações. Se ele é o código em execução no caso de teste ou caso de teste em si, o teste é executado automaticamente.

Usando o aplicativo Angular como exemplo, dois casos de teste Jasmine já são fornecidos para o `CounterComponent` no `counter.component.spec.ts` arquivo:

```
it('should display a title', async(() => {
    const titleText = fixture.nativeElement.querySelector('h1').textContent;
    expect(titleText).toEqual('Counter');
}));

it('should start with count 0, then increments by 1 when clicked', async(() => {
    const countElement = fixture.nativeElement.querySelector('strong');
    expect(countElement.textContent).toEqual('0');

    const incrementButton = fixture.nativeElement.querySelector('button');
    incrementButton.click();
    fixture.detectChanges();
    expect(countElement.textContent).toEqual('1');
}));
```

Abra o prompt de comando de *ClientApp* diretório. Execute o seguinte comando:

```
npm test
```

O script inicia o executor de teste Karma, que lê as configurações definidas na *karma.conf.js* arquivo. Entre outras configurações, o *karma.conf.js* identifica os arquivos de teste para ser executada por meio do seu `files` matriz:

```
module.exports = function (config) {
  config.set({
    files: [
      '../../wwwroot/dist/vendor.js',
      './boot-tests.ts'
    ],
  },
```

## Publicando o aplicativo

Combinando os ativos gerados do lado do cliente e os artefatos publicados do ASP.NET Core em um pacote pronto para implantar pode ser complicado. Felizmente, SpaServices orquestra o processo de publicação inteira com um destino MSBuild personalizado chamado `RunWebpack`:

```
<Target Name="RunWebpack" AfterTargets="ComputeFilesToPublish">
  <!-- As part of publishing, ensure the JS resources are freshly built in production mode -->
  <Exec Command="npm install" />
  <Exec Command="node node_modules/webpack/bin/webpack.js --config webpack.config.vendor.js --env.prod" />
  <Exec Command="node node_modules/webpack/bin/webpack.js --env.prod" />

  <!-- Include the newly-built files in the publish output -->
  <ItemGroup>
    <DistFiles Include="wwwroot\dist\**; ClientApp\dist\**" />
    <ResolvedFileToPublish Include="@({DistFiles->'%(FullPath)'})" Exclude="@({ResolvedFileToPublish})">
      <RelativePath>%({DistFiles.Identity})</RelativePath>
      <CopyToPublishDirectory>PreserveNewest</CopyToPublishDirectory>
    </ResolvedFileToPublish>
  </ItemGroup>
</Target>
```

O destino do MSBuild tem as seguintes responsabilidades:

1. Restaure os pacotes de npm
2. Criar uma compilação de nível de produção dos ativos de terceiros, do lado do cliente
3. Criar uma compilação de nível de produção dos ativos do cliente personalizados
4. Copie os ativos gerados pelo Webpack para a pasta de publicação

O destino do MSBuild é chamado durante a execução:

```
dotnet publish -c Release
```

## Recursos adicionais

- [Docs angular](#)

# Hospedar e implantar o ASP.NET Core

13/12/2018 • 7 minutes to read • [Edit Online](#)

Em geral, implantar um aplicativo ASP.NET Core em um ambiente de hospedagem:

- Implante o aplicativo publicado em uma pasta no servidor de hospedagem.
- Configure um gerenciador de processo que inicia o aplicativo quando a solicitação chega e reinicia-o depois que ele falha ou que o servidor é reinicializado.
- No caso da configuração de um proxy reverso, defina um proxy reverso para encaminhar solicitações para o aplicativo.

## Publicar em uma pasta

O comando `dotnet publish` compila o código do aplicativo e copia os arquivos necessários para executar o aplicativo em uma pasta *publish*. Ao implantar do Visual Studio, a etapa `dotnet publish` ocorre automaticamente antes de os arquivos serem copiados para o destino da implantação.

### Conteúdo da pasta

A pasta *publish* contém um ou mais arquivos do assembly, dependências e, opcionalmente, o tempo de execução do .NET.

Um aplicativo .NET Core pode ser publicado como uma *implantação autocontida* ou uma *implantação dependente de estrutura*. Se o aplicativo for autocontido, os arquivos do assembly que contêm o tempo de execução do .NET serão incluídos na pasta *publish*. Se o aplicativo depender da estrutura, os arquivos de tempo de execução do .NET não serão incluídos porque o aplicativo tem uma referência para uma versão do .NET que está instalada no servidor. O modelo de implantação padrão é dependente da estrutura. Para obter mais informações, consulte [Implantação de aplicativos .NET Core](#).

Além de arquivos *.exe* e *.dll*, a pasta *publish* para um aplicativo ASP.NET Core normalmente contém arquivos de configuração, ativos estáticos e exibições do MVC. Para obter mais informações, consulte [Estrutura do diretório do ASP.NET Core](#).

## Configure um gerenciador de processo

Um aplicativo ASP.NET Core é um aplicativo de console que deve ser iniciado quando um servidor é inicializado e reiniciado após falhas. Para automatizar inicializações e reinicializações, um gerenciador de processo é necessário. Os gerenciadores de processo mais comuns para o ASP.NET Core são:

- Linux
  - [Nginx](#)
  - [Apache](#)
- Windows
  - [IIS](#)
  - [Serviço Windows](#)

## Configurar um proxy reverso

Se o aplicativo usar o servidor Web do servidor [Kestrel](#), você poderá usar [Nginx](#), [Apache](#) ou [IIS](#) como um servidor proxy reverso. Um servidor proxy reverso recebe solicitações HTTP da Internet e encaminha-as para o Kestrel.

Qualquer configuração —com ou sem um servidor proxy reverso— é uma configuração de hospedagem compatível com o ASP.NET Core 2.0 ou aplicativos posteriores. Para obter mais informações, consulte [Quando usar Kestrel com um proxy reverso](#).

Se o aplicativo usar o servidor [Kestrel](#) e for exposto à Internet, você deverá usar [Nginx](#), [Apache](#) ou [IIS](#) como um servidor proxy reverso. Um servidor proxy reverso recebe solicitações HTTP da Internet e encaminha-as para o Kestrel. O principal motivo para usar um proxy reverso é a segurança. Para obter mais informações, consulte [Quando usar Kestrel com um proxy reverso](#).

## Servidor proxy e cenários de balanceador de carga

Configuração adicional pode ser necessária para aplicativos hospedados atrás de servidores proxy e平衡adores de carga. Sem configuração adicional, um aplicativo pode não ter acesso ao esquema (HTTP/HTTPS) e ao endereço IP remoto em que uma solicitação foi originada. Para obter mais informações, veja [Configurar o ASP.NET Core para trabalhar com servidores proxy e balanceadores de carga](#).

## Usar o Visual Studio e o MSBuild para automatizar as implantações

A implantação muitas vezes requer tarefas adicionais além de copiar a saída da `dotnet publish` para um servidor. Por exemplo, arquivos extras podem ser necessários ou excluídos da pasta `publish`. O MSBuild, que é usado pelo Visual Studio para implantação da Web, pode ser personalizado para fazer muitas outras tarefas durante a implantação. Para saber mais, confira [Perfis de publicação do Visual Studio para a implantação do aplicativo ASP.NET Core](#) e o livro [Using MSBuild and Team Foundation Build](#).

Você pode implantar diretamente do Visual Studio para o Serviço de Aplicativo do Azure usando [o recurso Publicar na Web](#) ou usando o [suporte ao Git interno](#). O Azure DevOps Services dá suporte à [implantação contínua para o Serviço de Aplicativo do Azure](#). Para obter mais informações, confira [DevOps com ASP.NET Core e Azure](#).

## Publicar no Azure

Confira [Publicar um aplicativo ASP.NET Core no Azure com o Visual Studio](#) para obter instruções sobre como publicar um aplicativo no Azure usando o Visual Studio. Um exemplo adicional é fornecido em [Criar um aplicativo Web ASP.NET Core no Azure](#).

## Publicar com MSDeploy no Windows

Confira [Perfis de publicação do Visual Studio para a implantação do aplicativo ASP.NET Core](#) para obter instruções sobre como publicar um aplicativo com um perfil de publicação do Visual Studio, inclusive de um prompt de comando do Windows, usando o comando `dotnet msbuild`.

## Hospedar em uma web farm

Para obter informações sobre a configuração para hospedar aplicativos do ASP.NET Core em um ambiente de web farm (por exemplo, a implantação de várias instâncias do aplicativo para escalabilidade), veja [Hospedar o ASP.NET Core em um web farm](#).

## Executar verificações de integridade

Use o Middleware de verificação de integridade para executar verificações de integridade em um aplicativo e suas dependências. Para obter mais informações, consulte [Verificações de integridade no ASP.NET Core](#).

## Recursos adicionais

- Hospedar o ASP.NET Core em contêineres do Docker
- Solucionar problemas de projetos do ASP.NET Core

# Implantar aplicativos ASP.NET Core no Serviço de Aplicativo do Azure

02/01/2019 • 18 minutes to read • [Edit Online](#)

Serviço de Aplicativo do Azure é um serviço de plataforma de computação em nuvem da Microsoft para hospedar aplicativos Web, incluindo o ASP.NET Core.

## Recursos úteis

A [Documentação de Aplicativos Web](#) do Azure é a página inicial para documentação de Azure Apps, tutoriais, exemplos, guias de instruções e outros recursos. Dois tutoriais importantes que pertencem à hospedagem de aplicativos ASP.NET Core são:

### [Início Rápido: Criar um aplicativo Web ASP.NET Core no Azure](#)

Use o Visual Studio para criar e implantar um aplicativo Web ASP.NET Core no Serviço de Aplicativo do Azure no Windows.

### [Início Rápido: Criar um aplicativo Web .NET Core no Serviço de Aplicativo no Linux](#)

Use a linha de comando do Visual Studio para criar e implantar um aplicativo Web ASP.NET Core no Serviço de Aplicativo do Azure no Linux.

Os artigos a seguir estão disponíveis na documentação do ASP.NET Core:

### [Publicar um aplicativo ASP.NET Core no Azure com o Visual Studio](#)

Aprenda como publicar um aplicativo ASP.NET Core no Serviço de Aplicativo do Azure usando o Visual Studio.

### [Implantação contínua no Azure com o Visual Studio e o GIT com o ASP.NET Core](#)

Saiba como criar um aplicativo Web ASP.NET Core usando o Visual Studio e implantá-lo no Serviço de Aplicativo do Azure, usando o Git para implantação contínua.

### [Criar seu primeiro pipeline com o Azure Pipelines](#)

Configurar um build de CI para um aplicativo ASP.NET Core e, em seguida, criar uma versão de implantação contínua para o Serviço de Aplicativo do Azure.

### [Área restrita de aplicativo Web do Azure](#)

Descubra as limitações de tempo de execução do Serviço de Aplicativo do Azure impostas pela plataforma de Aplicativos do Azure.

## Configuração do aplicativo

### **Plataforma**

Os tempos de execução para aplicativos de 32 bits (x86) e 64 bits (x64) estão presentes no Serviço de Aplicativo do Azure. O [SDK do .NET Core](#) disponível no Serviço de Aplicativo do Azure é de 32 bits, mas você pode implantar aplicativos de 64 bits usando o console do [Kudu](#) ou o [MSDeploy com um perfil de publicação do Visual Studio ou um comando da CLI](#).

Para aplicativos com dependências nativas, os tempos de execução para aplicativos de 32 bits (x86) estão presentes no Serviço de Aplicativo do Azure. O [SDK do .NET Core](#) disponível no Serviço de Aplicativo é de 32 bits.

### **Pacotes**

Incluem os seguintes pacotes NuGet para fornecer recursos de registro automático em log para aplicativos implantados no Serviço de Aplicativo do Azure:

- O [Microsoft.AspNetCore.AzureAppServices.HostingStartup](#) usa [IHostingStartup](#) para fornecer integração leve do ASP.NET Core com o Serviço de Aplicativo do Azure. Os recursos de registro em log adicionais são fornecidos pelo pacote [Microsoft.AspNetCore.AzureAppServicesIntegration](#).
- [Microsoft.AspNetCore.AzureAppServicesIntegration](#) executa [AddAzureWebAppDiagnostics](#) para adicionar provedores de log de diagnósticos do Serviço de Aplicativo do Azure no pacote [Microsoft.Extensions.Logging.AzureAppServices](#).
- [Microsoft.Extensions.Logging.AzureAppServices](#) fornece implementações de agente para dar suporte a recursos de streaming de log e logs de diagnóstico do Serviço de Aplicativo do Azure.

Os pacotes anteriores não estão disponíveis no [Microsoft.AspNetCore.App metapackage](#). Os aplicativos que são direcionados ao .NET Framework ou fazem referência a um metapacote [Microsoft.AspNetCore.App](#) precisam referenciar explicitamente os pacotes individuais no arquivo de projeto do aplicativo.

## Substituir a configuração do aplicativo no Portal do Azure

As configurações do aplicativo no portal do Azure permitem definir variáveis de ambiente para o aplicativo. As variáveis de ambiente podem ser consumidas pelo [Provedor de configuração de variáveis de ambiente](#).

Quando uma configuração de aplicativo é criada ou modificada no Portal do Azure e o botão **Salvar** é selecionado, o Aplicativo Azure é reiniciado. A variável de ambiente estará disponível para o aplicativo após o serviço ser reiniciado.

Quando o aplicativo usa o [host da Web](#) e cria o host usando [WebHost.CreateDefaultBuilder](#), as variáveis de ambiente que configuraram o host usam o prefixo `ASPNETCORE_`. Para saber mais, confira [Host da Web do ASP.NET Core](#) e o [Provedor de configuração de variáveis de ambiente](#).

Quando o aplicativo usa o [host genérico](#), as variáveis de ambiente não são carregadas na configuração do aplicativo por padrão, e o provedor de configuração deve ser adicionado pelo desenvolvedor. O desenvolvedor determina o prefixo da variável de ambiente quando o provedor de configuração é adicionado. Para saber mais, confira [Host Genérico .NET](#) e o [Provedor de configuração de variáveis de ambiente](#).

## Servidor proxy e cenários de balanceador de carga

O Middleware de integração do IIS, que configura Middleware de cabeçalhos encaminhados, e o módulo do ASP.NET Core são configurados para encaminhar o esquema (HTTP/HTTPS) e o endereço IP remoto de onde a solicitação foi originada. Configuração adicional pode ser necessária para aplicativos hospedados atrás de servidores proxy adicionais e平衡adores de carga. Para obter mais informações, veja [Configurar o ASP.NET Core para trabalhar com servidores proxy e balanceadores de carga](#).

## Monitoramento e registro em log

Os aplicativos ASP.NET Core implantados no Serviço de Aplicativo recebem automaticamente uma extensão do Serviço de Aplicativo, **Extensões de Log do ASP.NET Core**. A extensão habilita o registro em log do Azure.

Para monitoramento, registro em log e informações de solução de problemas, veja os seguintes artigos:

### [Como: monitorar aplicativos no Serviço de Aplicativo do Azure](#)

Saiba como examinar as cotas e métricas para aplicativos e planos do Serviço de Aplicativo.

### [Habilitar log de diagnósticos para aplicativos Web no Serviço de Aplicativo do Azure](#)

Descubra como habilitar e acessar o log de diagnósticos para os códigos de status HTTP, solicitações com falha e atividade do servidor Web.

## [Tratar erros no ASP.NET Core](#)

Entenda as abordagens comuns para o tratamento de erros em aplicativos ASP.NET Core.

## [Solucionar problemas no ASP.NET Core no Serviço de Aplicativo do Azure](#)

Saiba como diagnosticar problemas com implantações do Serviço de Aplicativo do Azure com aplicativos ASP.NET Core.

## [Referência de erros comuns para o Serviço de Aplicativo do Azure e o IIS com o ASP.NET Core](#)

Consulte os erros comuns de configuração de implantação para aplicativos hospedados pelo Serviço de Aplicativo do Azure/IIS com orientação para solução de problemas.

# Anel de chave de proteção de dados e slots de implantação

[Chaves de proteção de dados](#) são mantidas na pasta %HOME%\ASP.NET\DataProtection-Keys. Essa pasta é apoiada pelo repositório de rede e é sincronizada em todos os computadores que hospedam o aplicativo. As chaves não são protegidas em repouso. Essa pasta fornece o anel de chave para todas as instâncias de um aplicativo em um único slot de implantação. Slots de implantação separados, como de preparo e produção, não compartilham um anel de chave.

Quando ocorre a troca entre os slots de implantação, nenhum sistema que usa a proteção de dados consegue descriptografar dados armazenados usando o anel de chave dentro do slot anterior. O middleware de cookie do ASP.NET usa a proteção de dados para proteger seus cookies. Com isso, os usuários são desconectados de um aplicativo que usa o Middleware de Cookie padrão do ASP.NET. Para uma solução de anel de chave independente de slot, use um provedor de anel de chave externo, como:

- Armazenamento do Blobs do Azure
- Azure Key Vault
- Repositório SQL
- Cache redis

Para obter mais informações, consulte [Provedores de armazenamento de chaves no ASP.NET Core](#).

# Implantar a versão de visualização do ASP.NET Core para o Serviço de Aplicativo do Azure

Use uma das abordagens a seguir:

- [Instalar a extensão de site da versão prévia](#).
- [Implantar o aplicativo autocontido](#).
- [Usar o Docker com aplicativos Web para contêineres](#).

## **Instalar a extensão de site de visualização**

Se houver problemas ao usar a extensão de site de visualização, abra um problema no [GitHub](#).

1. No portal do Azure, navegue até o Serviço de Aplicativo.
2. Selecione o aplicativo Web.
3. Insira "ex" na caixa de pesquisa para filtrar por "Extensões" ou role para baixo na lista de ferramentas de gerenciamento.
4. Selecione **Extensões**.
5. Selecione **Adicionar**.
6. Selecione a extensão **Tempo de execução do ASP.NET Core {X.Y} ({x64|x86})** na lista, em que {X.Y} é a versão prévia do ASP.NET Core e {x64|x86} especifica a plataforma.
7. Selecione **OK** para aceitar os termos legais.
8. Selecione **OK** para instalar a extensão.

Quando a operação for concluída, a versão prévia mais recente do .NET Core será instalada. Verifique a instalação:

1. Selecione **Ferramentas Avançadas**.
2. Selecione **Acessar** em **Ferramentas Avançadas**.
3. Selecione o item de menu **Console de depuração > PowerShell**.
4. No prompt do PowerShell, execute o seguinte comando. Substitua a versão do tempo de execução do ASP.NET Core por `{X.Y}` e a plataforma por `{PLATFORM}` no comando:

```
Test-Path D:\home\SiteExtensions\AspNetCoreRuntime.{X.Y}.{PLATFORM}\
```

O comando retornará `True` quando o tempo de execução da versão prévia x64 estiver instalado.

#### NOTE

A arquitetura da plataforma (x86/x64) de um aplicativo dos Serviços de Aplicativos é definida nas configurações do aplicativo no portal do Azure para aplicativos hospedados em um nível de hospedagem de computação da série A ou melhor. Se o aplicativo for executado no modo em processo e a arquitetura da plataforma estiver configurada para 64 bits (x64), o Módulo do ASP.NET Core usará o tempo de execução da versão prévia de 64 bits, se estiver presente. Instale a extensão **Tempo de execução do ASP.NET Core {X.Y} (x64)**.

Depois de instalar o tempo de execução da versão prévia x64, execute o seguinte comando na janela de comando do Kudu PowerShell para verificar a instalação. Substitua a versão de tempo de execução do ASP.NET Core por `{X.Y}` no comando:

```
Test-Path D:\home\SiteExtensions\AspNetCoreRuntime.{X.Y}.x64\
```

O comando retornará `True` quando o tempo de execução da versão prévia x64 estiver instalado.

#### NOTE

As **Extensões do ASP.NET Core** habilitam uma funcionalidade adicional para o ASP.NET Core nos Serviços de Aplicativo do Azure, como a habilitação do registro em log do Azure. A extensão é instalada automaticamente durante a implantação do Visual Studio. Se a extensão não estiver instalada, instale-a para o aplicativo.

## Usar a extensão de site de visualização com um modelo do ARM

Se um modelo do ARM for usado para criar e implantar aplicativos, o tipo de recurso `siteextensions` poderá ser usado para adicionar a extensão de site a um aplicativo Web. Por exemplo:

```
{
  "type": "siteextensions",
  "name": "AspNetCoreRuntime",
  "apiVersion": "2015-04-01",
  "location": "[resourceGroup().location]",
  "properties": {
    "version": "[parameters('aspnetcoreVersion')]"
  },
  "dependsOn": [
    "[resourceId('Microsoft.Web/Sites', parameters('siteName'))]"
  ]
}
```

## Implantar o aplicativo autocontido

Uma SCD (implantação autocontida) voltada para um tempo de execução de versão prévia transporta o tempo de execução da versão prévia na implantação.

Ao implantar um aplicativo autocontido:

- O site no Serviço de Aplicativo do Azure não exige a [extensão de site da versão prévia](#).
- O aplicativo precisa ser publicado após uma abordagem diferente da publicação em uma [FDD \(implantação dependente de estrutura\)](#).

#### Publicar no Visual Studio

1. Selecione **Build > Publicar {Nome do Aplicativo}** na barra de ferramentas do Visual Studio.
2. Na caixa de diálogo **Escolher um destino de publicação**, confirme se o **Serviço de Aplicativo** está selecionado.
3. Selecione **Avançado**. A caixa de diálogo **Publicar** será aberta.
4. Na caixa de diálogo **Publicar**:
  - Confirme se a configuração **Versão** está selecionada.
  - Abra a lista suspensa **Modo de Implantação** e selecione **Autocontido**.
  - Selecione o tempo de execução de destino na lista suspensa **Tempo de Execução de Destino**. O padrão é `win-x86`.
  - Se você precisar remover arquivos adicionais após a implantação, abra as **Opções de Publicação do Arquivo** e marque a caixa de seleção para remover arquivos adicionais no destino.
  - Selecione **Salvar**.
5. Crie um novo site ou atualize um site existente seguindo as solicitações restantes do assistente de publicação.

#### Publicar usando as ferramentas de CLI (interface de linha de comando)

1. No arquivo de projeto, especifique um ou mais [IDs \(identificadores de tempo de execução\)](#). Use `<RuntimeIdentifier>` (singular) para um único RID ou use `<RuntimeIdentifiers>` (plural) para fornecer uma lista de RIDs delimitada por ponto e vírgula. No exemplo a seguir, o RID `win-x86` é especificado:

```
<PropertyGroup>
  <TargetFramework>netcoreapp2.1</TargetFramework>
  <RuntimeIdentifier>win-x86</RuntimeIdentifier>
</PropertyGroup>
```

2. Em um shell de comando, publique o aplicativo na configuração de Versão do tempo de execução do host com o comando `dotnet publish`. No exemplo a seguir, o aplicativo é publicado para o RID `win-x86`. O RID fornecido para a opção `--runtime` precisa ser fornecida na propriedade `<RuntimeIdentifier>` (ou `<RuntimeIdentifiers>`) no arquivo de projeto.

```
dotnet publish --configuration Release --runtime win-x86
```

3. Mova o conteúdo do diretório `bin/Release/{TARGET FRAMEWORK}/{RUNTIME IDENTIFIER}/publish` para o site no Serviço de Aplicativo.

#### Usar o Docker com aplicativos Web para contêineres

O [Docker Hub](#) contém as imagens de versão prévia do Docker mais recentes. As imagens podem ser usadas como uma imagem de base. Use a imagem e implante aplicativos Web para contêineres normalmente.

## Configurações de protocolo (HTTPS)

As associações de protocolo de segurança permitem que você especifique um certificado a ser usado ao responder a solicitações em HTTPS. A associação requer um certificado privado válido (.pfx) emitido para o nome do host específico. Para obter mais informações, confira [Tutorial: vincular um certificado SSL](#)

personalizado ao serviço Aplicativos Web do Azure.

## Recursos adicionais

- [Visão geral de aplicativos Web \(vídeo de visão geral com 5 minutos\)](#)
- [Serviço de Aplicativo do Azure: o melhor lugar para hospedar seus aplicativos .NET \(vídeo de visão geral com 55 minutos\)](#)
- [Azure Friday: experiência de diagnóstico e solução de problemas do Serviço de Aplicativo do Azure \(vídeo com 12 minutos\)](#)
- [Visão geral de diagnóstico do Serviço de Aplicativo do Azure](#)
- [Hospedar o ASP.NET Core em um web farm](#)

O Serviço de Aplicativo do Azure no Windows Server usa o [IIS \(Serviços de Informações da Internet\)](#). Os tópicos a seguir estão relacionados com a tecnologia subjacente do IIS:

- [Hospedar o ASP.NET Core no Windows com o IIS](#)
- [Módulo do ASP.NET Core](#)
- [Módulo do ASP.NET Core](#)
- [Módulos do IIS com o ASP.NET Core](#)
- [Biblioteca do Microsoft TechNet: Windows Server](#)

# Publicar um aplicativo ASP.NET Core no Azure com o Visual Studio

10/01/2019 • 7 minutes to read • [Edit Online](#)

Por [Rick Anderson](#) e [Cesar Blum Silveira](#)

## IMPORTANT

### Versões prévias do ASP.NET Core com o Serviço de Aplicativo do Azure

Versões prévias do ASP.NET Core não são implantadas para o Serviço de Aplicativo do Azure por padrão. Para hospedar um aplicativo que usa uma versão prévia do ASP.NET Core, veja [Implantar versão prévia do ASP.NET Core para o Serviço de Aplicativo do Azure](#).

Confira [Publicar no Azure do Visual Studio para Mac](#) se você estiver trabalhando no macOS.

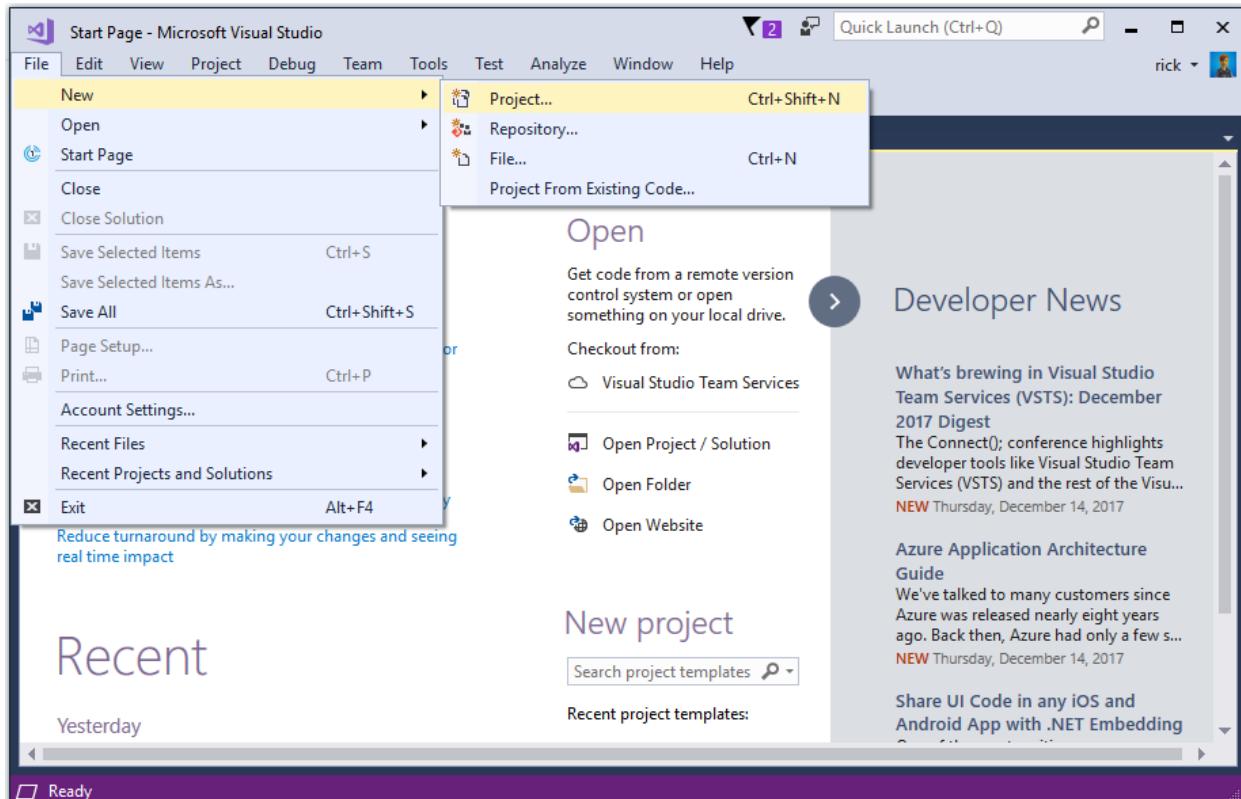
Para solucionar um problema de implantação do Serviço de Aplicativo, confira [Solucionar problemas no ASP.NET Core no Serviço de Aplicativo do Azure](#).

## Configurar

- Abra uma [conta do Azure gratuita](#) se você não tiver uma.

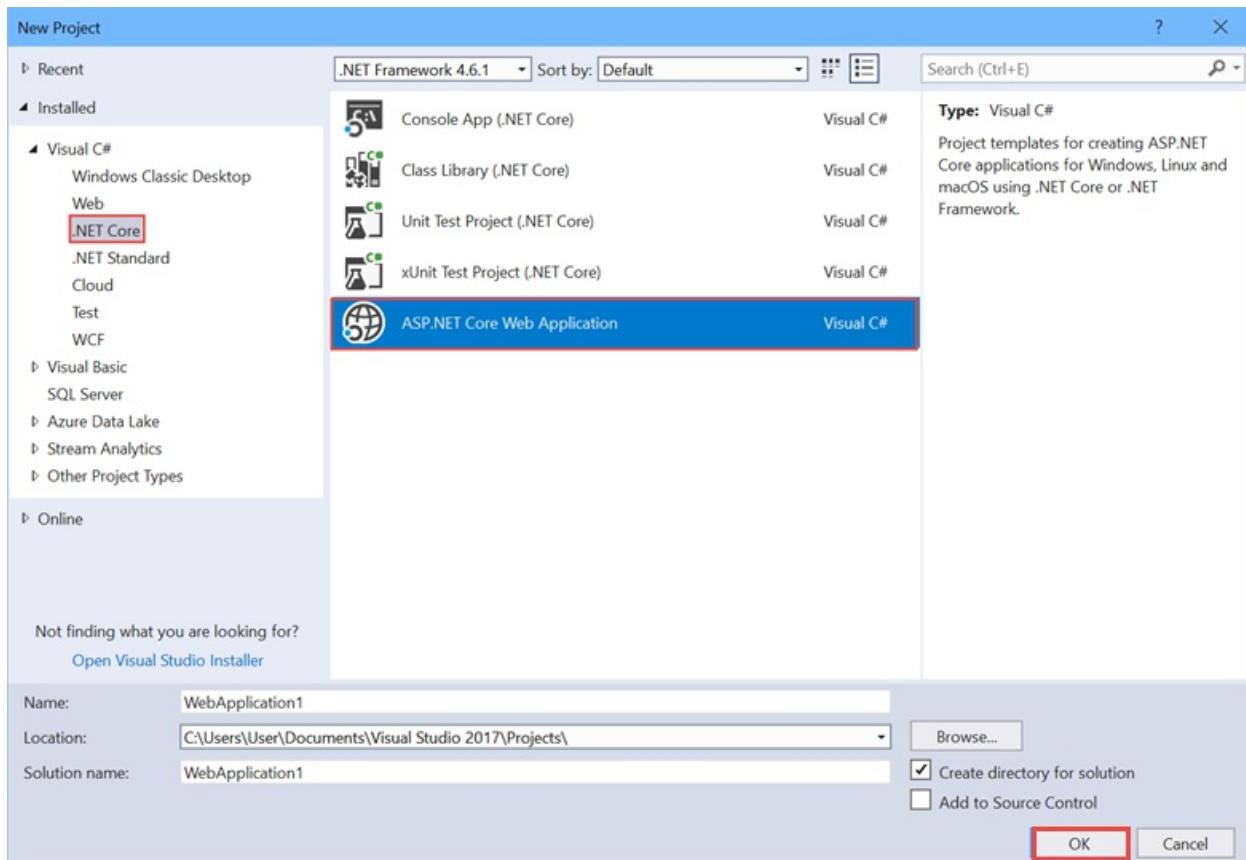
## Como criar um aplicativo Web

Na página inicial do Visual Studio, selecione **Arquivo > Novo > Projeto...**



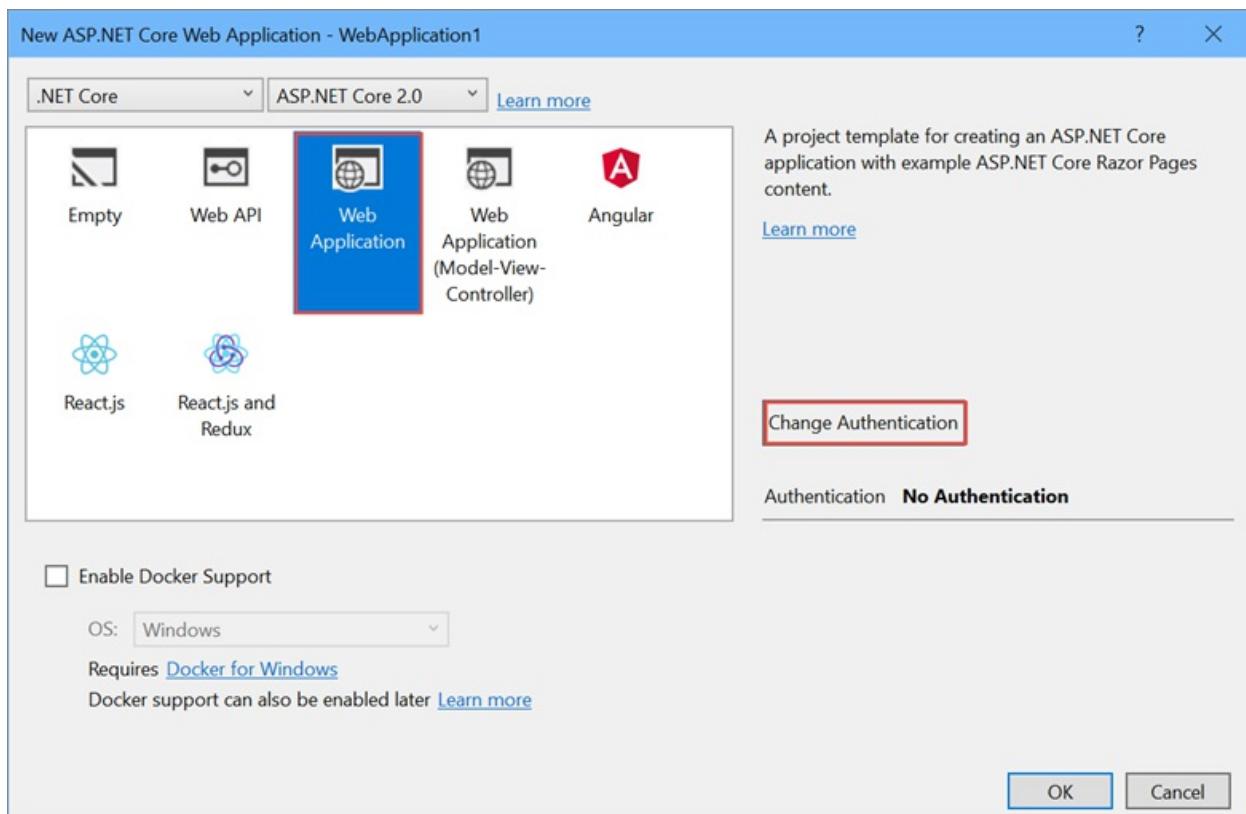
Complete a caixa de diálogo **Novo Projeto**:

- No painel esquerdo, selecione **.NET Core**.
- No painel central, toque em **Aplicativo Web ASP.NET Core**.
- Selecione **OK**.



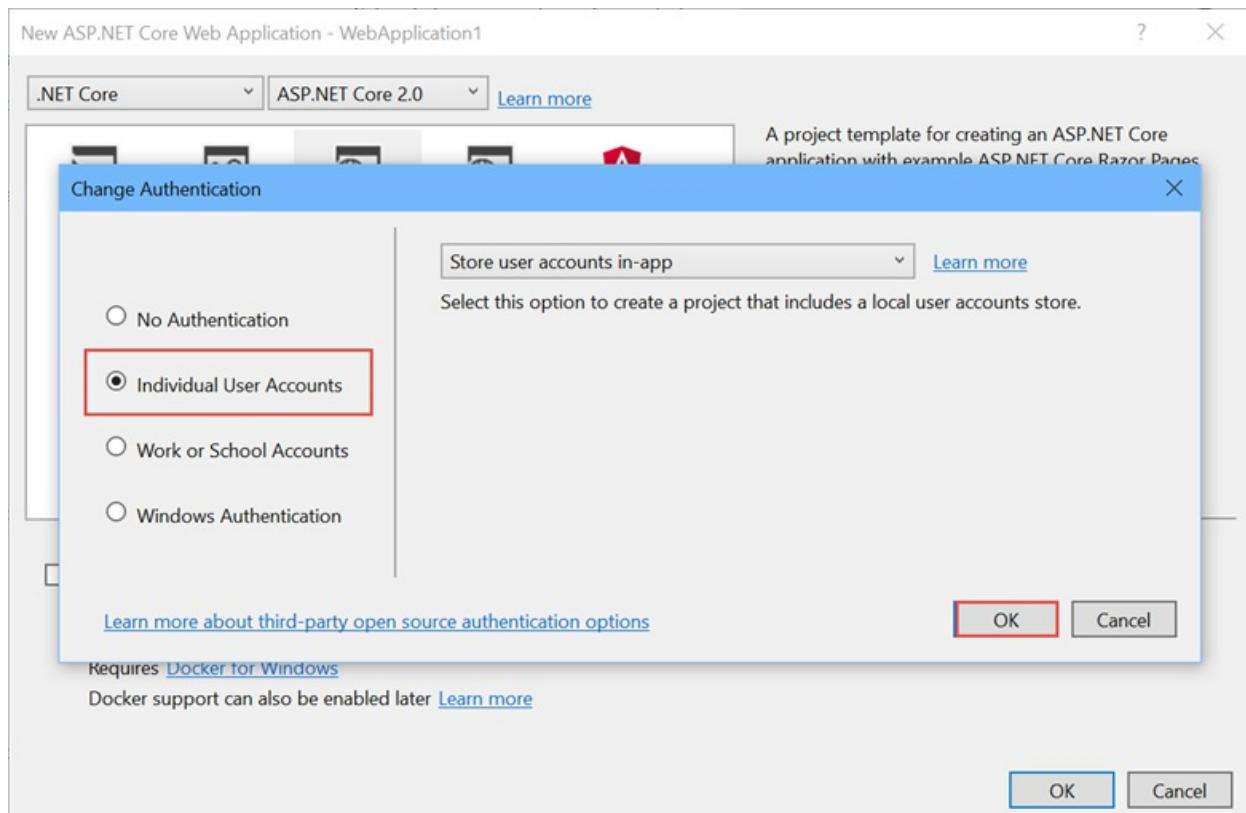
Na caixa de diálogo **Novo Aplicativo Web ASP.NET Core**:

- Selecione **Aplicativo Web**.
- Selecione **Mudar Autenticação**.



A caixa de diálogo **Mudar Autenticação** é exibida.

- Selecione **Contas de Usuário Individuais**.
- Selecione **OK** para retornar para o **Novo Aplicativo Web do ASP.NET Core**, em seguida, selecione **OK** novamente.



O Visual Studio cria a solução.

## Executar o aplicativo

- Pressione CTRL+F5 para executar o projeto.
- Teste os links **Sobre** e **Contato**.

The screenshot shows the ASP.NET Core documentation website at localhost:44349. At the top, there's a navigation bar with links for 'WebApplication1', 'Home', 'About', 'Contact', 'Register' (which is highlighted with a yellow box), and 'Log in'. Below the navigation is a large banner with the text 'ASP.NET Core | Windows | Linux | OSX'. The banner features a left arrow, a right arrow, and a 'Learn how to build ASP.NET apps that can run anywhere.' section with a 'Learn More' button and three circular progress indicators.

**Application uses**

- Sample pages using ASP.NET Core Razor Pages
- Bower for managing client-side libraries
- Theming using Bootstrap

**How to**

- Working with Razor Pages.
- Manage User Secrets using Secret Manager.
- Use logging to log a message.
- Add packages using NuGet.
- Add client packages using Bower.
- Target development, staging or production environment.

**Overview**

- Conceptual overview of what is ASP.NET Core
- Fundamentals of ASP.NET Core such as Startup and middleware.
- Working with Data
- Security
- Client side development
- Develop on different platforms
- Read more on the documentation site

**Run & Deploy**

- Run your app
- Run tools such as EF migrations and more
- Publish to Microsoft Azure App Service

## Registrar um usuário

- Selecione **Registrar** e registre um novo usuário. Você pode usar um endereço de email fictício. Ao enviar, a página exibirá o seguinte erro:

"*Erro Interno do Servidor: uma operação de banco de dados falhou ao processar a solicitação. Exceção do SQL: não é possível abrir o banco de dados. A aplicação de migrações existentes ao contexto do BD do Aplicativo pode resolver esse problema.*"
- Selecione **Aplicar Migrações** e, depois que a página for atualizada, atualize a página.

The screenshot shows an 'Internal Server Error' page from localhost:16841/Account/Register. The error message is 'A database operation failed while processing the request.' It includes the following text:

SqlException: Cannot open database "aspnet-WebApplication1-3caf68c5-2764-4579-a54b-15b98365379b" requested by the login. The login failed. Login failed for user 'D\user'.

Applying existing migrations for ApplicationDbContext may resolve this issue

There are migrations for ApplicationDbContext that have not been applied to the database

- 0000000000000000\_CreatelentitySchema

**Apply Migrations**

In Visual Studio, you can use the Package Manager Console to apply pending migrations to the database:

PM> Update-Database

Alternatively, you can apply pending migrations from a command prompt at your project directory:

> dotnet ef database update

O aplicativo exibe o email usado para registrar o novo usuário e um link **Fazer logout**.

The screenshot shows a web browser window with the title "Home page - WebApplic" and the URL "https://localhost:44323". The page content is from "WebApplication1". At the top right, there is a yellow box highlighting the greeting "Hello user@example.com!". Below the header, there is a banner for Microsoft Azure with the text: "Microsoft Azure | Learn how Microsoft's Azure cloud platform allows you to build, deploy, and scale web apps." It includes icons for a gear, a network, a database, and a grid. A "Learn More" button is present. The main content area has a section titled "Application uses" with a bulleted list: "Sample pages using ASP.NET Core Razor Pages" and "Theming using Bootstrap". Another section titled "How to" lists: "Working with Razor Pages.", "Manage User Secrets using Secret Manager.", "Use logging to log a message.", and "Add packages using NuGet".

## Application uses

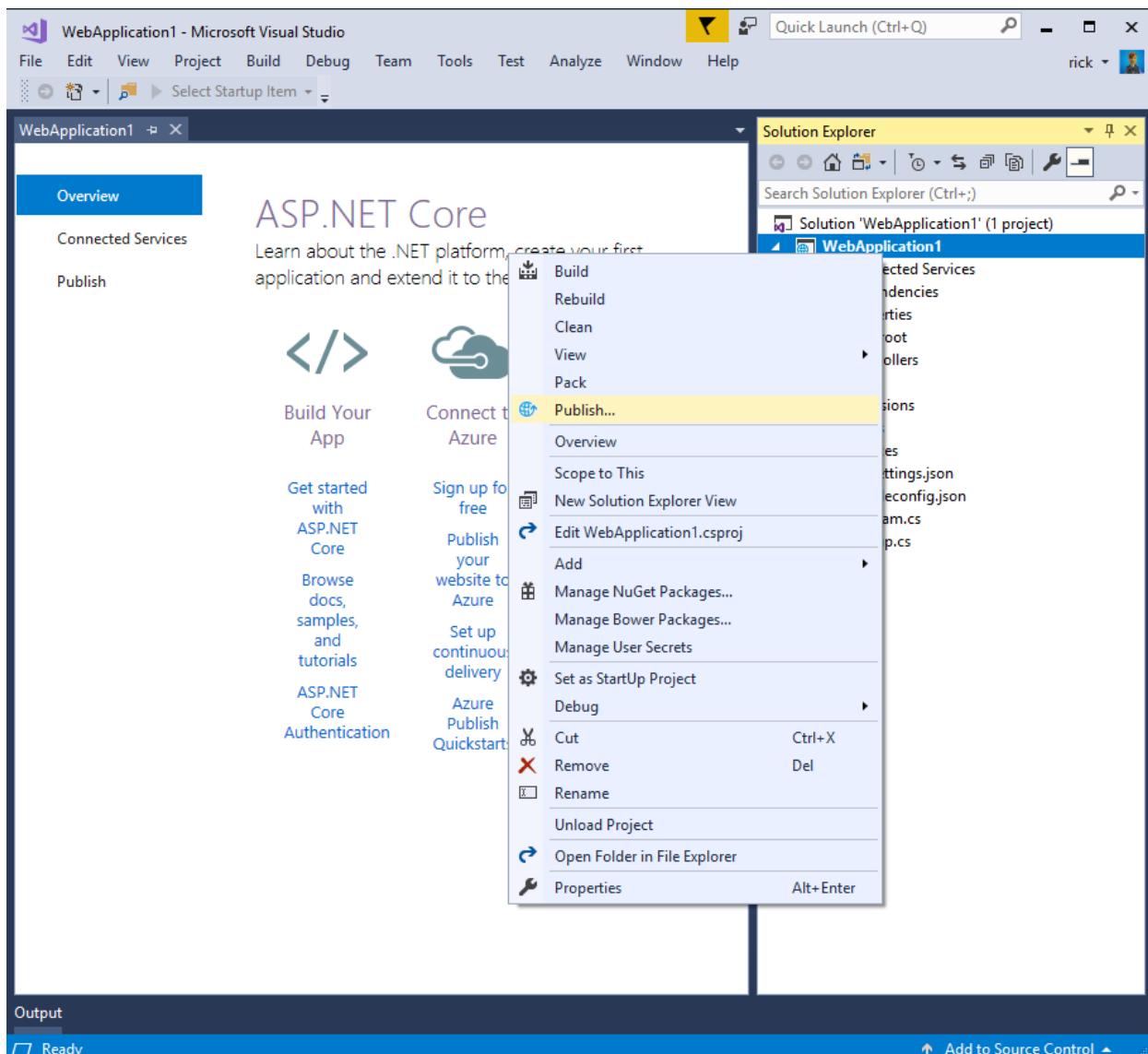
- Sample pages using ASP.NET Core Razor Pages
- Theming using Bootstrap

## How to

- Working with Razor Pages.
- Manage User Secrets using Secret Manager.
- Use logging to log a message.
- Add packages using NuGet

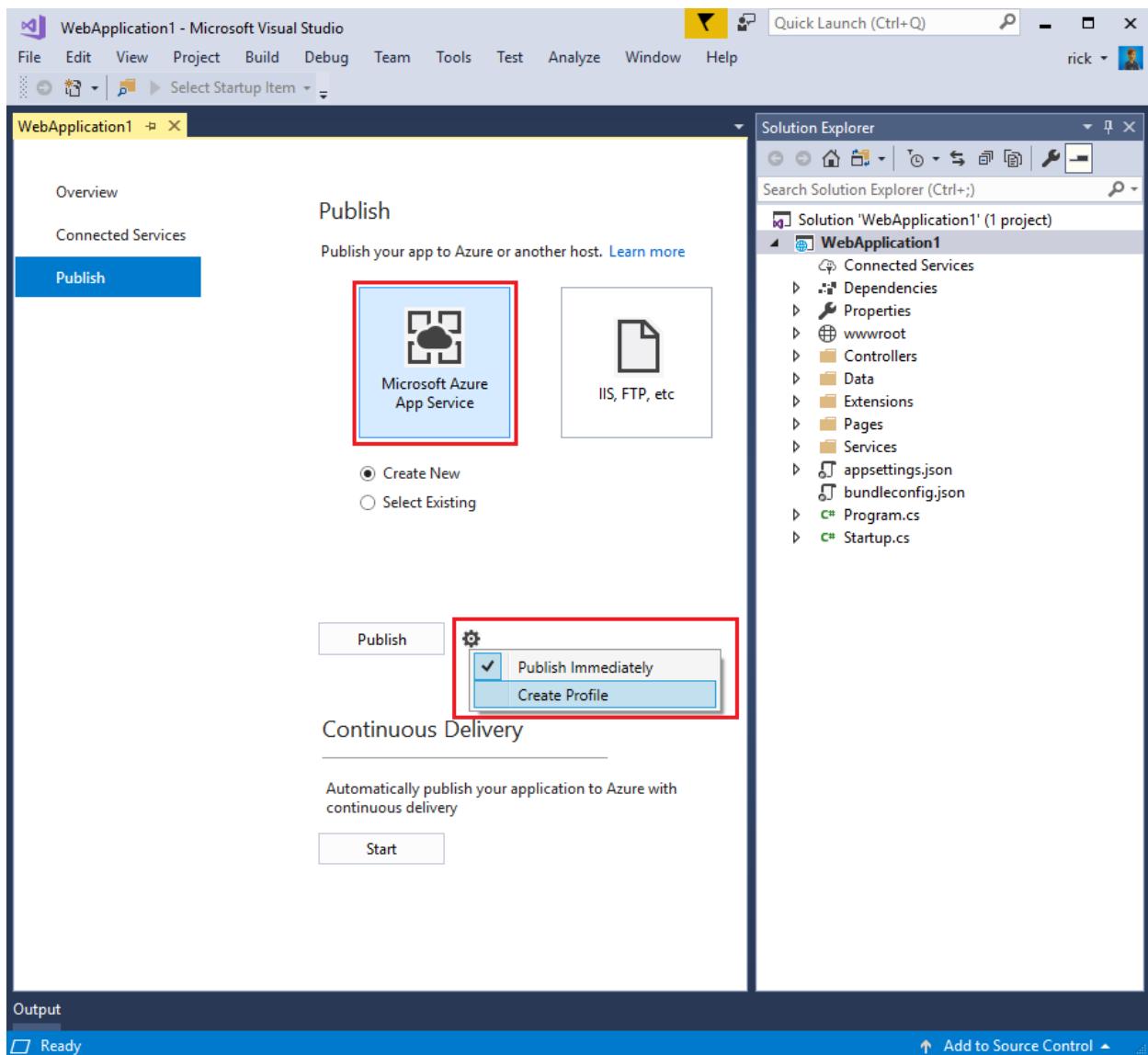
## Implantar o aplicativo no Azure

Clique com o botão direito do mouse no projeto no Gerenciador de Soluções e selecione **Publicar...**



Na caixa de diálogo **Publicar**:

- Selecione **Serviço de Aplicativo do Microsoft Azure**.
- Selecione o ícone de engrenagem e, em seguida, **Criar Perfil**.
- Selecione **Criar Perfil**.



## Criar recursos do Azure

A caixa de diálogo **Criar Serviço de Aplicativo** será exibida:

- Insira sua assinatura.
- Os campos de entrada **Nome do Aplicativo**, **Grupo de Recursos** e **Plano do Serviço de Aplicativo** serão populados. Você pode manter esses nomes ou alterá-los.

**Create App Service**

Host your web and mobile applications, REST APIs, and more in Azure

Microsoft account  
a.ricka00@gmail.com

**Hosting** i   **Services**  

App Name: WebApplication120171215025005

Change Type ▾

Subscription: MSDN

Resource Group: WebApplication120171215025005ResourceGroup\* New... i

App Service Plan: WebApplication120171215025005Plan\* New...

**Clicking the Create button will create the following Azure resources**

[Explore additional Azure services](#)

App Service - WebApplication120171215025005

App Service Plan - WebApplication120171215025005Plan

If you have removed your spending limit or you are using Pay as You Go, there may be monetary impact if you provision additional resources.

[Learn More](#)

Export... Create Cancel

- Selecione a guia **Serviços** para criar um novo banco de dados.
- Selecione o ícone verde + para criar um novo Banco de Dados SQL

**Create App Service**

Host your web and mobile applications, REST APIs, and more in Azure

Microsoft account  
a.ricka00@gmail.com

**Hosting** i   **Services**  

Select any additional Azure resources your app will need

Show: Recommended ▾

**Resource Type**

 **SQL Database**  
Scalable and managed database service for modern business-class apps

**Resources you've selected and configured**

**Resource Type**

 **WebApplication120171215025005Plan** ⚙️ X  
App Service Plan

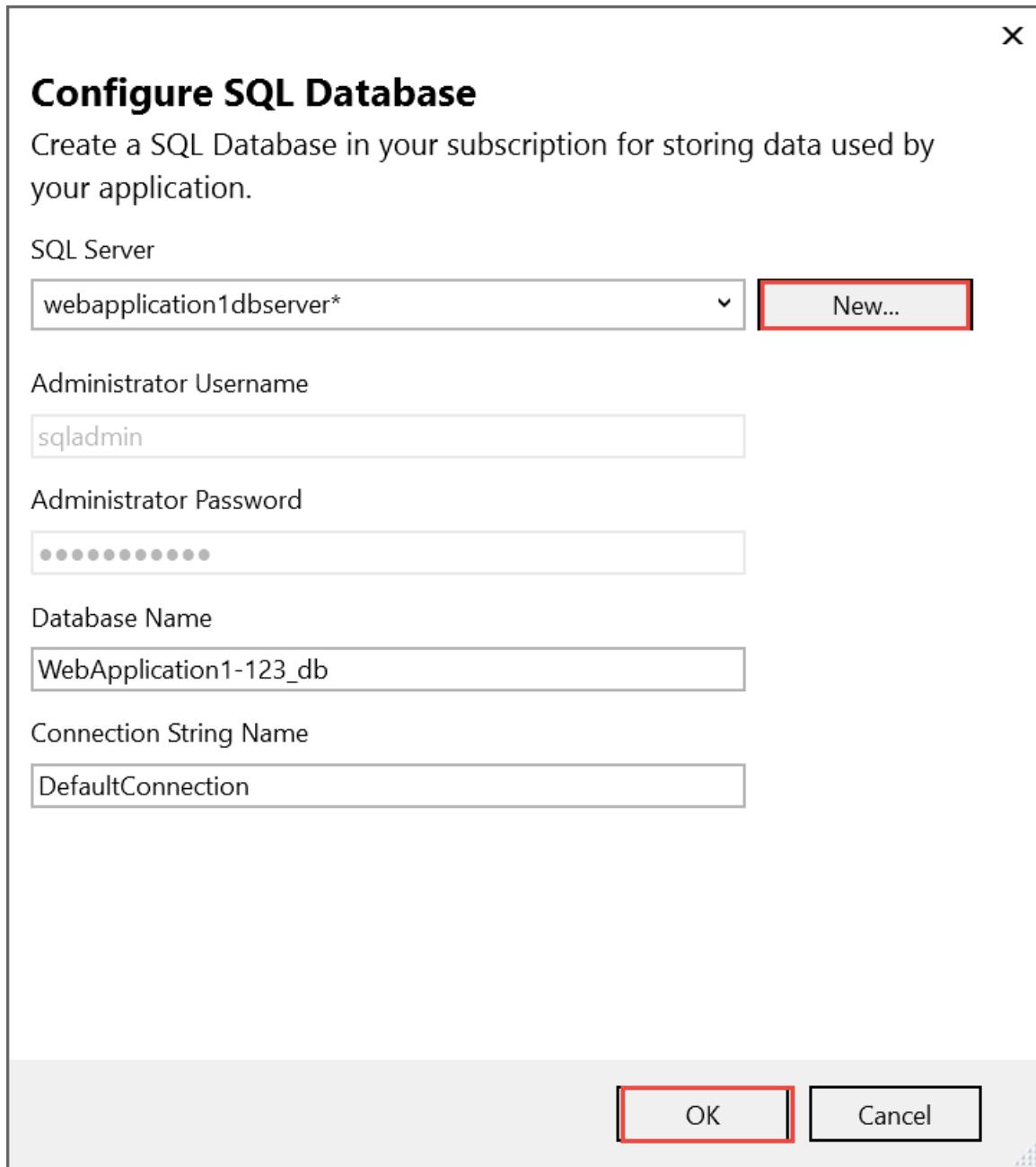
If you have removed your spending limit or you are using Pay as You Go, there may be monetary impact if you provision additional resources.

[Learn More](#)

Export... Create Cancel

- Selecione **Novo...** na caixa de diálogo **Configurar Banco de Dados SQL** para criar um novo banco de

dados.

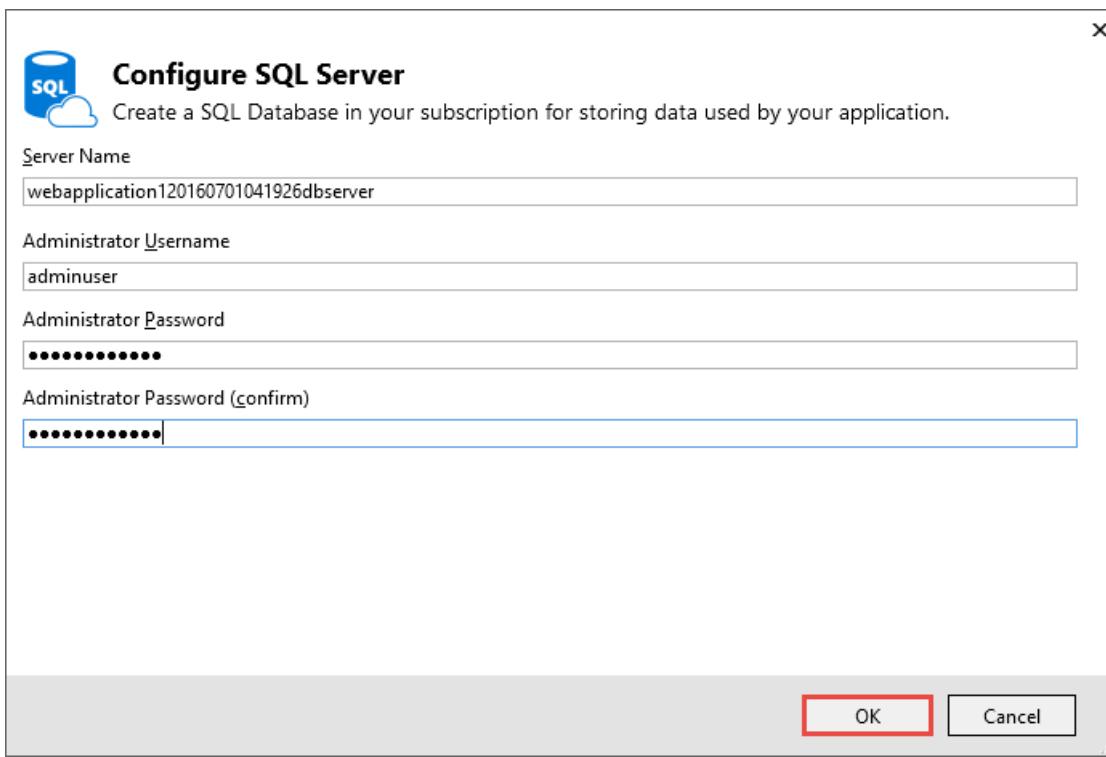


A caixa de diálogo **Configurar o SQL Server** é exibida.

- Insira um nome de usuário do administrador e a senha e, em seguida, selecione **OK**. Você pode manter o **Nome do Servidor** padrão.

**NOTE**

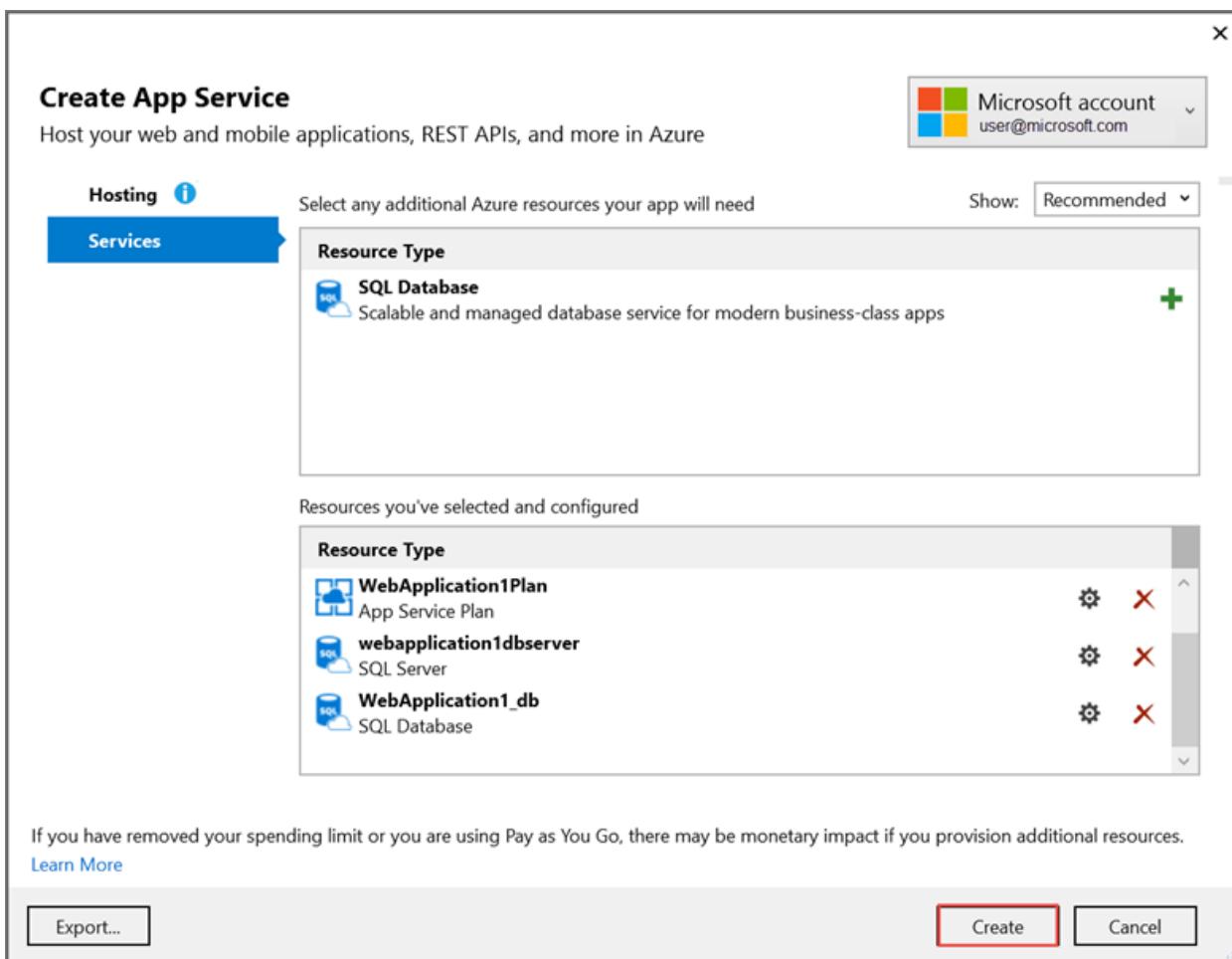
"admin" não é permitido como o nome de usuário administrador.



- Selecione **OK**.

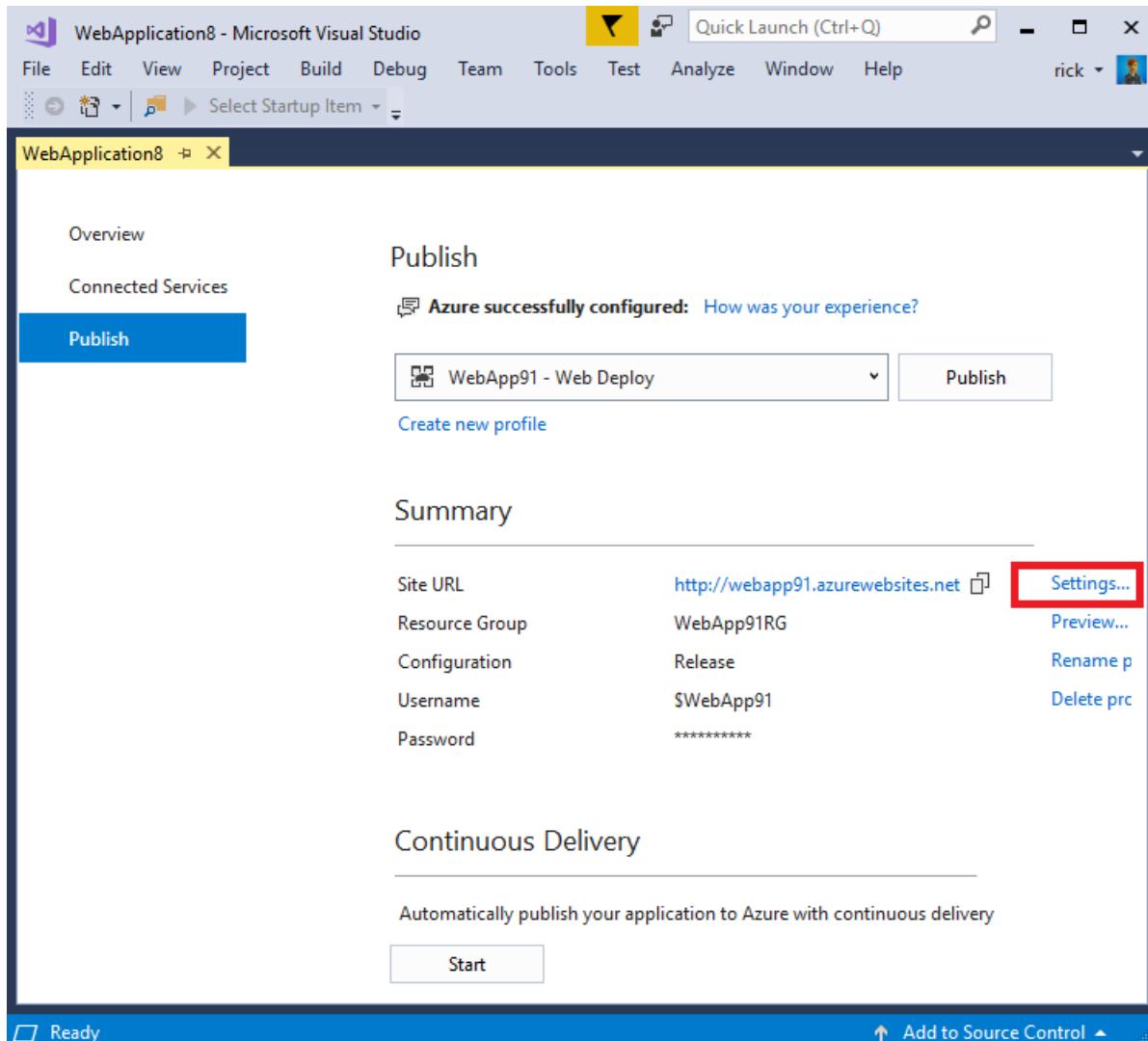
O Visual Studio retorna para a caixa de diálogo **Criar Serviço de Aplicativo**.

- Selecione **Criar** na caixa de diálogo **Criar Serviço de Aplicativo**.



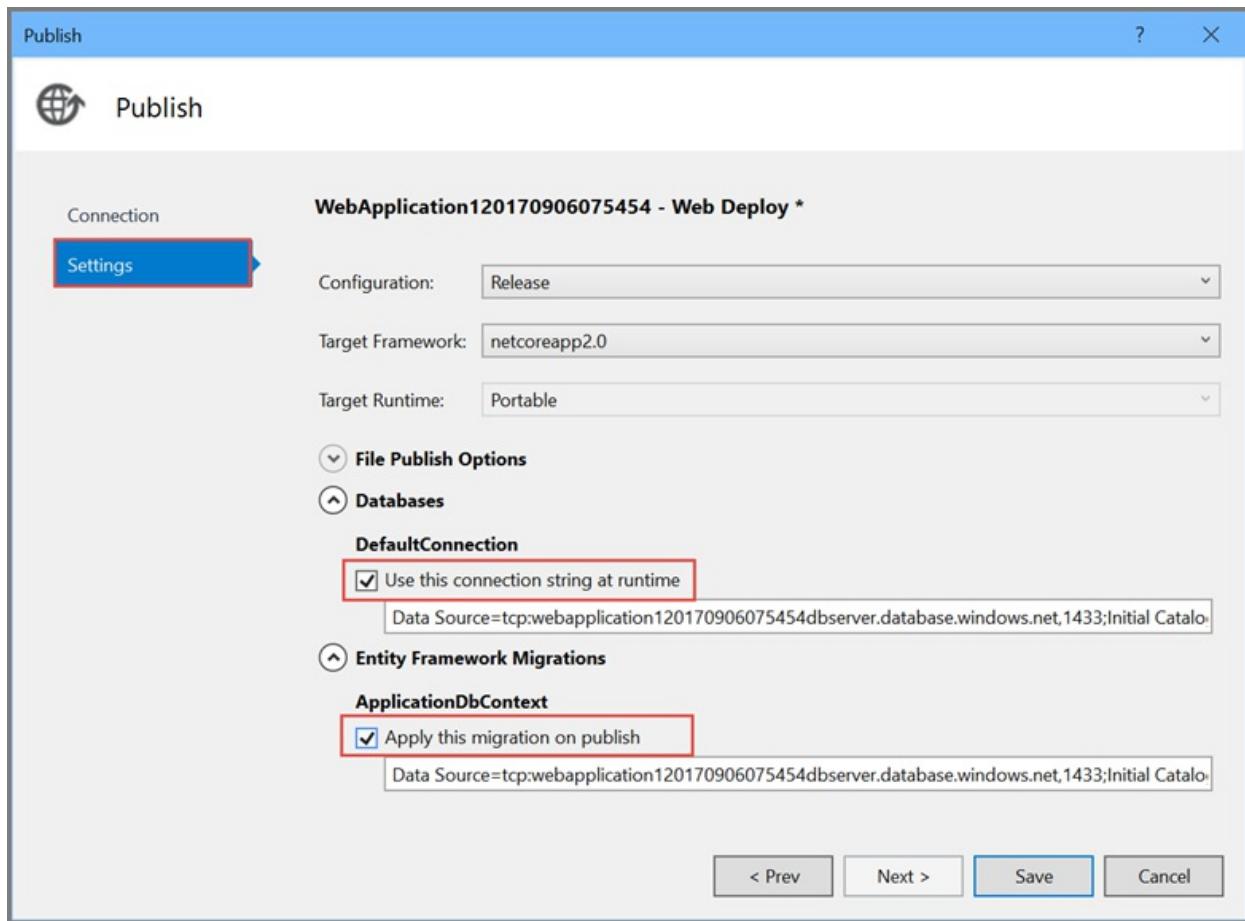
O Visual Studio cria o aplicativo Web e o SQL Server no Azure. Esta etapa pode levar alguns minutos. Para obter informações sobre os recursos criados, confira [Recursos adicionais](#).

Quando a implantação for concluída, selecione **Configurações**:



Na página **Configurações** da caixa de diálogo **Publicar**:

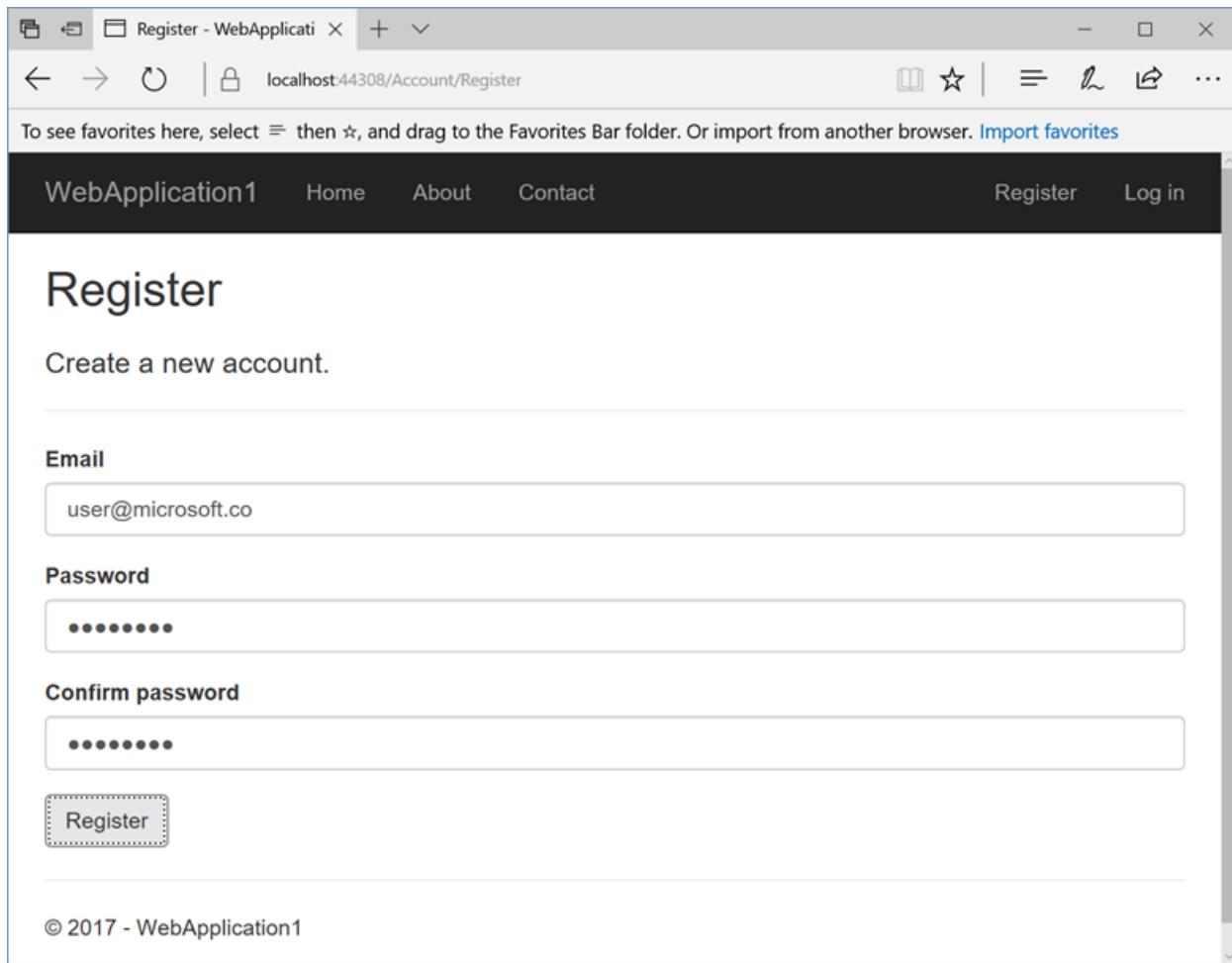
- Expanda **Bancos de Dados** e marque a opção **Usar esta cadeia de conexão no tempo de execução**.
- Expanda **Migrações do Entity Framework** e marque a opção **Aplicar esta migração durante a publicação**.
- Selecione **Salvar**. O Visual Studio retorna para a caixa de diálogo **Publicar**.



Clique em **Publicar**. O Visual Studio publica seu aplicativo no Azure. Quando a implantação for concluída, o aplicativo será aberto em um navegador.

#### Testar o aplicativo no Azure

- Teste os links **Sobre** e **Contato**
- Registrar um novo usuário



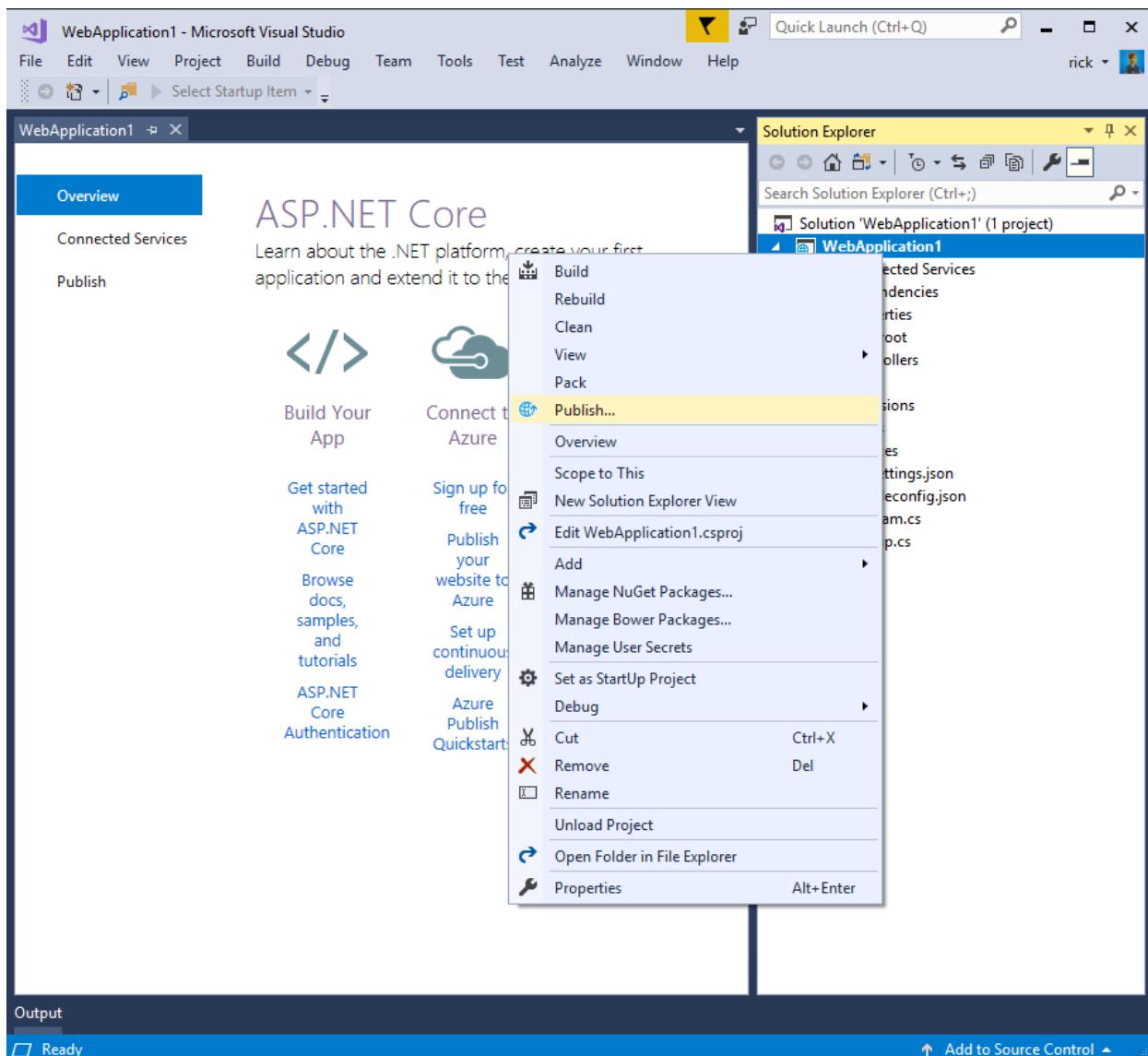
## Atualizar o aplicativo

- Edite a página do Razor *Pages/About.cshtml* e altere seu conteúdo. Por exemplo, você pode modificar o parágrafo para dizer "Hello ASP.NET Core!":

```
@page
@model AboutModel
 @{
    ViewData["Title"] = "About";
}
<h2>@ViewData["Title"]</h2>
<h3>@Model.Message</h3>

<p>Hello ASP.NET Core!</p>
```

- Clique com o botão direito do mouse no projeto e selecione **Publicar...** novamente.



- Depois que o aplicativo for publicado, verifique se as alterações feitas estão disponíveis no Azure.

To see favorites here, select then , and drag to the Favorites Bar folder. Or import from another browser. [Import favorites](#)

WebApplication1 Home About Contact Register Log in

# About

Your application description page.

Hello ASP.NET Core!

© 2017 - WebApplication2

## Limpar

Quando você concluir o teste do aplicativo, acesse o [portal do Azure](#) e exclua o aplicativo.

- Selecione **Grupos de recursos** e, em seguida, selecione o grupo de recursos criado.

Resource groups

Subscriptions: All 2 selected

NAME
WebApplication120160701041926

- Na página **Grupos de recursos**, selecione **Excluir**.

The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes the title 'Settings - Microsoft Azu', a back arrow, forward arrow, refresh icon, a lock icon, the URL 'portal.azure.com/#resource/subscript', a search icon, a bell icon, a pencil icon, a gear icon, a smiley face icon, a question mark icon, and a user profile icon. Below the navigation bar is the Microsoft Azure logo and a toolbar with icons for settings, add, columns, and refresh. The main content area is titled 'Resource groups' and shows a list of resource groups. One group, 'WebApplication120160701041926', is selected and its details are displayed on the right. The 'Delete' button for this group is highlighted with a red box. The 'Essentials' section for the selected group shows the following information:

Subscription name	Subscription
Visual Studio Ultimate with MSDN	Subscriptions
Last deployment	Location
7/1/2016 (Succeeded)	West U...

- Insira o nome do grupo de recursos e selecione **Excluir**. O aplicativo e todos os outros recursos criados neste tutorial agora foram excluídos do Azure.

#### Próximas etapas

- [Implantação contínua no Azure com o Visual Studio e o GIT com o ASP.NET Core](#)

## Recursos adicionais

- [Serviço de Aplicativo do Azure](#)
- [Grupo de recursos do Azure](#)
- [Banco de Dados SQL do Azure](#)
- [Perfis de publicação do Visual Studio para a implantação do aplicativo ASP.NET Core](#)
- [Solucionar problemas no ASP.NET Core no Serviço de Aplicativo do Azure](#)

# Implantação contínua no Azure com o Visual Studio e o GIT com o ASP.NET Core

10/01/2019 • 13 minutes to read • [Edit Online](#)

Por Erik Reitan

## IMPORTANT

### Versões prévias do ASP.NET Core com o Serviço de Aplicativo do Azure

Versões prévias do ASP.NET Core não são implantadas para o Serviço de Aplicativo do Azure por padrão. Para hospedar um aplicativo que usa uma versão prévia do ASP.NET Core, veja [Implantar versão prévia do ASP.NET Core para o Serviço de Aplicativo do Azure](#).

Este tutorial mostra como criar um aplicativo Web ASP.NET Core usando o Visual Studio e implantá-lo por meio do Visual Studio no Serviço de Aplicativo do Azure usando a implantação contínua.

Consulte também [Criar seu primeiro pipeline com o Azure Pipelines](#), que mostra como configurar um fluxo de trabalho de CD (entrega contínua) para o [Serviço de Aplicativo do Azure](#) usando o Azure DevOps Services. O Azure Pipelines (um serviço do Azure DevOps Services) simplifica a configuração de um pipeline de implantação robusta para publicar atualizações para aplicativos hospedados no Serviço de Aplicativo do Azure. O pipeline pode ser configurado no portal do Azure para criar, executar testes, implantar em um slot de preparo e, em seguida, implantar na produção.

## NOTE

Para concluir este tutorial, você precisa de uma conta do Microsoft Azure. Para obter uma conta, [ative os benefícios do assinante do MSDN](#) ou [inscreva-se em uma avaliação gratuita](#).

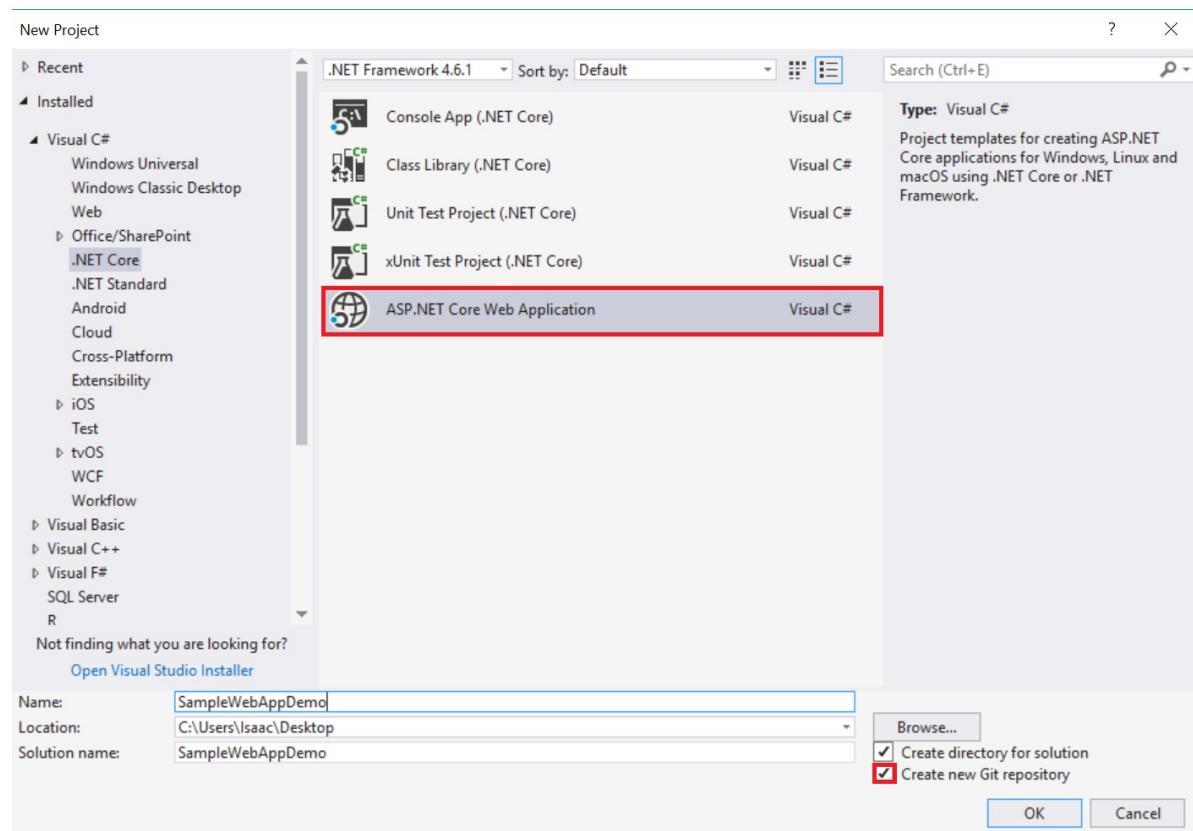
## Pré-requisitos

Este tutorial pressupõe que o seguinte software está instalado:

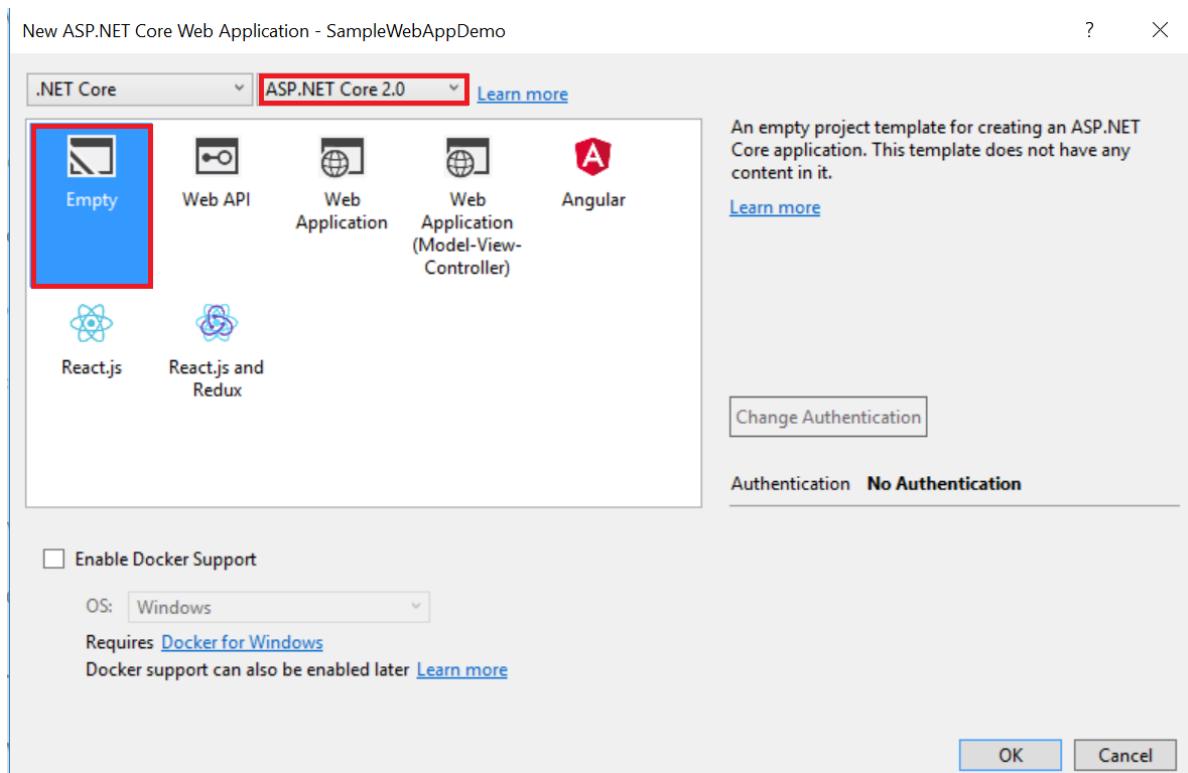
- [Visual Studio](#)
- [SDK 2.0 ou posterior do .NET Core](#)
- [Git para Windows](#)

## Criar um aplicativo Web ASP.NET Core

1. Inicie o Visual Studio.
2. No menu **Arquivo**, selecione **Novo > Projeto**.
3. Selecione o modelo de projeto **Aplicativo Web ASP.NET Core**. Ele será exibido em **Instalado > Modelos > Visual C# > .NET Core**. Nomeie o projeto `SampleWebAppDemo`. Selecione a opção **Criar novo repositório GIT** e clique em **OK**.



4. Na caixa de diálogo **Novo Projeto ASP.NET Core**, selecione o modelo **Vazio** do ASP.NET Core e clique em **OK**.



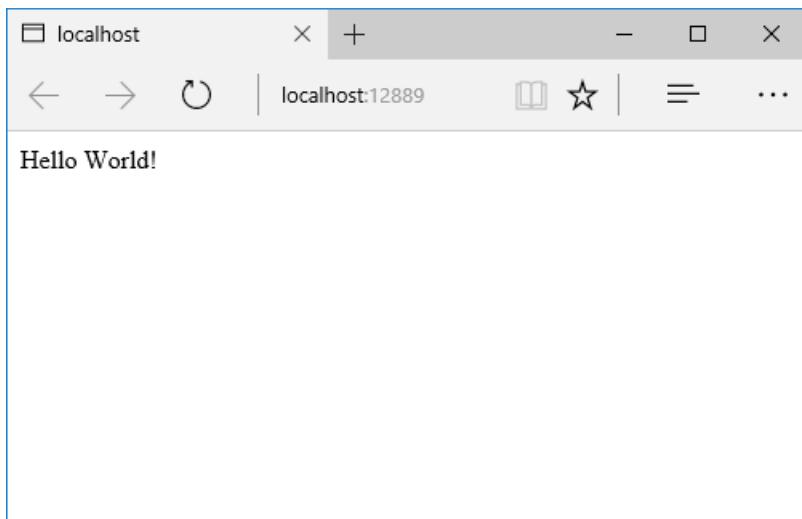
#### NOTE

A versão mais recente do .NET Core é a 2.0.

### Executando o aplicativo Web localmente

1. Depois que o Visual Studio concluir a criação do aplicativo, execute o aplicativo selecionando **Depurar > Iniciar Depuração**. Como alternativa, pressione **F5**.

Talvez seja necessário alguns instantes para inicializar o Visual Studio e o novo aplicativo. Quando isso for concluído, o navegador mostrará o aplicativo em execução.

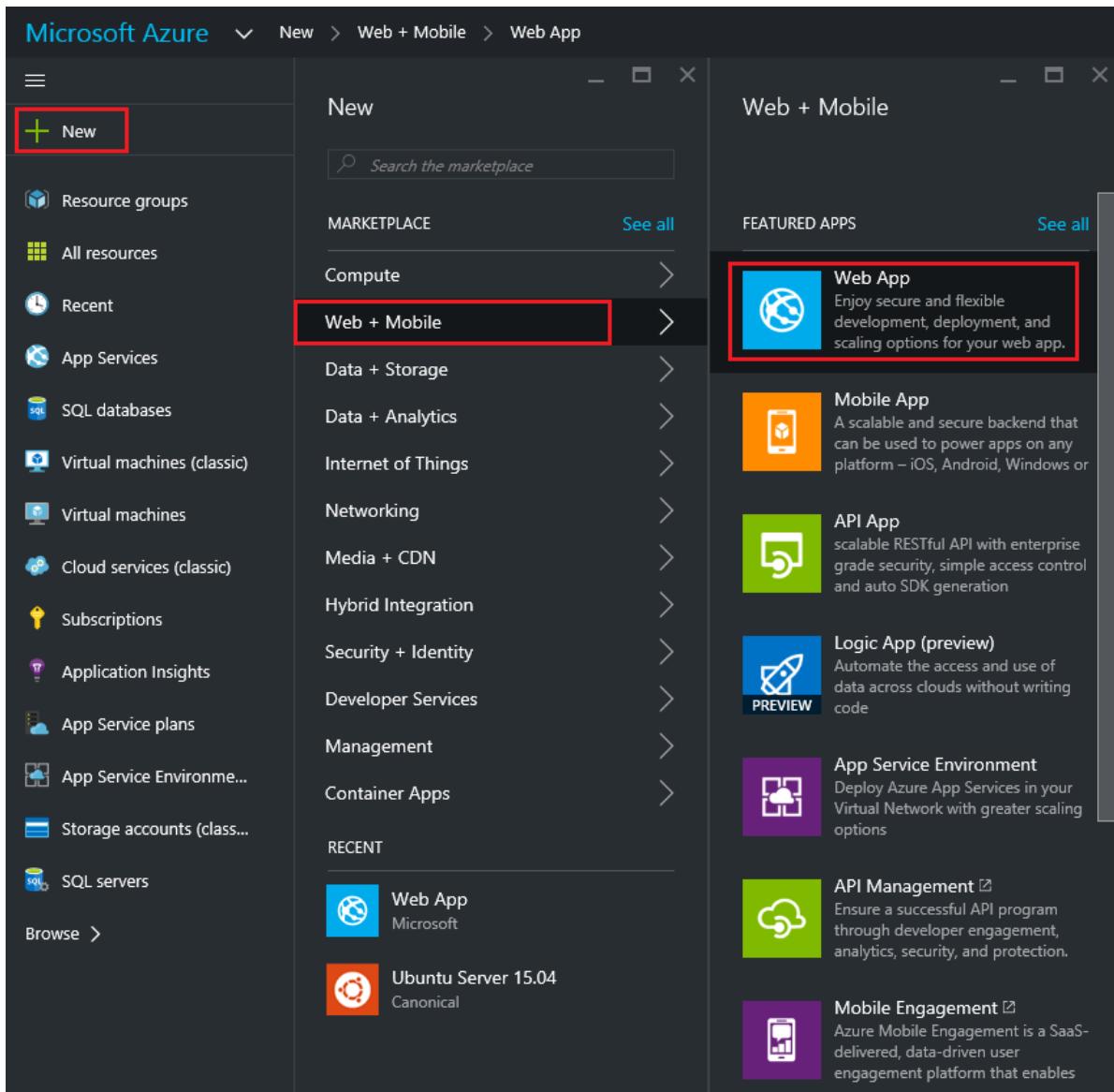


2. Depois de examinar o aplicativo Web em execução, feche o navegador e selecione o ícone "Parar Depuração" na barra de ferramentas do Visual Studio para interromper o aplicativo.

## Criar um aplicativo Web no Portal do Azure

As etapas a seguir criam um aplicativo Web no portal do Azure:

1. Faça logon no [portal do Azure](#).
2. Selecione **NOVO** na parte superior esquerda da interface do portal.
3. Selecione **Web + Celular > Aplicativo Web**.



4. Na folha **Aplicativo Web**, insira um valor exclusivo para o **Nome do Serviço de Aplicativo**.

Web App

\* App Service Name  
SampleWebAppDemo .azurewebsites.net

\* Subscription  
Visual Studio Ultimate with MSDN

\* Resource Group  
Default-Web-WestUS >  
[New](#)

\* App Service plan/Location  
Default0(West US) >

Pin to dashboard

**Create**

**NOTE**

O **Nome do Serviço de Aplicativo** precisa ser exclusivo. O portal impõe essa regra quando o nome é fornecido. Se você fornecer outro valor, você precisará substituí-lo para cada ocorrência do **SampleWebAppDemo** neste tutorial.

Também na folha **Aplicativo Web**, selecione um **Plano do Serviço de Aplicativo/Localização** existente ou crie um novo. Se você criar um novo plano, selecione o tipo de preço, o local e outras opções. Para obter mais informações sobre os planos do Serviço de Aplicativo, veja [Visão geral detalhada dos planos do Serviço de Aplicativo do Azure](#).

5. Selecione **Criar**. O Azure provisionará e iniciará o aplicativo Web.

Microsoft Azure SampleWebAppDemo01

SampleWebAppDemo01 Web app

Resource group: Default-Web-WestUS

Status: Running

Location: West US

Subscription name: Visual Studio Ultimate with MSDN

Subscription id: [redacted]

URL: http://samplewebappdemo01.azurewebsites...

App Service plan/pricing tier: Default0 (Free)

FTP/Deployment username: SampleWebAppDemo01\erikre01

FTP hostname: ftp://waws-prod-bay-005.ftp.azurewebsites...

FTPS hostname: ftps://waws-prod-bay-005.ftp.azurewebsites...

All settings →

## Habilitar a publicação do Git no novo aplicativo Web

O GIT é um sistema de controle de versão distribuída que pode ser usado para implantar um aplicativo Web do Serviço de Aplicativo do Azure. O código do aplicativo Web é armazenado em um repositório GIT local e implantado no Azure por push para um repositório remoto.

1. Faça logon no [portal do Azure](#).
2. Selecione **Serviços de Aplicativos** para exibir uma lista de serviços de aplicativos associados à assinatura do Azure.
3. Selecione o aplicativo Web criado na seção anterior deste tutorial.
4. Na folha **Implantação**, selecione **Opções de implantação > Escolher Origem > Repositório Git Local**.

Deployment option Set up deployment option

\* Choose Source Configure required settings

DEPLOYMENT

- Quickstart
- Deployment credentials
- Deployment slots
- Deployment options**
- Continuous Delivery (Preview)

Choose source

- Visual Studio Team Services By Microsoft
- OneDrive By Microsoft
- Local Git Repository** By Git
- GitHub By GitHub
- Bitbucket By Atlassian
- Dropbox By Dropbox
- External Repository

5. Selecione **OK**.
6. Caso você não tenha configurado anteriormente as credenciais de implantação para publicar um aplicativo Web ou outro aplicativo do Serviço de Aplicativo, configure-as agora:
  - Selecione **Configurações > Credenciais de implantação**. A folha **Definir credenciais de**

**implantação** é exibida.

- Crie um nome de usuário e uma senha. Salve a senha para uso posterior ao configurar o GIT.
  - Selecione **Salvar**.
7. Na folha **Aplicativo Web**, selecione **Configurações > Propriedades**. A URL do repositório GIT remoto no qual você implantará será mostrada em **URL do GIT**.
8. Copie o valor **URL do GIT** para uso posterior no tutorial.

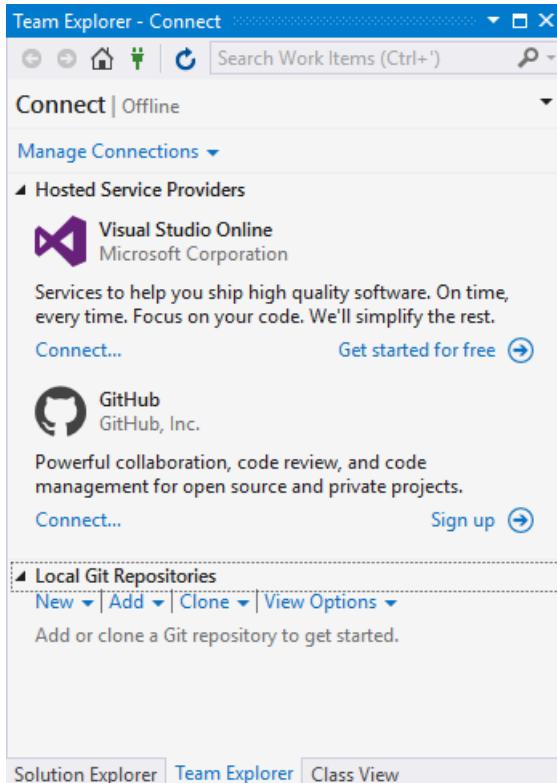
The screenshot shows the 'Properties' blade for an Azure App Service named 'samplewebappdemo01'. The 'URL' field contains the value 'samplewebappdemo01.azurewebsites.net', which is highlighted with a red box. The 'GIT URL' field below it contains 'https://erikre01@samplewebappdemo01.' and is also highlighted with a red box. Other fields shown include 'STATUS' (Running), 'VIRTUAL IP ADDRESS' (No IP-based SSL binding is configured), 'MODE' (Free), 'OUTBOUND IP ADDRESSES' (23.99.3.91,23.99.3.100,23.99.3.101,23.99.3), 'FTP/DEPLOYMENT USER' (SampleWebAppDemo01\erikre01), 'FTP HOST NAME' (ftp://waws-prod-bay-005.ftp.azurewebsites.net), 'FTP DIAGNOSTIC LOGS' (ftp://waws-prod-bay-005.ftp.azurewebsites.net), 'FTPS HOST NAME' (ftps://waws-prod-bay-005.ftp.azurewebsites.net), and 'FTPS DIAGNOSTIC LOGS' (ftps://waws-prod-bay-005.ftp.azurewebsites.net).

## Publicar o aplicativo Web no Serviço de Aplicativo do Azure

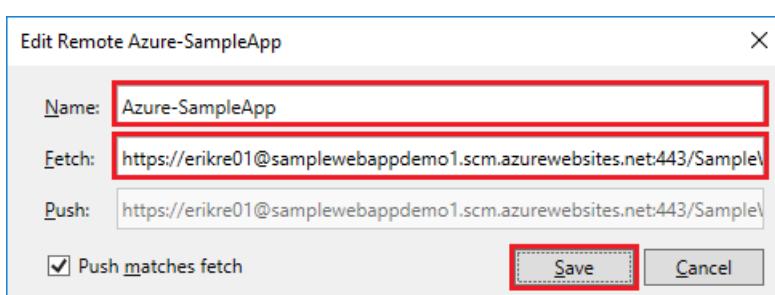
Nesta seção, você criará um repositório GIT local usando o Visual Studio e efetuará push desse repositório para o Azure para implantar o aplicativo Web. As etapas envolvidas incluem as seguintes:

- Adicione a configuração do repositório remoto usando o valor de URL do GIT, de modo que você possa implantar seu repositório local no Azure.
- Confirme as alterações do projeto.
- Envie as alterações do projeto por push do repositório local para o repositório remoto no Azure.

1. No **Gerenciador de Soluções**, clique com o botão direito do mouse em “**SampleWebAppDemo**” da **Solução** e selecione **Confirmar**. O **Team Explorer** é exibido.



2. No **Team Explorer**, selecione a **Página Inicial** (ícone da página inicial) > **Configurações** > **Configurações do Repositório**.
3. Na seção **Remotos** das **Configurações do Repositório**, selecione **Adicionar**. A caixa de diálogo **Adicionar Remoto** é exibida.
4. Defina o **Nome** do remoto como **Azure-SampleApp**.
5. Defina o valor de **Buscar** como a **URL do GIT** copiada do Azure anteriormente neste tutorial. Observe que essa é a URL que termina com **.git**.

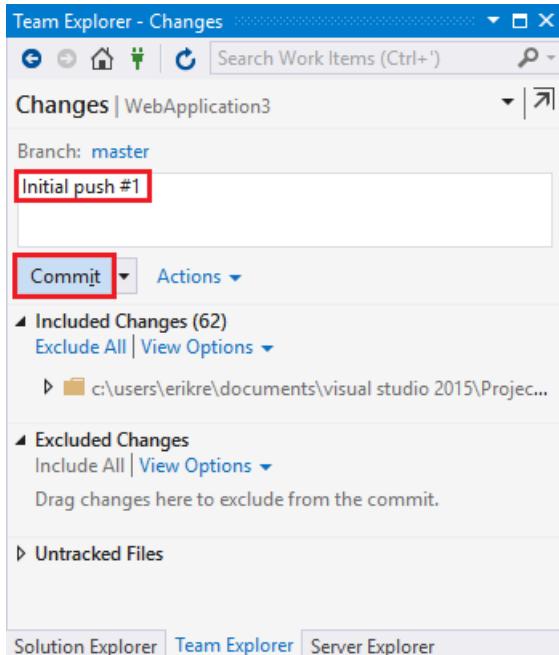


#### NOTE

Como alternativa, especifique o repositório remoto na **Janela Comando** abrindo a **Janela Comando**, alterando para o diretório do projeto e inserindo o comando. Exemplo:

```
git remote add Azure-SampleApp https://me@sampleapp.scm.azurewebsites.net:443/SampleApp.git
```

6. Selecione a **Página Inicial** (ícone da página inicial) > **Configurações** > **Configurações Globais**. Confirme se o nome e o endereço de email estão definidos. Se necessário, selecione **Atualizar**.
7. Selecione **Página Inicial** > **Alterações** para retornar à exibição **Alterações**.
8. Escreva uma mensagem de confirmação, como **Push Inicial nº 1** e selecione **Confirmar**. Essa ação cria uma *confirmação* localmente.



#### NOTE

Como alternativa, confirme as alterações na **Janela Comando** abrindo a **Janela Comando**, alterando para o diretório do projeto e inserindo os comandos do GIT. Exemplo:

```
git add .
git commit -am "Initial Push #1"
```

9. Selecione **Página Inicial** > **Sincronização** > **Ações** > **Abrir Prompt de Comando**. O prompt de comando se abre no diretório do projeto.

10. Insira o seguinte comando na janela de comando:

```
git push -u Azure-SampleApp master
```

11. Insira a senha das **credenciais de implantação** do Azure criada anteriormente no Azure.

Esse comando inicia o processo de envio por push dos arquivos de projeto locais para o Azure. A saída do comando acima termina com uma mensagem informando que a implantação foi bem-sucedida.

```
remote: Finished successfully.
remote: Running post deployment command(s)...
remote: Deployment successful.
To https://username@samplewebappdemo01.scm.azurewebsites.net:443/SampleWebAppDemo01.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from Azure-SampleApp.
```

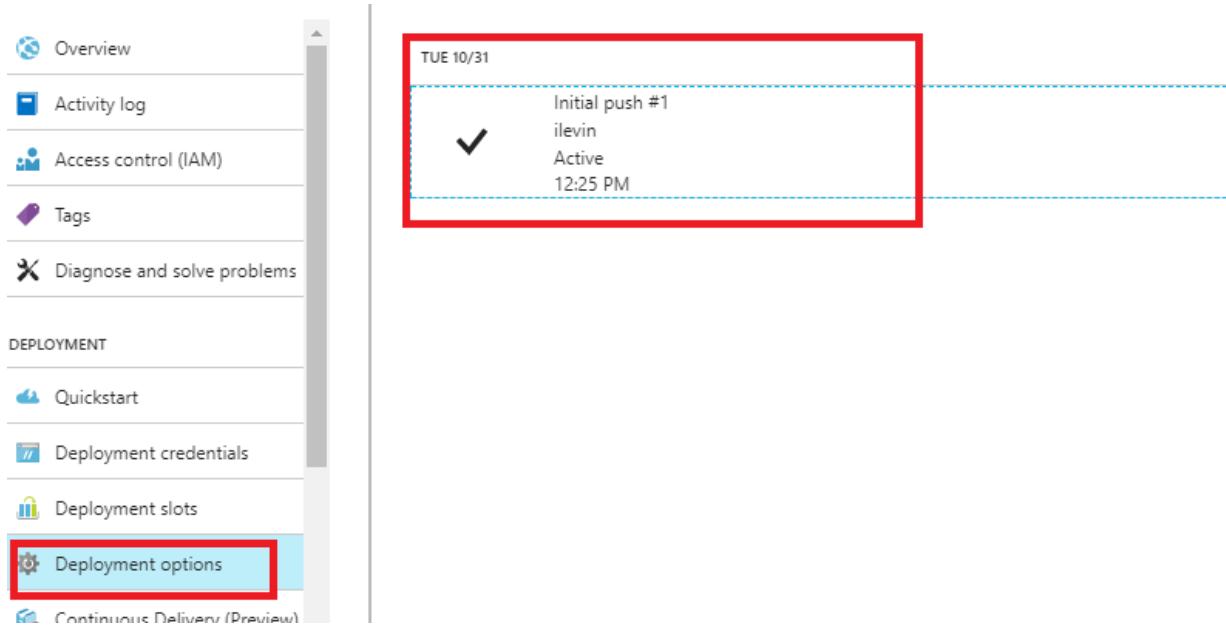
#### NOTE

Se a colaboração no projeto é necessária, considere a possibilidade de enviar por push para o [GitHub](#) antes de enviar por push para o Azure.

### Verificar a implantação ativa

Verifique se a transferência do aplicativo Web do ambiente local para o Azure é bem-sucedida.

No [portal do Azure](#), selecione o aplicativo Web. Selecione **Implantação > Opções de implantação**.



The screenshot shows the Azure portal's left navigation bar with various options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quickstart, Deployment credentials, Deployment slots, Deployment options (which is highlighted with a red box), and Continuous Delivery (Preview). The main content area displays deployment logs for a specific application. One log entry is highlighted with a red box: "Initial push #1" by "ilevin" at "12:25 PM" on "TUE 10/31". The log entry also indicates the deployment was "Active".

### Executar o aplicativo no Azure

Agora que o aplicativo Web é implantado no Azure, execute o aplicativo.

Isso pode ser feito de duas maneiras:

- No portal do Azure, localize a folha do aplicativo Web desse aplicativo Web. Selecione **Procurar** para exibir o aplicativo no navegador padrão.
- Abra um navegador e insira a URL para o aplicativo Web. Exemplo: <http://SampleWebAppDemo.azurewebsites.net>

### Atualize o aplicativo Web e publique-o novamente

Depois de fazer alterações ao código local, republique:

1. No **Gerenciador de Soluções** do Visual Studio, abra o arquivo *Startup.cs*.
2. No método `Configure`, modifique o método `Response.WriteAsync` para que ele seja exibido da seguinte maneira:

```
await context.Response.WriteAsync("Hello World! Deploy to Azure.");
```
3. Salve as alterações em *Startup.cs*.
4. No **Gerenciador de Soluções**, clique com o botão direito do mouse em “**SampleWebAppDemo**” da **Solução** e selecione **Confirmar**. O **Team Explorer** é exibido.
5. Escreva uma mensagem de confirmação, tal como `Update #2`.

6. Pressione o botão **Confirmar** para confirmar as alterações do projeto.
7. Selecione **Página Inicial > Sincronização > Ações > Enviar por Push.**

#### NOTE

Como alternativa, envie as alterações por push da **Janela Comando** abrindo a **Janela Comando**, alterando para o diretório do projeto e inserindo um comando do GIT. Exemplo:

```
git push -u Azure-SampleApp master
```

## Exibir o aplicativo Web atualizado no Azure

Exiba o aplicativo Web atualizado selecionando **Procurar** na folha do aplicativo Web no portal do Azure ou abrindo um navegador e inserindo a URL do aplicativo Web. Exemplo: <http://SampleWebAppDemo.azurewebsites.net>

## Recursos adicionais

- [Criar seu primeiro pipeline com o Azure Pipelines](#)
- [Kudu do projeto](#)
- [Perfis de publicação do Visual Studio para a implantação do aplicativo ASP.NET Core](#)

# Módulo do ASP.NET Core

30/01/2019 • 43 minutes to read • [Edit Online](#)

Por [Tom Dykstra](#), [Rick Strahl](#), [Chris Ross](#), [Rick Anderson](#), [Sourabh Shirhatti](#), [Justin Kotalik](#) e [Luke Latham](#)

O módulo do ASP.NET Core é um módulo nativo do IIS que se conecta ao pipeline do IIS para:

- Hoster um aplicativo ASP.NET Core dentro do processo de trabalho do IIS (`w3wp.exe`), chamado o [modelo de hospedagem no processo](#).
- Encaminhar solicitações da Web a um aplicativo ASP.NET Core de back-end que executa o [servidor Kestrel](#), chamado o [modelo de hospedagem de fora do processo](#).

Versões do Windows compatíveis:

- Windows 7 ou posterior
- Windows Server 2008 R2 ou posterior

Ao fazer uma hospedagem em processo, o módulo usa uma implementação de servidor em processo do IIS, chamado Servidor HTTP do IIS (`IISHttpServer`).

Ao hospedar de fora do processo, o módulo só funciona com o Kestrel. O módulo é incompatível com [HTTP.sys](#).

## Modelos de hospedagem

### Modelo de hospedagem em processo

Para configurar um aplicativo para hospedagem em processo, adicione a propriedade `<AspNetCoreHostingModel>` ao arquivo de projeto do aplicativo com um valor de `InProcess` (a hospedagem de fora do processo é definida com `OutOfProcess`):

```
<PropertyGroup>
  <AspNetCoreHostingModel>InProcess</AspNetCoreHostingModel>
</PropertyGroup>
```

Não há suporte para o modelo de hospedagem em processo para aplicativos ASP.NET Core direcionados ao .NET Framework.

Se a propriedade `<AspNetCoreHostingModel>` não estiver presente no arquivo, o valor padrão será `OutOfProcess`.

As seguintes características se aplicam ao hospedar em processo:

- O Servidor HTTP do IIS (`IISHttpServer`) é usado, em vez do servidor [Kestrel](#).
- O [requestTimeout atributo](#) não se aplica à hospedagem em processo.
- Não há suporte para o compartilhamento do pool de aplicativos entre aplicativos. Use um pool de aplicativos por aplicativo.
- Ao usar a [Implantação da Web](#) ou inserir manualmente um [arquivo app\\_offline.htm na implantação](#), o aplicativo talvez não seja desligado imediatamente se houver uma conexão aberta. Por exemplo, uma conexão websocket pode atrasar o desligamento do aplicativo.
- A arquitetura (número de bit) do aplicativo e o tempo de execução instalado (x64 ou x86) devem corresponder à arquitetura do pool de aplicativos.

- Se a configuração manual do host do aplicativo com `WebHostBuilder` (não usando `CreateDefaultBuilder`) e o aplicativo não forem executados diretamente no servidor Kestrel (auto-hospedado), chame `UseKestrel` antes de chamar `UseIISIntegration`. Se a ordem for invertida, a inicialização do host falhará.
- As desconexões do cliente são detectadas. O token de cancelamento `HttpContext.RequestAborted` é cancelado quando o cliente se desconecta.
- `GetCurrentDirectory` retorna o diretório de trabalho do processo iniciado pelo IIS em vez de o diretório do aplicativo (por exemplo, `C:\Windows\System32\inetsrv` para `w3wp.exe`).

Para obter o código de exemplo que define o diretório atual do aplicativo, confira a classe `CurrentDirectoryHelpers`. Chame o método `SetCurrentDirectory`. As chamadas seguintes a `GetCurrentDirectory` fornecem o diretório do aplicativo.

## Modelo de hospedagem de fora do processo

Para configurar um aplicativo para hospedagem fora do processo, use qualquer uma das abordagens a seguir no arquivo de projeto:

- não especifique a propriedade `<AspNetCoreHostingModel>`. Se a propriedade `<AspNetCoreHostingModel>` não estiver presente no arquivo, o valor padrão será `OutOfProcess`.
- Defina o valor da propriedade `<AspNetCoreHostingModel>` como `OutOfProcess` (a hospedagem no processo é definida com `InProcess`):

```
<PropertyGroup>
  <AspNetCoreHostingModel>OutOfProcess</AspNetCoreHostingModel>
</PropertyGroup>
```

O servidor `Kestrel` é usado, em vez do servidor HTTP do IIS (`IISHttpServer`).

## Alterações no modelo de hospedagem

Se a configuração `hostingModel` for alterada no arquivo `web.config` (explicado na seção [Configuração a Web.config](#)), o módulo reciclará o processo de trabalho do IIS.

Para o IIS Express, o módulo não recicla o processo de trabalho, mas em vez disso, dispara um desligamento normal do processo atual do IIS Express. A próxima solicitação para o aplicativo gera um novo processo do IIS Express.

## Nome do processo

`Process.GetCurrentProcess().ProcessName` relata `w3wp` / `iisexpress` (em processo) ou `dotnet` (fora do processo).

O Módulo do ASP.NET Core é um módulo nativo do IIS que se conecta ao pipeline do IIS para encaminhar solicitações da Web para aplicativos ASP.NET Core de back-end.

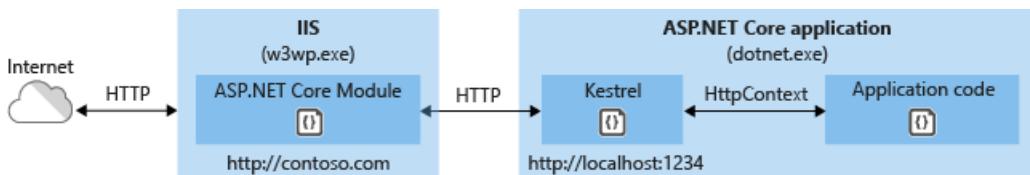
Versões do Windows compatíveis:

- Windows 7 ou posterior
- Windows Server 2008 R2 ou posterior

O módulo só funciona com o Kestrel. O módulo é incompatível com [HTTP.sys](#).

Como os aplicativos ASP.NET Core são executados em um processo separado do processo de trabalho do IIS, o módulo também realiza o gerenciamento de processos. O módulo inicia o processo para o aplicativo ASP.NET Core quando a primeira solicitação chega e reinicia o aplicativo quando ele falha. Isso é basicamente o mesmo comportamento que o dos aplicativos ASP.NET 4.x que são executados dentro do processo do IIS e são gerenciados pelo [WAS \(Serviço de Ativação de Processos do Windows\)](#).

O diagrama a seguir ilustra a relação entre o IIS, o Módulo do ASP.NET Core e um aplicativo:



As solicitações chegam da Web para o driver do HTTP.sys no modo kernel. O driver roteia as solicitações ao IIS na porta configurada do site, normalmente, a 80 (HTTP) ou a 443 (HTTPS). O módulo encaminha as solicitações ao Kestrel em uma porta aleatória do aplicativo, que não seja a porta 80 ou 443.

O módulo especifica a porta por meio de uma variável de ambiente na inicialização e o middleware de integração do IIS configura o servidor para escutar em `http://localhost:{port}`. Outras verificações são executadas e as solicitações que não se originam do módulo são rejeitadas. O módulo não é compatível com encaminhamento de HTTPS, portanto, as solicitações são encaminhadas por HTTP, mesmo se recebidas pelo IIS por HTTPS.

Depois que o Kestrel coleta a solicitação do módulo, a solicitação é enviada por push ao pipeline do middleware do ASP.NET Core. O pipeline do middleware manipula a solicitação e a passa como uma instância de `HttpContext` para a lógica do aplicativo. O middleware adicionado pela integração do IIS atualiza o esquema, o IP remoto e pathbase para encaminhar a solicitação para o Kestrel. A resposta do aplicativo é retornada ao IIS, que a retorna por push para o cliente HTTP que iniciou a solicitação.

Muitos módulos nativos, como a Autenticação do Windows, permanecem ativos. Para saber mais sobre módulos do IIS ativos com o Módulo do ASP.NET Core, confira [Módulos do IIS com o ASP.NET Core](#).

O Módulo do ASP.NET Core também pode:

- Definir variáveis de ambiente para o processo de trabalho.
- Registrar a saída StdOut no armazenamento de arquivo para a solução de problemas de inicialização.
- Encaminhar tokens de autenticação do Windows.

## Como instalar e usar o Módulo do ASP.NET Core

Para obter instruções sobre como instalar e usar o Módulo do ASP.NET Core, confira [Hospedar o ASP.NET Core no Windows com o IIS](#).

## Configuração com web.config

O Módulo do ASP.NET Core está configurado com a seção `aspNetCore` do nó `system.webServer` no arquivo `web.config` do site.

O seguinte arquivo `web.config` é publicado para uma [implantação dependente de estrutura](#) e configura o Módulo do ASP.NET Core para manipular solicitações de site:

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <location path="." inheritInChildApplications="false">
        <system.webServer>
            <handlers>
                <add name="aspNetCore" path="*" verb="*" modules="AspNetCoreModuleV2" resourceType="Unspecified" />
            </handlers>
            <aspNetCore processPath="dotnet"
                        arguments=".\\MyApp.dll"
                        stdoutLogEnabled="false"
                        stdoutLogFile=".\\logs\\stdout"
                        hostingModel="InProcess" />
        </system.webServer>
    </location>
</configuration>

```

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.webServer>
    <handlers>
      <add name="aspNetCore" path="*" verb="*" modules="AspNetCoreModule" resourceType="Unspecified" />
    </handlers>
    <aspNetCore processPath="dotnet"
      arguments=".\\MyApp.dll"
      stdoutLogEnabled="false"
      stdoutLogFile=".\\logs\\stdout" />
  </system.webServer>
</configuration>

```

O seguinte `web.config` é publicado para uma [implantação autossuficiente](#):

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <location path=". " inheritInChildApplications="false">
    <system.webServer>
      <handlers>
        <add name="aspNetCore" path="*" verb="*" modules="AspNetCoreModuleV2" resourceType="Unspecified" />
      </handlers>
      <aspNetCore processPath=".\\MyApp.exe"
        stdoutLogEnabled="false"
        stdoutLogFile=".\\logs\\stdout"
        hostingModel="InProcess" />
    </system.webServer>
  </location>
</configuration>

```

A propriedade [InheritInChildApplications](#) é definida como `false` para indicar que as configurações especificadas no elemento `<location>` não são herdadas por aplicativos que residem em um subdiretório do aplicativo.

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.webServer>
    <handlers>
      <add name="aspNetCore" path="*" verb="*" modules="AspNetCoreModule" resourceType="Unspecified" />
    </handlers>
    <aspNetCore processPath=".\\MyApp.exe"
      stdoutLogEnabled="false"
      stdoutLogFile=".\\logs\\stdout" />
  </system.webServer>
</configuration>

```

Quando um aplicativo é implantado no [Serviço de Aplicativo do Azure](#), o caminho `stdoutLogFile` é definido para `\\\%home%\LogFiles\stdout`. O caminho salva logs de stdout para a pasta *LogFiles*, que é um local criado automaticamente pelo serviço.

Para saber mais sobre a configuração de subaplicativos do IIS, confira [Hospedar o ASP.NET Core no Windows com o IIS](#).

### Atributos do elemento `aspNetCore`

ATRIBUTO	DESCRIÇÃO	PADRÃO
----------	-----------	--------

ATTRIBUTO	DESCRIÇÃO	PADRÃO
<code>arguments</code>	<p>Atributo de cadeia de caracteres opcional.</p> <p>Argumentos para o executável especificado em <b>processPath</b>.</p>	
<code>disableStartupErrorPage</code>	<p>Atributo booleano opcional.</p> <p>Se for true, a página <b>502.5 – Falha do Processo</b> será suprimida e a página de código de status 502, configurada no <i>web.config</i>, terá precedência.</p>	<code>false</code>
<code>forwardWindowsAuthToken</code>	<p>Atributo booleano opcional.</p> <p>Se for true, o token será encaminhado para o processo filho escutando em <code>%ASPNETCORE_PORT%</code> como um cabeçalho 'MS-ASPNETCORE-WINAUTHTOKEN' por solicitação. É responsabilidade desse processo chamar <code>CloseHandle</code> nesse token por solicitação.</p>	<code>true</code>
<code>hostingModel</code>	<p>Atributo de cadeia de caracteres opcional.</p> <p>Especifica o modelo de hospedagem como em processo (<code>InProcess</code>) ou de fora do processo (<code>OutOfProcess</code>).</p>	<code>OutOfProcess</code>
<code>processesPerApplication</code>	<p>Atributo inteiro opcional.</p> <p>Especifica o número de instâncias do processo especificado na configuração <b>processPath</b> que pode ser ativada por aplicativo.</p> <p>†Para hospedagem em processo, o valor está limitado a <code>1</code>.</p>	Padrão: <code>1</code> Mín.: <code>1</code> Máx.: <code>100 +</code>
<code>processPath</code>	<p>Atributo de cadeia de caracteres obrigatório.</p> <p>Caminho para o executável que inicia um processo que escuta solicitações HTTP. Caminhos relativos são compatíveis. Se o caminho começa com <code>.</code>, o caminho é considerado relativo à raiz do site.</p>	

ATTRIBUTO	DESCRIÇÃO	PADRÃO
<code>rapidFailsPerMinute</code>	<p>Atributo inteiro opcional.</p> <p>Especifica o número de vezes que o processo especificado em <b>processPath</b> pode falhar por minuto. Se esse limite for excedido, o módulo interromperá a inicialização do processo pelo restante do minuto.</p> <p>Sem suporte com hospedagem padrão.</p>	Padrão: <input type="text" value="10"/> Mín.: <input type="text" value="0"/> Máx.: <input type="text" value="100"/>
<code>requestTimeout</code>	<p>Atributo de intervalo de tempo opcional.</p> <p>Especifica a duração para a qual o Módulo do ASP.NET Core aguarda uma resposta do processo que escuta em %ASPNETCORE_PORT%.</p> <p>Em versões do Módulo do ASP.NET Core que acompanham a versão do ASP.NET Core 2.1 ou posterior, o <code>requestTimeout</code> é especificado em horas, minutos e segundos.</p> <p>Não se aplica à hospedagem em processo. Para a hospedagem em processo, o módulo aguarda o aplicativo processar a solicitação.</p>	Padrão: <input type="text" value="00:02:00"/> Mín.: <input type="text" value="00:00:00"/> Máx.: <input type="text" value="360:00:00"/>
<code>shutdownTimeLimit</code>	<p>Atributo inteiro opcional.</p> <p>Duração em segundos que o módulo espera para o executável desligar normalmente quando o arquivo <code>app_offline.htm</code> é detectado.</p>	Padrão: <input type="text" value="10"/> Mín.: <input type="text" value="0"/> Máx.: <input type="text" value="600"/>

ATTRIBUTO	DESCRIÇÃO	PADRÃO
<code>startupTimeLimit</code>	<p>Atributo inteiro opcional.</p> <p>Duração em segundos que o módulo espera para o arquivo executável iniciar um processo escutando na porta. Se esse tempo limite é excedido, o módulo encerra o processo. O módulo tentará reiniciar o processo quando ele receber uma nova solicitação e continuará a tentar reiniciar o processo em solicitações subsequentes de entrada, a menos que o aplicativo falhe em iniciar um número de vezes igual a <b>rapidFailsPerMinute</b> no último minuto sem interrupção.</p> <p>Um valor de 0 (zero) <b>não</b> é considerado um tempo limite infinito.</p>	Padrão: <input type="text" value="120"/> Mín.: <input type="text" value="0"/> Máx.: <input type="text" value="3600"/>
<code>stdoutLogEnabled</code>	<p>Atributo booleano opcional.</p> <p>Se for true, <b>stdout</b> e <b>stderr</b> para o processo especificado em <b>processPath</b> serão redirecionados para o arquivo especificado em <b>stdoutLogFile</b>.</p>	<input type="checkbox"/>
<code>stdoutLogFile</code>	<p>Atributo de cadeia de caracteres opcional.</p> <p>Especifica o caminho relativo ou absoluto para o qual <b>stdout</b> e <b>stderr</b> do processo especificado em <b>processPath</b> são registrados em log. Os caminhos relativos são relativos à raiz do site. Qualquer caminho começando com <code>.</code> é relativo à raiz do site e todos os outros caminhos são tratados como caminhos absolutos. Todas as pastas fornecidas no caminho são criadas pelo módulo quando o arquivo de log é criado. Usando delimitadores de sublinhado, um carimbo de data/hora, uma ID de processo e a extensão de arquivo (<code>.log</code>) são adicionados ao último segmento do caminho <b>stdoutLogFile</b>. Se <code>.\logs\stdout</code> é fornecido como um valor, um log de exemplo stdout é salvo como <code>stdout_20180205194132_1934.log</code> na pasta <code>logs</code> quando salvos em 5/2/2018, às 19:41:32, com uma ID de processo de 1934.</p>	<code>aspnetcore-stdout</code>

ATTRIBUTO	DESCRIÇÃO	PADRÃO
<code>arguments</code>	<p>Atributo de cadeia de caracteres opcional.</p> <p>Argumentos para o executável especificado em <b>processPath</b>.</p>	
<code>disableStartupErrorPage</code>	<p>Atributo booleano opcional.</p> <p>Se for true, a página <b>502.5 – Falha do Processo</b> será suprimida e a página de código de status 502, configurada no <i>web.config</i>, terá precedência.</p>	<code>false</code>
<code>forwardWindowsAuthToken</code>	<p>Atributo booleano opcional.</p> <p>Se for true, o token será encaminhado para o processo filho escutando em <code>%ASPNETCORE_PORT%</code> como um cabeçalho 'MS-ASPNETCORE-WINAUTHTOKEN' por solicitação. É responsabilidade desse processo chamar <code>CloseHandle</code> nesse token por solicitação.</p>	<code>true</code>
<code>processesPerApplication</code>	<p>Atributo inteiro opcional.</p> <p>Especifica o número de instâncias do processo especificado na configuração <b>processPath</b> que pode ser ativada por aplicativo.</p>	Padrão: <code>1</code> Mín.: <code>1</code> Máx.: <code>100</code>
<code>processPath</code>	<p>Atributo de cadeia de caracteres obrigatório.</p> <p>Caminho para o executável que inicia um processo que escuta solicitações HTTP. Caminhos relativos são compatíveis. Se o caminho começa com <code>.</code>, o caminho é considerado relativo à raiz do site.</p>	
<code>rapidFailsPerMinute</code>	<p>Atributo inteiro opcional.</p> <p>Especifica o número de vezes que o processo especificado em <b>processPath</b> pode falhar por minuto. Se esse limite for excedido, o módulo interromperá a inicialização do processo pelo restante do minuto.</p>	Padrão: <code>10</code> Mín.: <code>0</code> Máx.: <code>100</code>

ATTRIBUTO	DESCRIÇÃO	PADRÃO
<code>requestTimeout</code>	<p>Atributo de intervalo de tempo opcional.</p> <p>Especifica a duração para a qual o Módulo do ASP.NET Core aguarda uma resposta do processo que escuta em %ASPNETCORE_PORT%.</p> <p>Em versões do Módulo do ASP.NET Core que acompanham a versão do ASP.NET Core 2.1 ou posterior, o <code>requestTimeout</code> é especificado em horas, minutos e segundos.</p>	Padrão: <code>00:02:00</code> Mín.: <code>00:00:00</code> Máx.: <code>360:00:00</code>
<code>shutdownTimeLimit</code>	<p>Atributo inteiro opcional.</p> <p>Duração em segundos que o módulo espera para o executável desligar normalmente quando o arquivo <code>app_offline.htm</code> é detectado.</p>	Padrão: <code>10</code> Mín.: <code>0</code> Máx.: <code>600</code>
<code>startupTimeLimit</code>	<p>Atributo inteiro opcional.</p> <p>Duração em segundos que o módulo espera para o arquivo executável iniciar um processo escutando na porta. Se esse tempo limite é excedido, o módulo encerra o processo. O módulo tentará reiniciar o processo quando ele receber uma nova solicitação e continuará a tentar reiniciar o processo em solicitações subsequentes de entrada, a menos que o aplicativo falhe em iniciar um número de vezes igual a <b>rapidFailsPerMinute</b> no último minuto sem interrupção.</p> <p>Um valor de 0 (zero) <b>não</b> é considerado um tempo limite infinito.</p>	Padrão: <code>120</code> Mín.: <code>0</code> Máx.: <code>3600</code>
<code>stdoutLogEnabled</code>	<p>Atributo booleano opcional.</p> <p>Se for true, <b>stdout</b> e <b>stderr</b> para o processo especificado em <b>processPath</b> serão redirecionados para o arquivo especificado em <b>stdoutLogFile</b>.</p>	<code>false</code>

ATRIBUTO	DESCRIÇÃO	PADRÃO
<code>stdoutLogFile</code>	<p>Atributo de cadeia de caracteres opcional.</p> <p>Especifica o caminho relativo ou absoluto para o qual <b>stdout</b> e <b>stderr</b> do processo especificado em <b>processPath</b> são registrados em log. Os caminhos relativos são relativos à raiz do site. Qualquer caminho começando com <code>.</code> é relativo à raiz do site e todos os outros caminhos são tratados como caminhos absolutos. As pastas fornecidas no caminho devem existir para que o módulo crie o arquivo de log. Usando delimitadores de sublinhado, um carimbo de data/hora, uma ID de processo e a extensão de arquivo (<i>.log</i>) são adicionados ao último segmento do caminho <b>stdoutLogFile</b>. Se <code>.\logs\stdout</code> é fornecido como um valor, um log de exemplo <code>stdout</code> é salvo como <code>stdout_20180205194132_1934.log</code> na pasta <i>logs</i> quando salvos em 5/2/2018, às 19:41:32, com uma ID de processo de 1934.</p>	<code>aspnetcore-stdout</code>

ATRIBUTO	DESCRIÇÃO	PADRÃO
<code>arguments</code>	<p>Atributo de cadeia de caracteres opcional.</p> <p>Argumentos para o executável especificado em <b>processPath</b>.</p>	
<code>disableStartUpErrorPage</code>	<p>Atributo booleano opcional.</p> <p>Se for true, a página <b>502.5 – Falha do Processo</b> será suprimida e a página de código de status 502, configurada no <i>web.config</i>, terá precedência.</p>	<code>false</code>
<code>forwardWindowsAuthToken</code>	<p>Atributo booleano opcional.</p> <p>Se for true, o token será encaminhado para o processo filho escutando em %ASPNETCORE_PORT% como um cabeçalho 'MS-ASPNETCORE-WINAUTHTOKEN' por solicitação. É responsabilidade desse processo chamar CloseHandle nesse token por solicitação.</p>	<code>true</code>

ATTRIBUTO	DESCRIÇÃO	PADRÃO
<code>processesPerApplication</code>	Atributo inteiro opcional.  Especifica o número de instâncias do processo especificado na configuração <b>processPath</b> que pode ser ativada por aplicativo.	Padrão: <input type="text" value="1"/> Mín.: <input type="text" value="1"/> Máx.: <input type="text" value="100"/>
<code>processPath</code>	Atributo de cadeia de caracteres obrigatório.  Caminho para o executável que inicia um processo que escuta solicitações HTTP. Caminhos relativos são compatíveis. Se o caminho começa com <code>.</code> , o caminho é considerado relativo à raiz do site.	
<code>rapidFailsPerMinute</code>	Atributo inteiro opcional.  Especifica o número de vezes que o processo especificado em <b>processPath</b> pode falhar por minuto. Se esse limite for excedido, o módulo interromperá a inicialização do processo pelo restante do minuto.	Padrão: <input type="text" value="10"/> Mín.: <input type="text" value="0"/> Máx.: <input type="text" value="100"/>
<code>requestTimeout</code>	Atributo de intervalo de tempo opcional.  Especifica a duração para a qual o Módulo do ASP.NET Core aguarda uma resposta do processo que escuta em <code>%ASPNETCORE_PORT%</code> .  Em versões do Módulo ASP.NET Core que acompanham a versão do ASP.NET Core 2.0 ou anterior, o <code>requestTimeout</code> deve ser especificado somente em minutos inteiros, caso contrário, ele assume o valor padrão de 2 minutos.	Padrão: <input type="text" value="00:02:00"/> Mín.: <input type="text" value="00:00:00"/> Máx.: <input type="text" value="360:00:00"/>
<code>shutdownTimeLimit</code>	Atributo inteiro opcional.  Duração em segundos que o módulo espera para o executável desligar normalmente quando o arquivo <code>app_offline.htm</code> é detectado.	Padrão: <input type="text" value="10"/> Mín.: <input type="text" value="0"/> Máx.: <input type="text" value="600"/>

ATTRIBUTO	DESCRIÇÃO	PADRÃO
<code>startupTimeLimit</code>	<p>Atributo inteiro opcional.</p> <p>Duração em segundos que o módulo espera para o arquivo executável iniciar um processo escutando na porta. Se esse tempo limite é excedido, o módulo encerra o processo. O módulo tentará reiniciar o processo quando ele receber uma nova solicitação e continuará a tentar reiniciar o processo em solicitações subsequentes de entrada, a menos que o aplicativo falhe em iniciar um número de vezes igual a <b>rapidFailsPerMinute</b> no último minuto sem interrupção.</p>	Padrão: <code>120</code> Mín.: <code>0</code> Máx.: <code>3600</code>
<code>stdoutLogEnabled</code>	<p>Atributo booleano opcional.</p> <p>Se for true, <b>stdout</b> e <b>stderr</b> para o processo especificado em <b>processPath</b> serão redirecionados para o arquivo especificado em <b>stdoutLogFile</b>.</p>	<code>false</code>
<code>stdoutLogFile</code>	<p>Atributo de cadeia de caracteres opcional.</p> <p>Especifica o caminho relativo ou absoluto para o qual <b>stdout</b> e <b>stderr</b> do processo especificado em <b>processPath</b> são registrados em log. Os caminhos relativos são relativos à raiz do site. Qualquer caminho começando com <code>.</code> é relativo à raiz do site e todos os outros caminhos são tratados como caminhos absolutos. As pastas fornecidas no caminho devem existir para que o módulo crie o arquivo de log. Usando delimitadores de sublinhado, um carimbo de data/hora, uma ID de processo e a extensão de arquivo (<code>.log</code>) são adicionados ao último segmento do caminho <b>stdoutLogFile</b>. Se <code>.\logs\stdout</code> é fornecido como um valor, um log de exemplo stdout é salvo como <code>stdout_20180205194132_1934.log</code> na pasta <code>logs</code> quando salvos em 5/2/2018, às 19:41:32, com uma ID de processo de 1934.</p>	<code>aspnetcore-stdout</code>

## Definindo variáveis de ambiente

Variáveis de ambiente podem ser especificadas para o processo no atributo `processPath`. Especificar uma variável de ambiente com o elemento filho `<environmentVariable>` de um elemento de coleção `<environmentVariables>`.

Variáveis de ambiente definidas nesta seção têm precedência sobre variáveis de ambiente do sistema.

Variáveis de ambiente podem ser especificadas para o processo no atributo `processPath`. Especificar uma variável de ambiente com o elemento filho `<environmentVariable>` de um elemento de coleção `<environmentVariables>`.

#### WARNING

As variáveis de ambiente definidas nesta seção são conflitantes com as variáveis de ambiente do sistema definidas com o mesmo nome. Quando a variável de ambiente é definida no arquivo `web.config` e no nível do sistema do Windows, o valor do arquivo `web.config` fica anexado ao valor da variável de ambiente do sistema (por exemplo, `ASPNETCORE_ENVIRONMENT: Development;Development`), o que impede a inicialização do aplicativo.

O exemplo a seguir define duas variáveis de ambiente. `ASPNETCORE_ENVIRONMENT` configura o ambiente do aplicativo para `Development`. Um desenvolvedor pode definir esse valor temporariamente no arquivo `web.config` para forçar o carregamento da [Página de Exceções do Desenvolvedor](#) ao depurar uma exceção de aplicativo. `CONFIG_DIR` é um exemplo de uma variável de ambiente definida pelo usuário, em que o desenvolvedor escreveu código que lê o valor de inicialização para formar um caminho no qual carregar o arquivo de configuração do aplicativo.

```
<aspNetCore processPath="dotnet"
            arguments=".\\MyApp.dll"
            stdoutLogEnabled="false"
            stdoutLogFile="\\?\%home%\LogFiles\stdout"
            hostingModel="InProcess">
  <environmentVariables>
    <environmentVariable name="ASPNETCORE_ENVIRONMENT" value="Development" />
    <environmentVariable name="CONFIG_DIR" value="f:\\application_config" />
  </environmentVariables>
</aspNetCore>
```

```
<aspNetCore processPath="dotnet"
            arguments=".\\MyApp.dll"
            stdoutLogEnabled="false"
            stdoutLogFile="\\?\%home%\LogFiles\stdout">
  <environmentVariables>
    <environmentVariable name="ASPNETCORE_ENVIRONMENT" value="Development" />
    <environmentVariable name="CONFIG_DIR" value="f:\\application_config" />
  </environmentVariables>
</aspNetCore>
```

#### NOTE

Em vez de configurar o ambiente diretamente no `web.config`, você pode incluir a propriedade `<EnvironmentName>` no perfil de publicação (`.pubxml`) ou no perfil de projeto. Esta abordagem define o ambiente no arquivo `web.config` quando o projeto é publicado:

```
<PropertyGroup>
  <EnvironmentName>Development</EnvironmentName>
</PropertyGroup>
```

#### WARNING

Defina a variável de ambiente apenas `ASPNETCORE_ENVIRONMENT` para `Development` em servidores de preparo e de teste que não estão acessíveis a redes não confiáveis, tais como a Internet.

## app\_offline.htm

Se um arquivo com o nome `app_offline.htm` é detectado no diretório raiz de um aplicativo, o Módulo do ASP.NET Core tenta desligar normalmente o aplicativo e parar o processamento de solicitações de entrada. Se o aplicativo ainda está em execução após o número de segundos definido em `shutdownTimeLimit`, o Módulo do ASP.NET Core encerra o processo em execução.

Enquanto o arquivo `app_offline.htm` estiver presente, o Módulo do ASP.NET Core responderá às solicitações enviando o conteúdo do arquivo `app_offline.htm`. Quando o arquivo `app_offline.htm` é removido, a próxima solicitação inicia o aplicativo.

Ao usar o modelo de hospedagem de fora do processo, talvez o aplicativo não desligue imediatamente se houver uma conexão aberta. Por exemplo, uma conexão websocket pode atrasar o desligamento do aplicativo.

## Página de erro de inicialização

A hospedagem em processo e fora do processo produzem páginas de erro personalizadas quando falham ao iniciar o aplicativo.

Se o Módulo do ASP.NET Core falhar ao encontrar o manipulador de solicitação em processo ou fora do processo, uma página de código de status `500.0 – falha ao carregar manipulador no processo/fora do processo` será exibida.

No caso da hospedagem em processo, se o módulo do ASP.NET Core falhar ao iniciar o aplicativo, uma página de código de status `500.30 – falha ao iniciar` será exibida.

Para hospedagem fora do processo, se o Módulo do ASP.NET Core falhar ao iniciar o processo de back-end ou se o processo de back-end iniciar, mas falhar ao escutar na porta configurada, uma página de código de status `502.5 – falha no processo` será exibida.

Para suprimir essa página e reverter para a página de código de status 5xx padrão do IIS, use o atributo `disableStartupErrorPage`. Para obter mais informações sobre como configurar mensagens de erro personalizadas, veja [Erros HTTP <httpErrors>](#).

Se o Módulo do ASP.NET Core falhar ao iniciar o processo de back-end ou se o processo de back-end iniciar, mas falhar ao escutar na porta configurada, uma página de código de status `502.5 – falha no processo` será exibida. Para omitir esta página e reverter para a página de código de status 502 padrão do IIS, use o atributo `disableStartupErrorPage`. Para obter mais informações sobre como configurar mensagens de erro personalizadas, veja [Erros HTTP <httpErrors>](#).

The screenshot shows a browser window with the title bar 'IIS 502.5 Error'. The address bar shows 'localhost:9001'. The main content area displays the error message 'HTTP Error 502.5 - Process Failure'. Below the title, there is a section titled 'Common causes of this issue:' with three bullet points: 'The application process failed to start', 'The application process started but then stopped', and 'The application process started but failed to listen on the configured port'. Another section titled 'Troubleshooting steps:' contains three bullet points: 'Check the system event log for error messages', 'Enable logging the application process' stdout messages', and 'Attach a debugger to the application process and inspect'. At the bottom, a link 'For more information visit: <http://go.microsoft.com/fwlink/?LinkId=808681>' is provided.

## Criação de log e redirecionamento

O Módulo do ASP.NET Core redireciona as saídas de console `stdout` e `stderr` para o disco se os atributos `stdoutEnabled` e `stdoutLogFile` do elemento `aspNetCore` forem definidos. Todas as pastas no caminho `stdoutLogFile` são criadas pelo módulo quando o arquivo de log é criado. O pool de aplicativos deve ter acesso de gravação ao local em que os logs foram gravados (use `IIS AppPool\<app_pool_name>` para fornecer permissão de gravação).

Logs não sofrem rotação, a menos que ocorra a reciclagem/reinicialização do processo. É responsabilidade do hoster limitar o espaço em disco consumido pelos logs.

Usar o log de `stdout` é recomendado apenas para solucionar problemas de inicialização do aplicativo. Não use o log de `stdout` para fins gerais de registro em log do aplicativo. Para registro em log de rotina em um aplicativo ASP.NET Core, use uma biblioteca de registro em log que limita o tamanho do arquivo de log e realiza a rotação de logs. Para obter mais informações, veja [provedores de log de terceiros](#).

Uma extensão de arquivo e um carimbo de data/hora são adicionados automaticamente quando o arquivo de log é criado. O nome do arquivo de log é composto por meio do acréscimo do carimbo de data/hora, da ID do processo e da extensão de arquivo (`.log`) para o último segmento do caminho `stdoutLogFile` (normalmente `stdout`), delimitados por sublinhados. Se o caminho `stdoutLogFile` termina com `stdout`, um log para um aplicativo com um PID de 1934, criado em 5/2/2018 às 19:42:32, tem o nome de arquivo `stdout_20180205194132_1934.log`.

Se `stdoutEnabled` for falso, os erros que ocorrerem na inicialização do aplicativo serão capturados e emitidos no log de eventos até 30 KB. Após a inicialização, todos os logs adicionais são descartados.

O elemento `aspNetCore` de exemplo a seguir configura o registro em log de `stdout` para um aplicativo hospedado no Serviço de Aplicativo do Azure. Um caminho local ou um caminho de compartilhamento de rede é aceitável para o registro em log local. Confirme se a identidade do usuário AppPool tem permissão para gravar no caminho fornecido.

```
<aspNetCore processPath="dotnet"
    arguments=".\\MyApp.dll"
    stdoutLogEnabled="true"
    stdoutLogFile="\\?\%home%\LogFiles\stdout"
    hostingModel="InProcess">
</aspNetCore>
```

```
<aspNetCore processPath="dotnet"
    arguments=".\\MyApp.dll"
    stdoutLogEnabled="true"
    stdoutLogFile="\\?\%home%\LogFiles\stdout">
</aspNetCore>
```

## Logs de diagnóstico avançados

O Módulo do ASP.NET Core é configurável para fornecer logs de diagnóstico avançados. Adicione o elemento `<handlerSettings>` ao elemento `<aspNetCore>` no `web.config`. A definição de `debugLevel` como `TRACE` expõe uma fidelidade maior de informações de diagnóstico:

```
<aspNetCore processPath="dotnet"
    arguments=".\\MyApp.dll"
    stdoutLogEnabled="false"
    stdoutLogFile="\\?\%home%\LogFiles\stdout"
    hostingModel="InProcess">
<handlerSettings>
    <handlerSetting name="debugFile" value="aspnetcore-debug.log" />
    <handlerSetting name="debugLevel" value="FILE,TRACE" />
</handlerSettings>
</aspNetCore>
```

Os valores do nível de depuração (`debugLevel`) podem incluir o nível e a localização.

Níveis (na ordem do menos para o mais detalhado):

- ERROR
- WARNING
- INFO
- TRACE

Locais (vários locais são permitidos):

- CONSOLE
- EVENTLOG
- FILE

As configurações do manipulador também podem ser fornecidas por meio de variáveis de ambiente:

- `ASPNETCORE_MODULE_DEBUG_FILE` – caminho para o arquivo de log de depuração. (Padrão: `aspnetcore-debug.log`)
- `ASPNETCORE_MODULE_DEBUG` – configuração do nível de depuração.

### WARNING

**Não** deixe o log de depuração habilitado na implantação por mais tempo que o necessário para solucionar um problema. O tamanho do log não é limitado. Deixar o log de depuração habilitado pode esgotar o espaço em disco disponível e causar falha no servidor ou no serviço de aplicativo.

Veja [Configuração com web.config](#) para obter um exemplo do elemento `aspNetCore` no arquivo `web.config`.

## A configuração de proxy usa o protocolo HTTP e um token de emparelhamento

Só se aplica à hospedagem de fora do processo.

O proxy criado entre o Módulo do ASP.NET Core e o Kestrel usa o protocolo HTTP. O uso de HTTP é uma otimização de desempenho na qual o tráfego entre o módulo e o Kestrel ocorre em um endereço de loopback fora do adaptador de rede. Não há nenhum risco de interceptação do tráfego entre o módulo e o Kestrel em um local fora do servidor.

Um token de emparelhamento é usado para assegurar que as solicitações recebidas pelo Kestrel foram transmitidas por proxy pelo IIS e que não são provenientes de outra origem. O token de emparelhamento é criado e definido em uma variável de ambiente (`ASPNETCORE_TOKEN`) pelo módulo. O token de emparelhamento também é definido em um cabeçalho (`MS-ASPNETCORE-TOKEN`) em cada solicitação com proxy. O Middleware do IIS verifica cada solicitação recebida para confirmar se o valor de cabeçalho do token de emparelhamento corresponde ao valor da variável de ambiente. Se os valores do token forem incompatíveis, a solicitação será registrada em log e rejeitada. A variável de ambiente do token de emparelhamento e o tráfego entre o módulo e o Kestrel não são acessíveis em um local fora do servidor. Sem saber o valor do token de emparelhamento, um invasor não pode enviar solicitações que ignoram a verificação no Middleware do IIS.

## Módulo do ASP.NET Core com uma configuração do IIS compartilhada

O instalador do módulo do ASP.NET Core é executado com os privilégios da conta de **SISTEMA**. Já que a conta de sistema local não tem permissão para modificar o caminho do compartilhamento usado pela configuração compartilhada de IIS, o instalador experimenta um erro de acesso negado ao tentar definir as configurações de módulo em `applicationHost.config` no compartilhamento. Ao usar uma configuração compartilhada de IIS, siga estas etapas:

1. Desabilite a configuração compartilhada de IIS.
2. Execute o instalador.
3. Exportar o arquivo `applicationHost.config` atualizado para o compartilhamento.
4. Reabilite a Configuração Compartilhada do IIS.

## Versão do módulo e logs do instalador do pacote de hospedagem

Para determinar a versão do Módulo do ASP.NET Core instalado:

1. No sistema de hospedagem, navegue até `%windir%\System32\inetsrv`.
2. Localize o arquivo `aspnetcore.dll`.
3. Clique com o botão direito do mouse no arquivo e selecione **Propriedades** no menu contextual.
4. Selecione a guia **Detalhes**. A **Versão do arquivo** e a **Versão do produto** representam a versão instalada do módulo.

Os logs de instalador do pacote de hospedagem para o módulo são encontrados em `C:\Usuários\%UserName%\AppData\Local\Temp`. O arquivo é nomeado `dd_DotNetCoreWinSvrHosting_<carimbo de data/hora>_000_AspNetCoreModule_x64.log`.

## Locais dos arquivos de módulo, de esquema e de configuração

### Módulo

#### IIS (x86/amd64):

- %windir%\System32\inetsrv\aspnetcore.dll
- %windir%\SysWOW64\inetsrv\aspnetcore.dll
- %ProgramFiles%\IIS\Asp.Net Core Module\V2\aspnetcorev2.dll
- %ProgramFiles(x86)%\IIS\Asp.Net Core Module\V2\aspnetcorev2.dll

#### IIS Express (x86/amd64):

- %ProgramFiles%\IIS Express\aspnetcore.dll
- %ProgramFiles(x86)%\IIS Express\aspnetcore.dll
- %ProgramFiles%\IIS Express\Asp.Net Core Module\V2\aspnetcorev2.dll
- %ProgramFiles(x86)%\IIS Express\Asp.Net Core Module\V2\aspnetcorev2.dll

#### Esquema

##### IIS

- %windir%\System32\inetsrv\config\schema\aspnetcore\_schema.xml
- %windir%\System32\inetsrv\config\schema\aspnetcore\_schema\_v2.xml

##### IIS Express

- %ProgramFiles%\IIS Express\config\schema\aspnetcore\_schema.xml
- %ProgramFiles%\IIS Express\config\schema\aspnetcore\_schema\_v2.xml

#### Configuração

##### IIS

- %windir%\System32\inetsrv\config\applicationHost.config

##### IIS Express

- Visual Studio: {APPLICATION ROOT}\.vs\config\applicationHost.config
- *iisexpress.exe* CLI: %USERPROFILE%\Documents\IIS Express\config\applicationhost.config

Os arquivos podem ser encontrados pesquisando por *aspnetcore* no arquivo *applicationHost.config*.

## Recursos adicionais

- [Hospedar o ASP.NET Core no Windows com o IIS](#)
- [Repositório do GitHub do Módulo do ASP.NET Core \(origem de referência\)](#)
- [Módulos do IIS com o ASP.NET Core](#)

# Solucionar problemas no ASP.NET Core no Serviço de Aplicativo do Azure

21/01/2019 • 22 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

## IMPORTANT

### Versões prévias do ASP.NET Core com o Serviço de Aplicativo do Azure

Versões prévias do ASP.NET Core não são implantadas para o Serviço de Aplicativo do Azure por padrão. Para hospedar um aplicativo que usa uma versão prévia do ASP.NET Core, veja [Implantar versão prévia do ASP.NET Core para o Serviço de Aplicativo do Azure](#).

Este artigo fornece instruções sobre como diagnosticar um problema de inicialização do aplicativo ASP.NET Core ao usar as ferramentas de diagnóstico do Serviço de Aplicativo do Azure. Para obter conselhos sobre solução de problemas adicionais, consulte [Visão geral de diagnóstico do Serviço de Aplicativo do Azure](#) e [Como: monitorar aplicativos no Serviço de Aplicativo do Azure](#) na documentação do Azure.

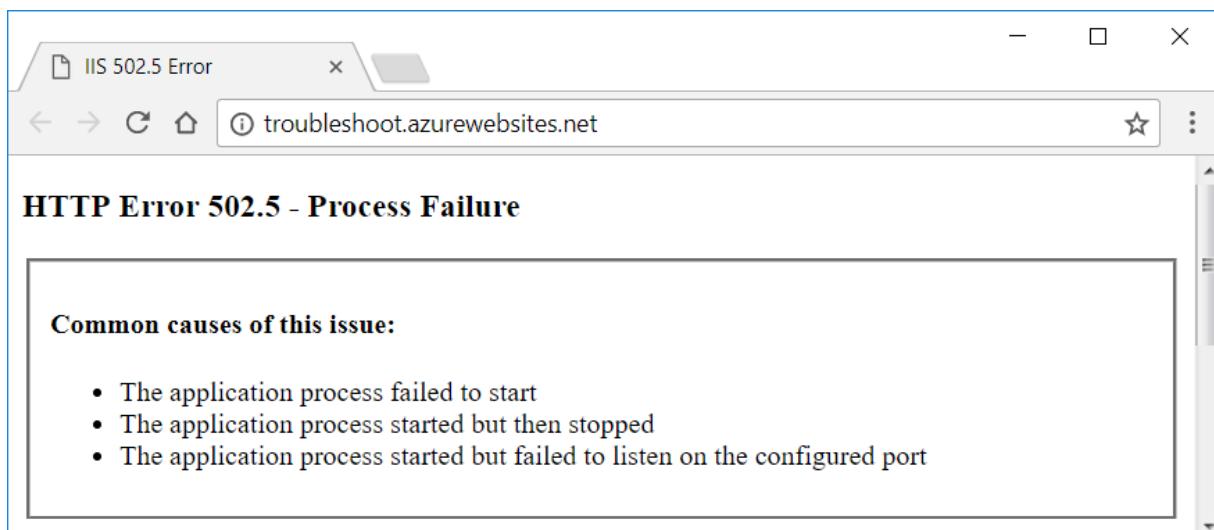
## Erros de inicialização do aplicativo

### 502.5 – Falha de Processo

O processo de trabalho falha. O aplicativo não foi iniciado.

O [Módulo do ASP.NET Core](#) tenta iniciar o processo de trabalho, mas falhar ao iniciar. Examinar o Log de Eventos do Aplicativo geralmente ajuda a solucionar esse tipo de problema. O acesso ao log é explicado na seção [Log de Eventos do Aplicativo](#).

A página do erro *502.5 – Falha no Processo* é retornada quando um erro de configuração do aplicativo faz com que o processo de trabalho falhe:



### 500 – Erro Interno do Servidor

O aplicativo é iniciado, mas um erro impede o servidor de atender à solicitação.

Esse erro ocorre no código do aplicativo durante a inicialização ou durante a criação de uma resposta. A resposta poderá não conter nenhum conteúdo, ou a resposta poderá ser exibida como um *500 – Erro Interno*.

*do Servidor* no navegador. O Log de Eventos do Aplicativo geralmente indica que o aplicativo iniciou normalmente. Da perspectiva do servidor, isso está correto. O aplicativo foi iniciado, mas não é capaz de gerar uma resposta válida. [Execute o aplicativo no console do Kudu](#) ou [habilite o log de stdout do Módulo do ASP.NET Core](#) para solucionar o problema.

### Redefinição de conexão

Se um erro ocorrer após os cabeçalhos serem enviados, será tarde demais para o servidor enviar um **500 – Erro Interno do Servidor** no caso de um erro ocorrer. Isso geralmente acontece quando ocorre um erro durante a serialização de objetos complexos para uma resposta. Esse tipo de erro é exibida como um erro de *redefinição de conexão* no cliente. O [Log de aplicativo](#) pode ajudar a solucionar esses tipos de erros.

## Limites de inicialização padrão

O Módulo do ASP.NET Core está configurado com um *startupTimeLimit* padrão de 120 segundos. Quando deixado no valor padrão, um aplicativo pode levar até dois minutos para iniciar antes que uma falha do processo seja registrada em log pelo módulo. Para obter informações sobre como configurar o módulo, veja [Atributos do elemento aspNetCore](#).

## Solucionar problemas de inicialização do aplicativo

### Log de Eventos do Aplicativo

Para acessar o Log de Eventos do Aplicativo, use a folha **Diagnosticar e solucionar problemas** no portal do Azure:

1. No portal do Azure, abra o aplicativo nos **Serviços de Aplicativos**.
2. Selecione **Diagnóstico e solução de problemas**.
3. Selecione o título **Ferramentas de Diagnóstico**.
4. Em **Ferramentas de Suporte**, selecione o botão **Eventos do Aplicativo**.
5. Examine o erro mais recente fornecido pela entrada *IIS AspNetCoreModule* ou *IIS AspNetCoreModule V2* na coluna **Origem**.

Uma alternativa ao uso da folha **Diagnosticar e resolver problemas** é examinar o arquivo de Log de Eventos do Aplicativo diretamente usando o [Kudu](#):

1. Abra **Ferramentas Avançadas** na área **Ferramentas de Desenvolvimento**. Selecione o botão **Ir→**. O console do Kudu é aberto em uma nova janela ou guia do navegador.
2. Usando a barra de navegação na parte superior da página, abra **Console de depuração** e selecione **CMD**.
3. Abra a pasta **LogFile**.
4. Selecione o ícone de lápis ao lado do arquivo *eventlog.xml*.
5. Examine o log. Role até o final do log para ver os eventos mais recentes.

### Execute o aplicativo no console do Kudu

Muitos erros de inicialização não produzem informações úteis no Log de Eventos do Aplicativo. Você pode executar o aplicativo no Console de Execução Remota do [Kudu](#) para descobrir o erro:

1. Abra **Ferramentas Avançadas** na área **Ferramentas de Desenvolvimento**. Selecione o botão **Ir→**. O console do Kudu é aberto em uma nova janela ou guia do navegador.
2. Usando a barra de navegação na parte superior da página, abra **Console de depuração** e selecione **CMD**.
3. Abra as pastas no caminho **site > wwwroot**.
4. No console, executando o assembly do aplicativo, execute o aplicativo propriamente dito.

- Se o aplicativo for uma [implantação dependente de estrutura](#), execute o assembly do aplicativo com `dotnet.exe`. No comando a seguir, substitua o nome do assembly do aplicativo para `<assembly_name>` : `dotnet .\<assembly_name>.dll`
- Se o aplicativo for uma [implantação autossuficiente](#), execute o arquivo executável do aplicativo. No comando a seguir, substitua o nome do assembly do aplicativo para `<assembly_name>` : `<assembly_name>.exe`

5. A saída do console do aplicativo, mostrando eventuais erros, é conectada ao console do Kudu.

### Log de stdout do Módulo do ASP.NET Core

O log de stdout do Módulo do ASP.NET Core geralmente registra mensagens de erro úteis não encontradas no Log de Eventos do Aplicativo. Para habilitar e exibir logs de stdout:

1. Navegue até a folha **Diagnosticar e resolver problemas** no portal do Azure.
2. Em **SELECIONAR CATEGORIA DE PROBLEMA**, selecione o botão **Aplicativo Web Inoperante**.
3. Em **Soluções Sugeridas > Habilitar o Redirecionamento de Log de Stdout**, selecione o botão para **Abrir o Console do Kudu para editar o Web.Config**.
4. No **Console de Diagnóstico** do Kudu, abra as pastas no caminho **site > wwwroot**. Role para baixo para revelar o arquivo `web.config` na parte inferior da lista.
5. Clique no ícone de lápis ao lado do arquivo `web.config`.
6. Defina **stdoutLogEnabled** para `true` e altere o caminho **stdoutLogFile** para `\?\%home%\LogFiles\stdout`.
7. Selecione **Salvar** para salvar o arquivo `web.config` atualizado.
8. Faça uma solicitação ao aplicativo.
9. Retorne para o portal do Azure. Selecione a folha **Ferramentas Avançadas** na área **FERRAMENTAS DE DESENVOLVIMENTO**. Selecione o botão **Ir→**. O console do Kudu é aberto em uma nova janela ou guia do navegador.
10. Usando a barra de navegação na parte superior da página, abra **Console de depuração** e selecione **CMD**.
11. Selecione a pasta **LogFiles**.
12. Inspecione a coluna **Modificado em** e selecione o ícone de lápis para editar o log de stdout com a data da última modificação.
13. Quando o arquivo de log é aberto, o erro é exibido.

Desabilite o registro em log de stdout quando a solução de problemas for concluída:

1. No **Console de Diagnóstico** do Kudu, retorne para o caminho **site > wwwroot** para revelar o arquivo `web.config`. Abra o arquivo **web.config** novamente, selecionando o ícone de lápis.
2. Defina **stdoutLogEnabled** para `false`.
3. Selecione **Salvar** para salvar o arquivo.

#### WARNING

Falha ao desabilitar o log de stdout pode levar a falhas de aplicativo ou de servidor. Não há limites para o tamanho do arquivo de log ou para o número de arquivos de log criados. Somente use o log de stdout para solucionar problemas de inicialização de aplicativo.

Para registro em log geral em um aplicativo ASP.NET Core após a inicialização, use uma biblioteca de registro em log que limita o tamanho do arquivo de log e realiza a rotação de logs. Para obter mais informações, veja [provedores de log de terceiros](#).

### Log de depuração do Módulo do ASP.NET Core

O log de depuração do Módulo do ASP.NET Core fornece registro em log adicional e mais profundo do

Módulo do ASP.NET Core. Para habilitar e exibir logs de stdout:

1. Para habilitar o log de diagnóstico avançado, execute um destes procedimentos:
  - Siga as instruções em [Logs de diagnóstico avançados](#) para configurar o aplicativo para um log de diagnósticos avançado. Reimplante o aplicativo.
  - Adicione a `<handlerSettings>` mostrada em [Logs de diagnóstico avançados](#) para o arquivo `web.config` do aplicativo ao vivo usando o console do Kudu:
    - a. Abra **Ferramentas Avançadas** na área **Ferramentas de Desenvolvimento**. Selecione o botão **Ir→**. O console do Kudu é aberto em uma nova janela ou guia do navegador.
    - b. Usando a barra de navegação na parte superior da página, abra **Console de depuração** e selecione **CMD**.
    - c. Abra as pastas no caminho **site > wwwroot**. Edite o arquivo `web.config` selecionando o botão de lápis. Adicione a seção `<handlerSettings>` conforme mostrado em [Logs de diagnóstico avançados](#). Selecione o botão **Salvar**.
2. Abra **Ferramentas Avançadas** na área **Ferramentas de Desenvolvimento**. Selecione o botão **Ir→**. O console do Kudu é aberto em uma nova janela ou guia do navegador.
3. Usando a barra de navegação na parte superior da página, abra **Console de depuração** e selecione **CMD**.
4. Abra as pastas no caminho **site > wwwroot**. Se você não fornecer um caminho para o arquivo `aspnetcore-debug.log`, o arquivo aparecerá na lista. Se você tiver fornecido um caminho, navegue até o local do arquivo de log.
5. Abra o arquivo de log com o botão de lápis ao lado do nome do arquivo.

Desabilite o registro em log de depuração quando a solução de problemas for concluída:

1. Para desabilitar o log de depuração avançado, execute um destes procedimentos:
  - Remova o `<handlerSettings>` do arquivo `web.config` localmente e reimplante o aplicativo.
  - Use o console do Kudu para editar o arquivo `web.config` e remover a seção `<handlerSettings>`. Salve o arquivo.

#### WARNING

A falha ao desabilitar o log de depuração pode levar a falhas de aplicativo ou de servidor. Não há nenhum limite no tamanho do arquivo de log. Somente use o log de depuração para solucionar problemas de inicialização de aplicativo.

Para registro em log geral em um aplicativo ASP.NET Core após a inicialização, use uma biblioteca de registro em log que limita o tamanho do arquivo de log e realiza a rotação de logs. Para obter mais informações, veja [provedores de log de terceiros](#).

## Erros de inicialização comuns

Consulte [Referência de erros comuns para o Serviço de Aplicativo do Azure e o IIS com o ASP.NET Core](#). A maioria dos problemas comuns que impedem a inicialização do aplicativo é abordada no tópico de referência.

## Aplicativo lento ou travando

Quando um aplicativo responde lentamente ou trava em uma solicitação, confira [Solucionar problemas de desempenho de aplicativo Web lento no Serviço de Aplicativo do Azure](#) para diretrizes de depuração.

## Depuração remota

Confira os seguintes tópicos:

- Seção "Depuração remota de aplicativos Web" de "Solucionar problemas de um aplicativo Web no Serviço de Aplicativo do Azure usando o Visual Studio" (documentação do Azure)
- Depuração Remota do ASP.NET Core no IIS no Azure no Visual Studio 2017 (documentação do Visual Studio)

## Informações do aplicativo

O [Application Insights](#) fornece telemetria de aplicativos hospedados no Serviço de Aplicativo do Azure, incluindo recursos de relatório e de registro de erros em log. O Application Insights só pode relatar erros ocorridos depois que o aplicativo é iniciado quando os recursos de registro em log do aplicativo se tornam disponíveis. Para obter mais informações, veja [Application Insights para ASP.NET Core](#).

## Folhas de monitoramento

As folhas de monitoramento fornecem uma experiência alternativa de solução de problemas para os métodos descritos anteriormente no tópico. Essas folhas podem ser usadas para diagnosticar erros da série 500.

Verifique se as Extensões do ASP.NET Core estão instaladas. Se as extensões não estiverem instaladas, instale-as manualmente:

1. Na seção de folha **FERRAMENTAS DE DESENVOLVIMENTO**, selecione a folha **Extensões**.
2. As **Extensões do ASP.NET Core** devem aparecer na lista.
3. Se as extensões não estiverem instaladas, selecione o botão **Adicionar**.
4. Escolha as **Extensões do ASP.NET Core** da lista.
5. Selecione **OK** para aceitar os termos legais.
6. Selecione **OK** na folha **Adicionar extensão**.
7. Uma mensagem pop-up informativa indica quando as extensões são instaladas com êxito.

Se o registro em log de stdout não estiver habilitado, siga estas etapas:

1. No portal do Azure, selecione a folha **Ferramentas Avançadas** na área **FERRAMENTAS DE DESENVOLVIMENTO**. Selecione o botão **Ir→**. O console do Kudu é aberto em uma nova janela ou guia do navegador.
2. Usando a barra de navegação na parte superior da página, abra **Console de depuração** e selecione **CMD**.
3. Abra as pastas no caminho **site > wwwroot** e role para baixo para revelar o arquivo *web.config* na parte inferior da lista.
4. Clique no ícone de lápis ao lado do arquivo *web.config*.
5. Defina **stdoutLogFile** para `true` e altere o caminho **stdoutLogFile** para  
`\?\%home%\LogFiles\stdout`.
6. Selecione **Salvar** para salvar o arquivo *web.config* atualizado.

Prossiga para ativar o log de diagnóstico:

1. No portal do Azure, selecione a folha **Logs de diagnóstico**.
2. Selecione a opção **Ligado** para **Log de Aplicativo (Sistema de arquivos)** e **Mensagens de erro detalhadas**. Selecione o botão **Salvar** na parte superior da folha.
3. Para incluir o rastreamento de solicitação com falha, também conhecido como FREB (Buffer de Evento de Solicitação com Falha), selecione a opção **Ligado** para o **Rastreamento de solicitação com falha**.
4. Selecione a folha **Fluxo de log**, que é listada imediatamente sob a folha **Logs de diagnóstico** no portal.
5. Faça uma solicitação ao aplicativo.
6. Dentro dos dados de fluxo de log, a causa do erro é indicada.

Sempre desabilite o registro em log de stdout após concluir a solução de problemas. Veja as instruções na seção [log de stdout do Módulo do ASP.NET Core](#).

Para exibir os logs de rastreamento de solicitação com falha (logs FReB):

1. Navegue até a folha **Diagnosticar e resolver problemas** no portal do Azure.
2. Selecione **Logs de Rastreamento de Solicitação com Falha** da área **FERRAMENTAS DE SUPORTE** da barra lateral.

Confira a [seção de Rastreamentos de solicitação com falha](#) no tópico [Habilitar o log de diagnósticos para aplicativos Web no Serviço de Aplicativo do Azure](#) e as [Perguntas frequentes do desempenho do aplicativo para Aplicativos Web no Azure: como ativar o rastreamento de solicitação com falha?](#) para obter mais informações.

Para obter mais informações, veja [Habilitar log de diagnósticos para aplicativos Web no Serviço de Aplicativo do Azure](#).

#### **WARNING**

Falha ao desabilitar o log de stdout pode levar a falhas de aplicativo ou de servidor. Não há limites para o tamanho do arquivo de log ou para o número de arquivos de log criados.

Para registro em log de rotina em um aplicativo ASP.NET Core, use uma biblioteca de registro em log que limita o tamanho do arquivo de log e realiza a rotação de logs. Para obter mais informações, veja [provedores de log de terceiros](#).

## Recursos adicionais

- [Tratar erros no ASP.NET Core](#)
- [Referência de erros comuns para o Serviço de Aplicativo do Azure e o IIS com o ASP.NET Core](#)
- [Soluçanar problemas de um aplicativo Web no Serviço de Aplicativo do Azure usando o Visual Studio](#)
- [Soluçanar problemas de erros HTTP de "502 – gateway incorreto" e "503 – serviço não disponível" em seus aplicativos Web do Azure](#)
- [Soluçanar problemas de desempenho lento do aplicativo Web no Serviço de Aplicativo do Azure](#)
- [Perguntas frequentes sobre o desempenho do aplicativo para aplicativos Web no Azure](#)
- [Área restrita do aplicativo Web do Azure \(limitações de execução de tempo de execução do Serviço de Aplicativo\)](#)
- [Azure Friday: experiência de diagnóstico e solução de problemas do Serviço de Aplicativo do Azure \(vídeo com 12 minutos\)](#)

# Referência de erros comuns para o Serviço de Aplicativo do Azure e o IIS com o ASP.NET Core

21/01/2019 • 27 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

Este tópico oferece conselhos de solução de problemas para erros comuns ao hospedar aplicativos ASP.NET Core no Serviço de Aplicativos do Azure e no IIS.

Colete as seguintes informações:

- Comportamento do navegador (código de status e mensagem de erro)
- Entradas do Log de Eventos do Aplicativo
  - Serviço de Aplicativo do Azure – Confira [Solucionar problemas no ASP.NET Core no Serviço de Aplicativo do Azure](#).
  - IIS
    1. Selecione **Iniciar** no menu **Windows**, digite *Visualizador de Eventos* e pressione **Enter**.
    2. Após o **Visualizador de Eventos** ser aberto, expanda **Logs do Windows > Aplicativo** na barra lateral.
- Entradas do log de depuração e stdout do Módulo do ASP.NET Core
  - Serviço de Aplicativo do Azure – Confira [Solucionar problemas no ASP.NET Core no Serviço de Aplicativo do Azure](#).
  - IIS – Siga as instruções nas seções [Criação de log e redirecionamento](#) e [Logs de diagnóstico avançados](#) do tópico Módulo do ASP.NET Core.

Compare as informações do erro para os erros comuns a seguir. Se uma correspondência for encontrada, siga o aviso de solução de problemas.

A lista de erros neste tópico não é exaustiva. Se você encontrar um erro não listado aqui, abra um novo problema usando o botão **Comentários sobre o Conteúdo** na parte inferior deste tópico com instruções detalhadas sobre como reproduzir o erro.

## IMPORTANT

### Versões prévias do ASP.NET Core com o Serviço de Aplicativo do Azure

Versões prévias do ASP.NET Core não são implantadas para o Serviço de Aplicativo do Azure por padrão. Para hospedar um aplicativo que usa uma versão prévia do ASP.NET Core, veja [Implantar versão prévia do ASP.NET Core para o Serviço de Aplicativo do Azure](#).

## O instalador não pode obter os Pacotes Redistribuíveis do VC++

- **Exceção do instalador:** 0x80072efd –OU– 0x80072f76 – erro não especificado
- **Exceção do log do instalador:** Erro 0x80072efd –OU– 0x80072f76: Falha ao executar o pacote EXE

<sup>+</sup>O log está localizado em

*C:\Users\{USER}\AppData\Local\Temp\dd\_DotNetCoreWinSvrHosting\_{TIMESTAMP}.log*.

Solução de problemas:

Se o sistema não tiver acesso à Internet durante a [instalação do pacote de hospedagem do .NET Core](#), essa exceção ocorrerá quando o instalador for impedido de obter os *Pacotes Redistribuíveis do Microsoft Visual C++ 2015*. Obtenha um instalador do [Centro de Download da Microsoft](#). Se o instalador falhar, o servidor poderá não receber o tempo de execução do .NET Core necessário para hospedar uma **FDD (implantação dependente de estrutura)**. Se estiver hospedando uma FDD, confirme se o tempo de execução está instalado em **Programas e Recursos** ou **Aplicativos e recursos**. Se um tempo de execução específico for necessário, baixe o tempo de execução dos [Arquivos de Download do .NET](#) e instale-o no sistema. Depois de instalar o tempo de execução, reinicie o sistema ou o IIS executando **net stop was /y** seguido por **net start w3svc** em um prompt de comando.

## O upgrade do sistema operacional removeu o Módulo do ASP.NET Core de 32 bits

**Log do Aplicativo:** A DLL do Módulo **C:\WINDOWS\system32\inetsrv\aspnetcore.dll** falhou ao ser carregada. Os dados são o erro.

Solução de problemas:

Arquivos que não são do sistema operacional no diretório **C:\Windows\SysWOW64\inetsrv** não são preservados durante um upgrade do sistema operacional. Se o Módulo do ASP.NET Core estiver instalado antes de uma atualização do sistema operacional e, em seguida, qualquer pool de aplicativos for executado no modo de 32 bits após uma atualização do sistema operacional, esse problema será encontrado. Após um upgrade do sistema operacional, repare o Módulo do ASP.NET Core. Veja [Instalar o pacote de Hospedagem do .NET Core](#). Selecione **Reparar** ao executar o instalador.

## Um aplicativo x86 é implantado, mas o pool de aplicativos não está habilitado para aplicativos de 32 bits

- **Navegador:** Erro HTTP 500.30 – Falha de início no processo do ANCM
- **Log do Aplicativo:** Aplicativo '/LM/W3SVC/5/ROOT' com raiz física '{PATH}' atingiu uma exceção gerenciada inesperada, código de exceção = '0xe0434352'. Verifique os logs de stderr para obter mais informações. Aplicativo '/LM/W3SVC/5/ROOT' com raiz física '{PATH}' falhou ao carregar o clr e o aplicativo gerenciado. O thread de trabalho do CLR foi encerrado prematuramente
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log é criado, mas vazio.
- **Log de depuração do módulo do ASP.NET Core:** HRESULT com falha retornou: 0x8007023e

Esse cenário é interceptado pelo SDK ao publicar um aplicativo autocontido. O SDK produzirá um erro se o RID não coincidir com o destino da plataforma (por exemplo, RID `win10-x64` com `<PlatformTarget>x86</PlatformTarget>` no arquivo de projeto).

Solução de problemas:

Para uma implantação dependente da estrutura x86 (`<PlatformTarget>x86</PlatformTarget>`), habilite o pool de aplicativos de IIS para aplicativos de 32 bits. No Gerenciador do IIS, abra as **Configurações Avançadas** do pool de aplicativos e defina **Habilitar Aplicativos de 32 Bits** como **Verdadeiro**.

## Conflitos de plataforma com o RID

- **Navegador:** Erro HTTP 502.5 – falha do processo
- **Log do Aplicativo:** O aplicativo 'MACHINE/WEBROOT/APPHOST/{ASSEMBLY}' com raiz física 'C:{PATH}' falhou ao iniciar o processo com a linha de comando '"C:{PATH}{ASSEMBLY}.{exe|dll}"', ErrorCode = '0x80004005 : ff.'

- **Log de stdout do Módulo do ASP.NET Core:** Exceção sem tratamento:  
System.BadImageFormatException: Não foi possível carregar arquivo ou o assembly '{ASSEMBLY}.dll'. Foi feita uma tentativa de carregar um programa com um formato incorreto.

Solução de problemas:

- Confirme se o aplicativo é executado localmente no Kestrel. Uma falha do processo pode ser o resultado de um problema no aplicativo. Para obter mais informações, confira [Solução de problemas \(IIS\)](#) ou [Solução de problemas \(Serviço de Aplicativo do Azure\)](#).
- Se essa exceção ocorrer para uma implantação dos Aplicativos do Azure ao fazer upgrade de um aplicativo e implantar assemblies mais recentes, exclua manualmente todos os arquivos da implantação anterior. Assemblies incompatíveis remanescentes podem resultar em uma exceção `System.BadImageFormatException` durante a implantação de um aplicativo atualizado.

## Ponto de extremidade de URI incorreto ou site interrompido

- **Navegador:** ERR\_CONNECTION\_REFUSED –OU– Não é possível estabelecer conexão
- **Log do Aplicativo:** Nenhuma entrada
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log não é criado.
- **Log de depuração do módulo do ASP.NET Core:** O arquivo de log não é criado.

Solução de problemas:

- Confirme se o ponto de extremidade do URI correto para o aplicativo está sendo usado. Verifique as associações.
- Confirme que o site do IIS não está no estado *Parado*.

## Recursos do servidor CoreWebEngine ou W3SVC desabilitados

**Exceção do Sistema Operacional:** Os recursos CoreWebEngine e W3SVC do IIS 7.0 devem ser instalados para usar o Módulo do ASP.NET Core.

Solução de problemas:

Confirme que a função e os recursos apropriados estão habilitados. Consulte [Configuração do IIS](#).

## Caminho físico do site incorreto ou aplicativo ausente

- **Navegador:** 403 Proibido – acesso negado –OU– 403.14 Proibido – o servidor Web está configurado para não listar o conteúdo deste diretório.
- **Log do Aplicativo:** Nenhuma entrada
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log não é criado.
- **Log de depuração do módulo do ASP.NET Core:** O arquivo de log não é criado.

Solução de problemas:

Confira as **Configurações Básicas** no site do IIS e a pasta do aplicativo físico. Confirme que o aplicativo está na pasta no **Caminho físico** do site do IIS.

## Função incorreta, Módulo do ASP.NET Core Não Instalado ou

## permissões incorretas

- **Navegador:** 500.19 Erro interno do servidor – a página solicitada não pode ser acessada porque os dados de configuração relacionados da página são inválidos. **–OU–** Esta página não pode ser exibida
- **Log do Aplicativo:** Nenhuma entrada
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log não é criado.
- **Log de depuração do módulo do ASP.NET Core:** O arquivo de log não é criado.

Solução de problemas:

- Confirme que você habilitou a função apropriada. Consulte [Configuração do IIS](#).
- Abra **Programas e Recursos** ou **Aplicativos e Recursos** e confirme se a **Hospedagem do Windows Server** está instalada. Se a **Hospedagem do Windows Server** não estiver presente na lista de programas instalados, baixe e instale o Pacote de Hospedagem do .NET Core.

[Instalador de pacote de hospedagem do .NET Core atual \(download direto\)](#)

Para obter mais informações, confira [Instalar o pacote de hospedagem do .NET Core](#).

- Verifique se o **Pool de aplicativos > Modelo de processo > Identidade** está definido como **ApplicationPoolIdentity** ou se a identidade personalizada tem as permissões corretas para acessar a pasta de implantação do aplicativo.

processPath incorreto, variável de PATH ausente, pacote de hospedagem não instalado, sistema/IIS não reiniciado, Pacotes Redistribuíveis do VC++ não instalados ou violação de acesso de dotnet.exe

- **Navegador:** Erro HTTP 500.0 – Falha de carregamento de manipulador em processo ANCM
- **Log do Aplicativo:** Aplicativo 'MACHINE/WEBROOT/APPHOST/{ASSEMBLY}' com raiz física 'C:{PATH}' falhou ao iniciar o processo com a linha de comando '"..."', ErrorCode = '0x80070002 : 0. Não foi possível iniciar o aplicativo '{PATH}'. O executável não foi encontrado em '{PATH}'. Falha ao iniciar o aplicativo '/LM/W3SVC/2/ROOT', ErrorCode '0x8007023e'.
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log não é criado.
- **Log de depuração do módulo do ASP.NET Core:** Log de eventos: Não foi possível iniciar o aplicativo '{PATH}'. O executável não foi encontrado em '{PATH}'. HRESULT com falha retornou: 0x8007023e
- **Navegador:** Erro HTTP 502.5 – falha do processo
- **Log do Aplicativo:** Aplicativo 'MACHINE/WEBROOT/APPHOST/{ASSEMBLY}' com raiz física 'C:{PATH}' falhou ao iniciar o processo com a linha de comando '"..."', ErrorCode = '0x80070002 : 0.
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log é criado, mas vazio.

Solução de problemas:

- Confirme se o aplicativo é executado localmente no Kestrel. Uma falha do processo pode ser o resultado de um problema no aplicativo. Para obter mais informações, confira [Solução de problemas \(IIS\)](#) ou [Solução de problemas \(Serviço de Aplicativo do Azure\)](#).
- Verifique o atributo *processPath* no elemento `<aspNetCore>` em `web.config` para confirmar se ele é `dotnet` para uma FDD (implantação dependente de estrutura) ou `.\{ASSEMBLY}.exe` para uma [SCD \(implantação\)](#)

autossuficiente).

- Para uma FDD, o `dotnet.exe` pode não estar acessível por meio das configurações de PATH. Confirme se \*C:\Program Files\dotnet\* existe nas configurações de PATH do Sistema.
- Para uma FDD, o `dotnet.exe` pode não estar acessível para a identidade do usuário do pool de aplicativos. Confirme se a identidade do usuário do pool de aplicativos tem acesso ao diretório C:\Arquivos de Programas\dotnet. Confirme se não há nenhuma regra de negação configurada para a identidade do usuário do pool de aplicativos no C:\Arquivos de Programas\dotnet e nos diretórios do aplicativo.
- Talvez você tenha implantado uma FDD e instalado o .NET Core sem reiniciar o IIS. Reinicie o servidor ou o IIS executando **net stop was /y** seguido por **net start w3svc** em um prompt de comando.
- Você pode ter implantado uma FDD sem instalar o tempo de execução do .NET Core no sistema de hospedagem. Se o tempo de execução do .NET Core ainda não foi instalado, execute o **Instalador do Pacote de Hospedagem do .NET Core** no sistema.

[Instalador de pacote de hospedagem do .NET Core atual \(download direto\)](#)

Para obter mais informações, confira [Instalar o pacote de hospedagem do .NET Core](#).

Se um tempo de execução específico for necessário, baixe o tempo de execução dos [Arquivos de Download do .NET](#) e instale-o no sistema. Conclua a instalação reiniciando o sistema ou o IIS executando **net stop was /y** seguido por **net start w3svc** em um prompt de comando.

- Talvez você tenha implantado uma FDD e os *Pacotes redistribuíveis do Microsoft Visual C++ 2015 (x64)* não estejam instalados no sistema. Obtenha um instalador do [Centro de Download da Microsoft](#).

## Argumentos incorretos do elemento <aspNetCore>

- **Navegador:** Erro HTTP 500.0 – Falha de carregamento de manipulador em processo ANCM
- **Log do Aplicativo:** A invocação do hostfxr para encontrar o manipulador de solicitação inprocess falha sem encontrar nenhuma dependência nativa. Isso provavelmente significa que o aplicativo está configurado incorretamente, verifique as versões do Microsoft.NetCore.App e Microsoft.AspNetCore.App que são afetadas pelo aplicativo e estão instaladas no computador. Não foi possível localizar o manipulador de solicitação inprocess. Saída capturada da invocação do hostfxr: Você quis dizer executar comandos do SDK do dotnet? Instale o SDK do dotnet de: <https://go.microsoft.com/fwlink/?LinkID=798306&clcid=0x409> Falha ao iniciar o aplicativo '/LM/W3SVC/3/ROOT', ErrorCode '0x8000ffff'.
- **Log de stdout do Módulo do ASP.NET Core:** Você quis dizer executar comandos do SDK do dotnet? Instale o SDK do dotnet de: <https://go.microsoft.com/fwlink/?LinkID=798306&clcid=0x409>
- **Log de depuração do módulo do ASP.NET Core:** A invocação do hostfxr para encontrar o manipulador de solicitação inprocess falha sem encontrar nenhuma dependência nativa. Isso provavelmente significa que o aplicativo está configurado incorretamente, verifique as versões do Microsoft.NetCore.App e Microsoft.AspNetCore.App que são afetadas pelo aplicativo e estão instaladas no computador. HRESULT com falha retornou: 0x8000ffff Não foi possível localizar o manipulador de solicitação inprocess. Saída capturada da invocação do hostfxr: Você quis dizer executar comandos do SDK do dotnet? Instale o SDK do dotnet de: <https://go.microsoft.com/fwlink/?LinkID=798306&clcid=0x409> HRESULT com falha retornou: 0x8000ffff
- **Navegador:** Erro HTTP 502.5 – falha do processo
- **Log do Aplicativo:** Aplicativo 'MACHINE/WEBROOT/APPHOST/{ASSEMBLY}' com a raiz física 'C:{PATH}' falha ao iniciar o processo com a linha de comando '"dotnet" .{ASSEMBLY}.dll', ErrorCode = '0x80004005: 80008081'.

- **Log de stdout do Módulo do ASP.NET Core:** O aplicativo a ser executado não existe: 'PATH\{ASSEMBLY}.dll'

Solução de problemas:

- Confirme se o aplicativo é executado localmente no Kestrel. Uma falha do processo pode ser o resultado de um problema no aplicativo. Para obter mais informações, confira [Solução de problemas \(IIS\)](#) ou [Solução de problemas \(Serviço de Aplicativo do Azure\)](#).
- Examine o atributo *arguments* no elemento `<aspNetCore>` no `web.config` para confirmar se ele: (a) é `.\{ASSEMBLY}.dll` de uma FDD (implantação dependente de estrutura); ou (b) não está presente, é uma cadeia de caracteres vazia (`arguments=""`) ou uma lista de argumentos do aplicativo (`arguments="{ARGUMENT_1}, {ARGUMENT_2}, ... {ARGUMENT_X}"`) para uma SCD (implantação autossuficiente).

## Estrutura compartilhada do .NET Core ausente

- **Navegador:** Erro HTTP 500.0 – Falha de carregamento de manipulador em processo ANCM
- **Log do Aplicativo:** A invocação do hostfxr para encontrar o manipulador de solicitação inprocess falha sem encontrar nenhuma dependência nativa. Isso provavelmente significa que o aplicativo está configurado incorretamente, verifique as versões do Microsoft.AspNetCore.App e Microsoft.AspNetCore.App que são afetadas pelo aplicativo e estão instaladas no computador. Não foi possível localizar o manipulador de solicitação inprocess. Saída capturada da invocação do hostfxr: Não foi possível encontrar nenhuma versão de estrutura compatível. A estrutura especificada 'Microsoft.AspNetCore.App', versão '{VERSION}', não foi encontrada.

Falha ao iniciar o aplicativo '/LM/W3SVC/5/ROOT', ErrorCode '0x8000ffff'.

- **Log de stdout do Módulo do ASP.NET Core:** Não foi possível encontrar nenhuma versão de estrutura compatível. A estrutura especificada 'Microsoft.AspNetCore.App', versão '{VERSION}', não foi encontrada.
- **Log de depuração do módulo do ASP.NET Core:** HRESULT com falha retornou: 0x8000ffff

Solução de problemas:

Para uma FDD (implantação dependente de estrutura), confirme se você tem o tempo de execução correto instalado no sistema.

## Pool de aplicativos interrompido

- **Navegador:** 503 Serviço Não Disponível
- **Log do Aplicativo:** Nenhuma entrada
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log não é criado.
- **Log de depuração do módulo do ASP.NET Core:** O arquivo de log não é criado.

Solução de problemas:

Confirme que o Pool de Aplicativos não está no estado *Parado*.

## O subaplicativo inclui uma seção <manipuladores>

- **Navegador:** Erro HTTP 500.19 – Erro Interno do Servidor
- **Log do Aplicativo:** Nenhuma entrada
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log do aplicativo raiz é criado e mostra uma

operação normal. O arquivo de log do subaplicativo não é criado.

- **Log de depuração do módulo do ASP.NET Core:** O arquivo de log do aplicativo raiz é criado e mostra uma operação normal. O arquivo de log do subaplicativo não é criado.

Solução de problemas:

Confirme se o arquivo `web.config` do subaplicativo não inclui uma seção `<handlers>` ou que o subaplicativo não herda os manipuladores do aplicativo pai.

A seção `<system.webServer>` do aplicativo pai de `web.config` é colocada dentro de um elemento `<location>`. A propriedade `InheritInChildApplications` é definida como `false` para indicar que as configurações especificadas no elemento `<location>` não são herdadas por aplicativos que residem em um subdiretório do aplicativo pai. Para obter mais informações, consulte [Módulo do ASP.NET Core](#).

Confirme se o arquivo `web.config` do subaplicativo não inclui uma seção `<handlers>`.

## caminho do log de stdout incorreto

- **Navegador:** O aplicativo responde normalmente.
- **Log do Aplicativo:** Não foi possível iniciar o redirecionamento de stdout em C:\Arquivos de Programas\IIS\Asp.Net Core Module\V2\aspnetcorev2.dll. Mensagem de exceção: HRESULT 0x80070005 retornado em {PATH}\aspnetcoremodulev2\commonlib\fileoutputmanager.cpp:84. Não foi possível parar o redirecionamento de stdout em C:\Arquivos de Programas\IIS\Asp.Net Core Module\V2\aspnetcorev2.dll. Mensagem de exceção: HRESULT 0x80070002 retornado em {PATH}. Não foi possível iniciar o redirecionamento de stdout em {PATH}\aspnetcorev2\_inprocess.dll.
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log não é criado.
- **Log de depuração do módulo do ASP.NET Core:** Não foi possível iniciar o redirecionamento de stdout em C:\Arquivos de Programas\IIS\Asp.Net Core Module\V2\aspnetcorev2.dll. Mensagem de exceção: HRESULT 0x80070005 retornado em {PATH}\aspnetcoremodulev2\commonlib\fileoutputmanager.cpp:84. Não foi possível parar o redirecionamento de stdout em C:\Arquivos de Programas\IIS\Asp.Net Core Module\V2\aspnetcorev2.dll. Mensagem de exceção: HRESULT 0x80070002 retornado em {PATH}. Não foi possível iniciar o redirecionamento de stdout em {PATH}\aspnetcorev2\_inprocess.dll.
- **Log do Aplicativo:** Aviso: Não foi possível criar `stdoutLogFile \?` {PATH}\path\_doesnt\_exist\stdout\_{PROCESS ID}{TIMESTAMP}.log, ErrorCode = -2147024893.
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log não é criado.

Solução de problemas:

- O caminho `stdoutLogFile` especificado no elemento `<aspNetCore>` de `web.config` não existe. Para obter mais informações, confira [Módulo ASP.NET Core: criação de log e redirecionamento](#).
- O usuário do pool de aplicativos não tem acesso de gravação para o caminho do log de stdout.

## Problema geral de configuração do aplicativo

- **Navegador:** Erro HTTP 500.0 – Falha de carregamento de manipulador em processo ANCM –**OU**– Erro HTTP 500.30 – Falha de início no processo do ANCM
- **Log do Aplicativo:** Variável
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log é criado, mas vazio, ou criado com entradas normais até o ponto da falha do aplicativo.

- **Log de depuração do módulo do ASP.NET Core:** Variável
- **Navegador:** Erro HTTP 502.5 – falha do processo
- **Log do Aplicativo:** Aplicativo 'MACHINE/WEBROOT/APPHOST/{ASSEMBLY}' com raiz física 'C:{PATH}' criada no processo com a linha de comando '"C:{PATH}{ASSEMBLY}.{exe|dll}" ', mas falhou, não respondeu ou não escutou na porta '{PORT}' fornecida, ErrorCode = '{ERROR CODE}'
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log é criado, mas vazio.

Solução de problemas:

O processo não pôde ser iniciado, provavelmente, devido a um problema de programação ou configuração do aplicativo.

Para mais informações, consulte os seguintes tópicos:

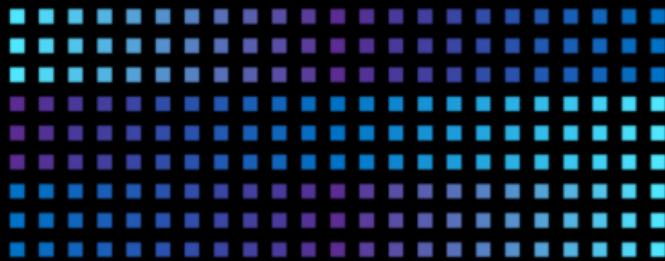
- [Solucionar problemas do ASP.NET Core no IIS](#)
- [Solucionar problemas no ASP.NET Core no Serviço de Aplicativo do Azure](#)
- [Solucionar problemas de projetos do ASP.NET Core](#)

# DevOps com ASP.NET Core e Azure

10/12/2018 • 3 minutes to read • [Edit Online](#)



# DevOps with ASP.NET Core and Azure



Cam Soper  
Scott Addie

Microsoft Corporation

Por [Cam Soper](#) e [Scott Addie](#)

Este guia está disponível como um [e-book em PDF para download](#).

## Bem-vindo

Bem-vindo ao guia de ciclo de vida de desenvolvimento do Azure para .NET. Este guia apresenta os conceitos básicos da criação de um ciclo de vida de desenvolvimento para o Azure usando processos e ferramentas do .NET. Depois de concluir-lo, você aproveitará os benefícios de uma cadeia de ferramentas madura do DevOps.

## A quem esse guia se destina

Você deve ser um desenvolvedor experiente do ASP.NET Core (nível 200 a 300). Você não precisa saber nada sobre o Azure, uma vez que abordaremos isso nesta introdução. Esse guia também pode ser útil para engenheiros de DevOps que estão mais focados nas operações do que no desenvolvimento.

Esse guia destina-se aos desenvolvedores do Windows. No entanto, Linux e macOS são totalmente compatíveis com .NET Core. Para adaptar esse guia para Linux/macOS, fique atento a textos explicativos para diferenças do Linux/macOS.

## O que este guia não aborda

Esse guia se concentra em uma experiência de implantação contínua de ponta a ponta para desenvolvedores do .NET. Ele não é um guia completo para tudo relacionado ao Azure e não se concentra extensivamente em APIs .NET para serviços do Azure. A ênfase está totalmente concentrada na integração contínua, na implantação, no monitoramento e na depuração. No final do guia, são oferecidas recomendações para as próximas etapas. Serviços da plataforma do Azure que úteis para desenvolvedores do ASP.NET Core estão incluídos nas sugestões.

## O que há neste guia

### [Ferramentas e downloads](#)

Saiba onde obter as ferramentas usadas neste guia.

### [Implantar no Serviço de Aplicativo](#)

Aprenda os vários métodos para implantar um aplicativo ASP.NET Core no Serviço de Aplicativo do Azure.

### [Integração contínua e implantação](#)

Crie uma solução de implantação e integração contínua de ponta a ponta para seu aplicativo ASP.NET Core com o GitHub, o Azure DevOps Services e o Azure.

### [Monitorar e depurar](#)

Use as ferramentas do Azure para monitorar, solucionar problemas e ajustar seu aplicativo.

### [Próximas etapas](#)

Outros roteiros de aprendizado para desenvolvedores do ASP.NET Core que estão aprendendo sobre o Azure.

## Leitura adicional de introdução

Se esta é sua primeira experiência com a computação em nuvem, estes artigos explicam os conceitos básicos.

- [O que é a computação em nuvem?](#)
- [Exemplos de computação em nuvem](#)
- [O que é IaaS?](#)
- [O que é PaaS?](#)

# Ferramentas e downloads

12/12/2018 • 2 minutes to read • [Edit Online](#)

O Azure tem várias interfaces para provisionamento e gerenciamento de recursos, como o [portal do Azure](#), [CLI do Azure](#), [Azure PowerShell](#), [nuvem do Azure Shell](#) ou Visual Studio. Este guia usa uma abordagem minimalista e usa o Azure Cloud Shell sempre que possível para reduzir as etapas necessárias. No entanto, o portal do Azure deve ser usado para algumas partes.

## Pré-requisitos

As assinaturas a seguir são necessárias:

- Azure — se você não tiver uma conta [Obtenha uma avaliação gratuita](#).
- Os serviços do Azure DevOps — sua assinatura de DevOps do Azure e a organização é criado no capítulo 4.
- GitHub — se você não tiver uma conta [Inscreve-se gratuitamente](#).

As ferramentas a seguir são necessárias:

- [Git](#) — um entendimento fundamental do Git é recomendado para este guia. Examine os [documentação do Git](#), especificamente [git remoto](#) e [git push](#).
- [SDK do .NET core](#) — versão 2.1.300 ou posterior é necessário para compilar e executar o aplicativo de exemplo. Se o Visual Studio é instalado com o [desenvolvimento de plataforma cruzada do .NET Core](#) carga de trabalho, o SDK do .NET Core já está instalada.

Verifique se a instalação do SDK do .NET Core. Abra um shell de comando e execute o seguinte comando:

```
dotnet --version
```

## Ferramentas recomendadas (somente Windows)

- [Visual Studio](#) robustas ferramentas do Azure fornecem uma interface gráfica para a maioria das funcionalidades descritas neste guia. Qualquer edição do Visual Studio funcionará, incluindo o Visual Studio Community Edition gratuito. Os tutoriais são gravados para demonstrar o desenvolvimento, implantação e operações de desenvolvimento com e sem o Visual Studio.

Confirme se o Visual Studio tem o seguinte [cargas de trabalho](#) instalado:

- Desenvolvimento do ASP.NET e para a Web
- Desenvolvimento do Azure
- Desenvolvimento de plataforma cruzada do .NET Core

# Implantar um aplicativo de serviço de aplicativo

12/12/2018 • 15 minutes to read • [Edit Online](#)

O [serviço de aplicativo do Azure](#) é plataforma de hospedagem de web do Azure. Implantar um aplicativo web no serviço de aplicativo do Azure pode ser feito manualmente ou por um processo automatizado. Esta seção do guia discute métodos de implantação que podem ser disparados manualmente ou por script usando a linha de comando ou disparada manualmente usando o Visual Studio.

Nesta seção, você vai realizar as seguintes tarefas:

- Baixe e compile o aplicativo de exemplo.
- Crie um aplicativo serviço de aplicativo Web usando o Azure Cloud Shell.
- Implante o aplicativo de exemplo no Azure usando Git.
- Implante uma alteração no aplicativo usando o Visual Studio.
- Adicione um slot de preparo para o aplicativo web.
- Implante uma atualização para o slot de preparo.
- Troca os slots de preparo e produção.

## Baixar e testar o aplicativo

O aplicativo usado neste guia é um aplicativo ASP.NET Core pré-criados [simples de leitor de Feed](#). É um aplicativo páginas Razor que usa o `Microsoft.SyndicationFeed.ReaderWriter` API para recuperar um feed RSS/Atom e exibir os itens de notícias em uma lista.

Fique à vontade examinar o código, mas é importante entender que não há nada de especial sobre este aplicativo. Ele é apenas um aplicativo ASP.NET Core simple para fins ilustrativos.

Em um shell de comando, baixe o código, compile o projeto e executá-lo da seguinte maneira.

*Observação: Os usuários do Linux/macOS devem fazer as alterações apropriadas para caminhos, por exemplo, usando a barra invertida (/) em vez de barra invertida (\).*

1. Clone o código para uma pasta no seu computador local.

```
git clone https://github.com/Azure-Samples/simple-feed-reader/
```

2. Alterar pasta de trabalho para o *leitor de feeds simples* pasta que foi criada.

```
cd .\simple-feed-reader\SimpleFeedReader
```

3. Restaure os pacotes e compile a solução.

```
dotnet build
```

4. Execute o aplicativo.

```
dotnet run
```

```
Command Prompt - dotnet run  
C:\Src\simple-feed-reader\SimpleFeedReader>dotnet run  
Hosting environment: Production  
Content root path: C:\Src\simple-feed-reader\SimpleFeedReader  
Now listening on: http://localhost:5000  
Application started. Press Ctrl+C to shut down.
```

5. Abra um navegador e navegue até <http://localhost:5000>. O aplicativo permite que você digite ou cole um URL de feed de distribuição e exibir uma lista de itens de notícias.

Title	Published
Penny Pinching in the Cloud: Deploying Containers cheaply to Azure	6/15/2018 11:50:21 PM +00:00
EarTrumpet 2.0 makes Windows 10's audio subsystem even better...and it's free!	6/13/2018 7:34:00 AM +00:00
ASP.NET Core Architect David Fowler's hidden gems in 2.1	6/11/2018 5:15:24 AM +00:00
Carriage Returns and Line Feeds will ultimately bite you - Some Git Tips	6/5/2018 11:37:47 PM +00:00

6. Quando estiver satisfeito o aplicativo está funcionando corretamente, desligá-lo pressionando **Ctrl+C** no shell de comando.

## Criar o aplicativo Web do serviço de aplicativo do Azure

Para implantar o aplicativo, você precisará criar um serviço de aplicativo [aplicativo Web](#). Após a criação do aplicativo Web, você implantará a ele em seu computador local usando o Git.

1. Entrar para o [Azure Cloud Shell](#). Observação: Quando você entra pela primeira vez, o Cloud Shell solicita a criar uma conta de armazenamento para arquivos de configuração. Aceite os padrões ou forneça um nome exclusivo.
2. Use o Cloud Shell para as etapas a seguir.
  - a. Declare uma variável para armazenar o nome do seu aplicativo web. O nome deve ser exclusivo a ser usado na URL padrão. Usando o `$RANDOM` função Bash para construir o nome garante a exclusividade e resulta no formato `webappname99999`.

```
webappname=mywebapp$RANDOM
```

- b. Crie um grupo de recursos. Grupos de recursos fornecem um meio para agrregar os recursos do Azure para ser gerenciado como um grupo.

```
az group create --location centralus --name AzureTutorial
```

O `az` comando invoca o [CLI do Azure](#). A CLI pode ser executada localmente, mas usá-lo no Cloud Shell economiza tempo e configuração.

- c. Crie um plano de serviço de aplicativo na camada S1. Um plano do serviço de aplicativo é um agrupamento de aplicativos web que compartilham o mesmo tipo de preço. A camada S1 não é gratuita, mas ela é necessária para o recurso de slots de preparo.

```
az appservice plan create --name $webappname --resource-group AzureTutorial --sku S1
```

- d. Crie recurso de aplicativo web usando o plano de serviço de aplicativo no mesmo grupo de recursos.

```
az webapp create --name $webappname --resource-group AzureTutorial --plan $webappname
```

- e. Defina as credenciais de implantação. Essas credenciais de implantação se aplicam a todos os aplicativos web em sua assinatura. Não use caracteres especiais no nome do usuário.

```
az webapp deployment user set --user-name REPLACE_WITH_USER_NAME --password REPLACE_WITH_PASSWORD
```

- f. Configurar o aplicativo web para aceitar as implantações do Git local e a exibição de *URL de implantação do Git*. **Anote essa URL para referência posterior.**

```
echo Git deployment URL: $(az webapp deployment source config-local-git --name $webappname --resource-group AzureTutorial --query url --output tsv)
```

- g. Exibição de *URL do aplicativo web*. Navegue até essa URL para ver o aplicativo web em branco. **Anote essa URL para referência posterior.**

```
echo Web app URL: http://$webappname.azurewebsites.net
```

3. Usando um shell de comando no computador local, navegue até a pasta do projeto do aplicativo web (por exemplo, `.\simple-feed-reader\SimpleFeedReader`). Execute os seguintes comandos para configurar o Git para enviar por push para a URL de implantação:

- a. Adicione a URL remota no repositório local.

```
git remote add azure-prod GIT_DEPLOYMENT_URL
```

- b. Enviar por push o local *mestre* ramificar para o *azure prod* do remote *mestre* branch.

```
git push azure-prod master
```

Você será solicitado para as credenciais de implantação que você criou anteriormente. Observe a saída no

shell de comando. Azure cria o aplicativo ASP.NET Core remotamente.

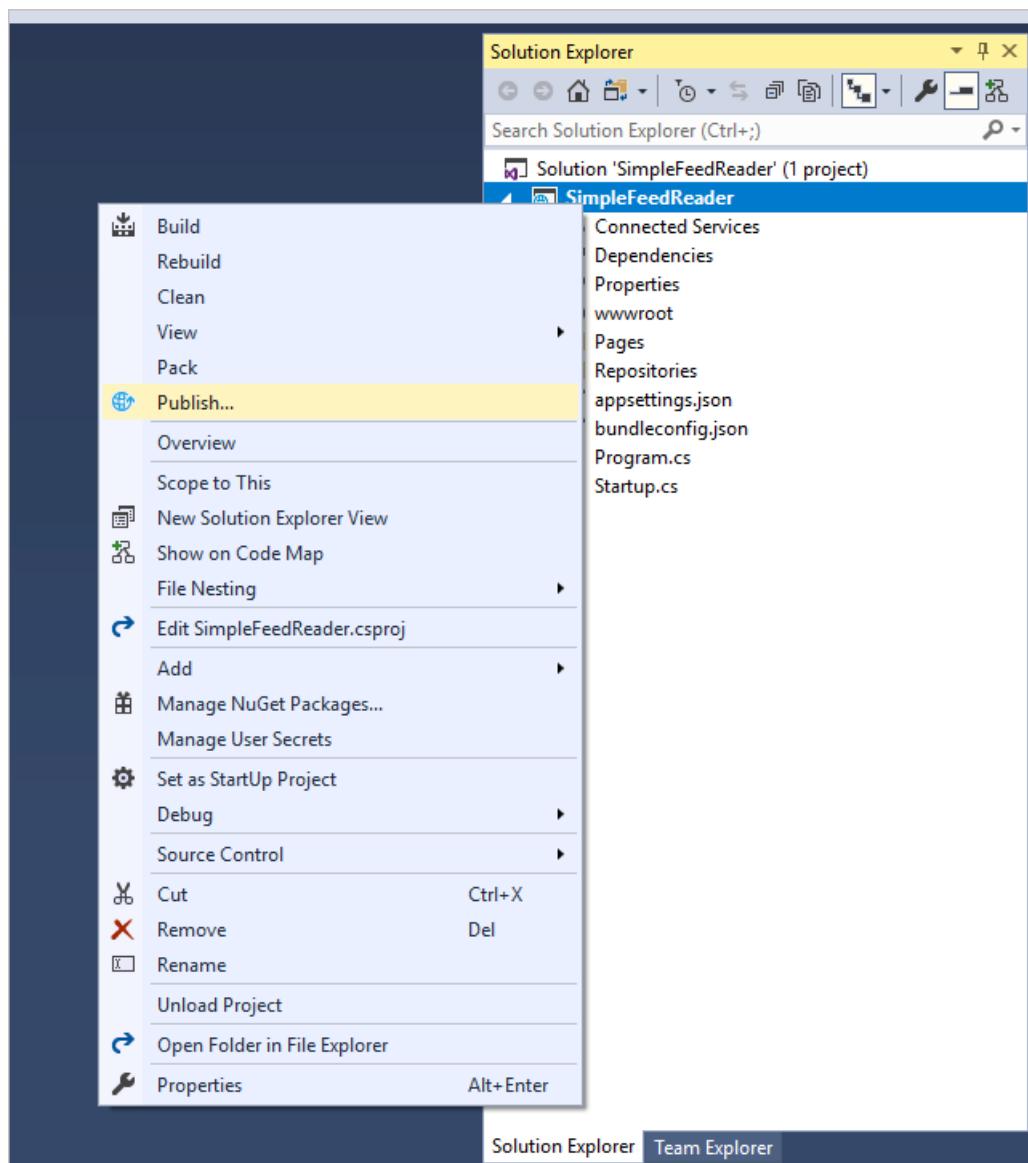
- Em um navegador, navegue até a *URL do aplicativo Web* e observe o aplicativo foi compilado e implantado. Alterações adicionais podem ser confirmadas no repositório Git local com `git commit`. Essas alterações são enviadas por push para o Azure com o anterior `git push` comando.

## Implantação com o Visual Studio

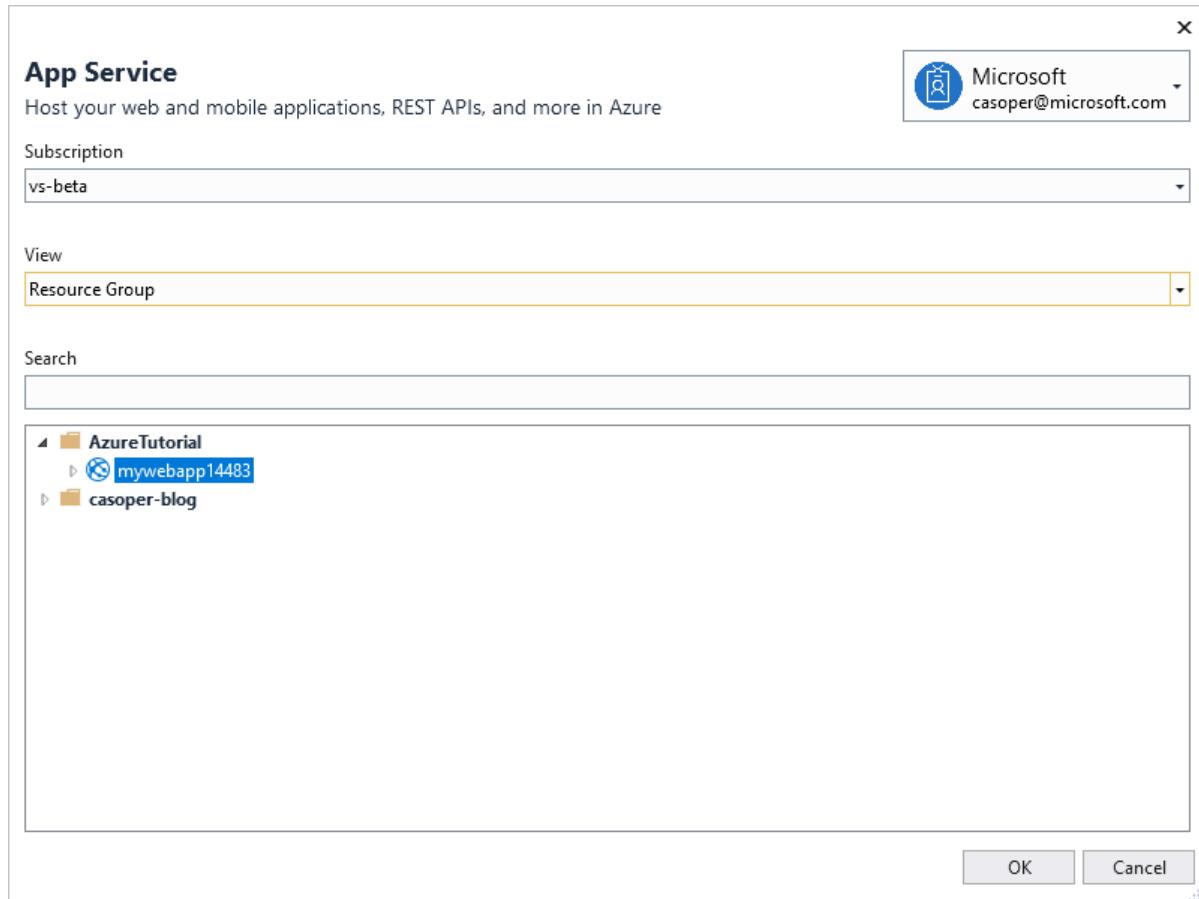
*Observação: Esta seção se aplica somente ao Windows. Os usuários do Linux e macOS devem fazer a alteração descrita na etapa 2 abaixo. Salve o arquivo e confirmar a alteração no repositório local com `git commit`. Por fim, envie por push a alteração com `git push`, como na primeira seção.*

O aplicativo já foi implantado no shell de comando. Vamos usar ferramentas integradas do Visual Studio para implantar uma atualização para o aplicativo. Nos bastidores, o Visual Studio realiza a mesma coisa, como a ferramentas de linha de comando, mas dentro da interface de usuário familiar do Visual Studio.

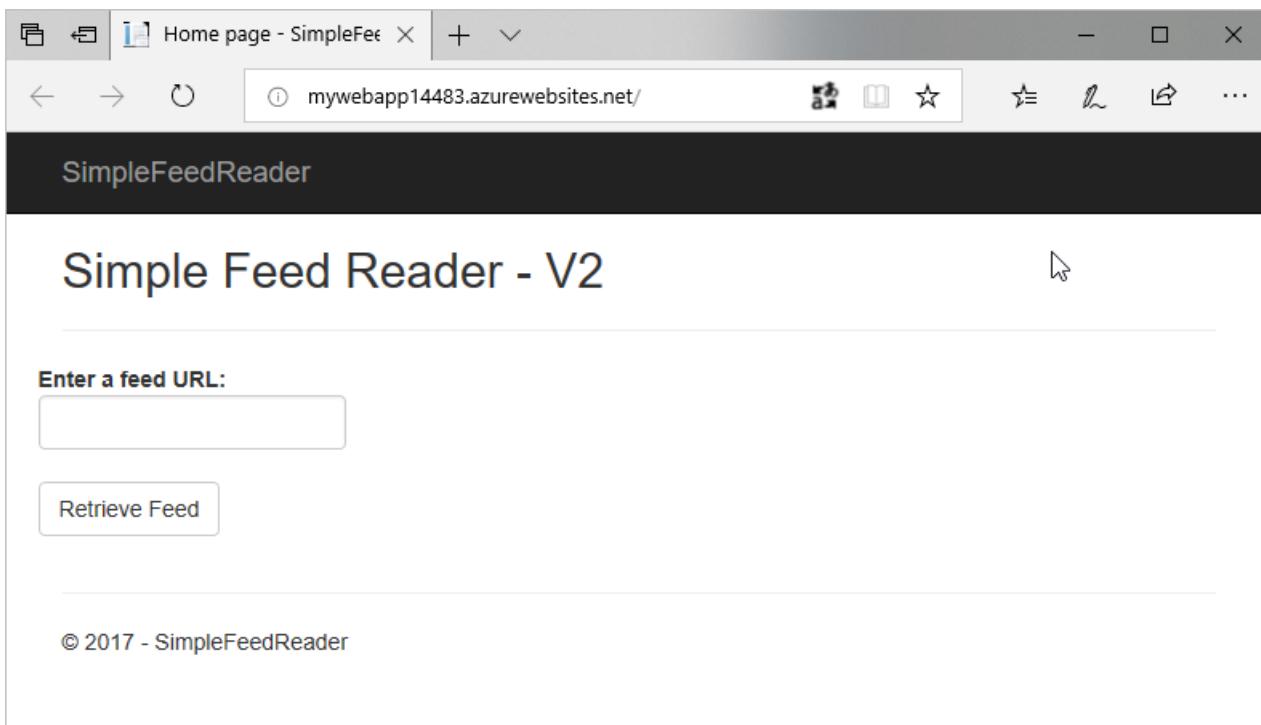
- Abra *SimpleFeedReader.sln* no Visual Studio.
- No Gerenciador de soluções, abra *Pages\Index.cshtml*. Alteração `<h2>Simple Feed Reader</h2>` para `<h2>Simple Feed Reader - V2</h2>`.
- Pressione **Ctrl+Shift+B** para compilar o aplicativo.
- No Gerenciador de soluções, clique com botão direito no projeto e clique em **publicar**.



5. Visual Studio pode criar um novo recurso do serviço de aplicativo, mas essa atualização será publicada a implantação existente. No **escolher um destino de publicação** caixa de diálogo, selecione **serviço de aplicativo** na lista à esquerda e, em seguida, selecione **selecionar existente**. Clique em **Publicar**.
6. No **serviço de aplicativo** caixa de diálogo, confirme que a Microsoft ou conta organizacional usada para criar sua assinatura do Azure é exibida no canto superior direito. Se não estiver, clique na lista suspensa e adicioná-lo.
7. Confirme se o Azure correto **assinatura** está selecionado. Para **modo de exibição**, selecione **grupo de recursos**. Expanda o **AzureTutorial** grupo de recursos e, em seguida, selecione o aplicativo web existente. Clique em **OK**.



Visual Studio compila e implanta o aplicativo do Azure. Navegue até a URL do aplicativo web. Validar que o `<h2>` modificação do elemento está ativo.



## Slots de implantação

Slots de implantação oferecem suporte à preparação de alterações sem afetar o aplicativo em execução na produção. Depois que a versão de preparada do aplicativo é validada por uma equipe de garantia de qualidade, slots de preparo e produção podem ser trocados. O aplicativo no ambiente de preparo é promovido para produção dessa maneira. As etapas a seguir criam um slot de preparo, implantar algumas alterações nele e trocar o slot de preparo em produção após a verificação.

1. Entrar para o [Azure Cloud Shell](#), se não estiver conectado.

2. Crie o slot de preparo.

a. Criar um slot de implantação com o nome *preparo*.

```
az webapp deployment slot create --name $webappName --resource-group AzureTutorial --slot staging
```

b. Configurar o slot de preparo para usar a implantação do Git local e obter o **preparo** URL de implantação.

**Anote essa URL para referência posterior.**

```
echo Git deployment URL for staging: $(az webapp deployment source config-local-git --name $webappName --resource-group AzureTutorial --slot staging --query url --output tsv)
```

c. Exiba a URL do slot de preparo. Navegue até a URL para ver o slot de preparo vazio. **Anote essa URL para referência posterior.**

```
echo Staging web app URL: http://$webappName-staging.azurewebsites.net
```

3. Em um editor de texto ou o Visual Studio, modifique *Pages* novamente para que o `<h2>` elemento lê `<h2>Simple Feed Reader - V3</h2>` e salve o arquivo.

4. Confirmar o arquivo para o repositório Git local, usando a **alterações** página no Visual Studio *Team Explorer* guia, ou digitando o seguinte usando o shell de comando do computador local:

```
git commit -a -m "upgraded to V3"
```

5. Usando o shell de comando do computador local, adicione a URL de implantação de preparo como um Git remoto e enviar por push as alterações confirmadas:

- a. Adicione a URL remota para a preparação para o repositório Git local.

```
git remote add azure-staging <Git_staging_deployment_URL>
```

- b. Enviar por push o local *mestre* ramificar para o *azure preparo* do remote *mestre* branch.

```
git push azure-staging master
```

Aguarde enquanto o Azure cria e implanta o aplicativo.

6. Para verificar se V3 foi implantado para o slot de preparo, abra duas janelas de navegador. Em uma janela, navegue até a URL do aplicativo web original. Em outra janela, navegue até a URL do aplicativo web preparo. A URL de produção serve V2 do aplicativo. A URL de preparo serve V3 do aplicativo.

The image contains two side-by-side screenshots of a web browser window. Both screenshots show the same web application, "SimpleFeedReader", but with different versions displayed.

**Screenshot 1 (Top):** The browser title bar says "Home page - SimpleFee". The address bar shows "mywebapp14483.azurewebsites.net/". The page content includes the heading "Simple Feed Reader - V2", a form with the placeholder "Enter a feed URL:", and a "Retrieve Feed" button. At the bottom, it says "© 2017 - SimpleFeedReader".

**Screenshot 2 (Bottom):** The browser title bar says "Home page - SimpleFee". The address bar shows "mywebapp14483-staging.azurewebsites.net/". The page content includes the heading "Simple Feed Reader - V3", a form with the placeholder "Enter a feed URL:", and a "Retrieve Feed" button. At the bottom, it says "© 2017 - SimpleFeedReader".

7. No Cloud Shell, troca o slot de preparo verificado/começando-up para produção.

```
az webapp deployment slot swap --name $webapppname --resource-group AzureTutorial --slot staging
```

8. Verifique se que a troca ocorreu ao atualizar as janelas do navegador de dois.

The image contains two side-by-side screenshots of a web browser window. Both screenshots show the same application, "Simple Feed Reader", but at different versions.

**Top Screenshot (Version V3):**

- The title bar says "Home page - SimpleFee x".
- The address bar shows "mywebapp14483.azurewebsites.net/".
- The main content area displays "Simple Feed Reader - V3".
- Below the title, there is a form with the label "Enter a feed URL:" followed by an empty input field.
- Below the input field is a "Retrieve Feed" button.
- At the bottom of the page, there is a copyright notice: "© 2017 - SimpleFeedReader".

**Bottom Screenshot (Version V2):**

- The title bar says "Home page - SimpleFee x".
- The address bar shows "mywebapp14483-staging.azurewebsites.net/".
- The main content area displays "Simple Feed Reader - V2".
- This screenshot is identical to the top one, except it does not contain the "Enter a feed URL:" form and input field.
- It also lacks the "Retrieve Feed" button.
- At the bottom of the page, there is a copyright notice: "© 2017 - SimpleFeedReader".

## Resumo

Nesta seção, as tarefas a seguir foram concluídas:

- Baixou e compilou o aplicativo de exemplo.
- Criado um aplicativo serviço de aplicativo Web usando o Azure Cloud Shell.
- Implantou o aplicativo de exemplo para o Azure usando Git.
- Implantada uma alteração para o aplicativo usando o Visual Studio.
- Adicionado um slot de preparo para o aplicativo web.
- Implantasse uma atualização para o slot de preparo.
- Trocado os slots de preparo e produção.

Na próxima seção, você aprenderá a criar um pipeline de DevOps com Pipelines do Azure.

## Leitura adicional

- [Visão geral de aplicativos Web](#)
- [Criar um aplicativo web .NET Core e o banco de dados SQL no serviço de aplicativo do Azure](#)
- [Configurar as credenciais de implantação do serviço de aplicativo do Azure](#)
- [Configurar ambientes de preparo no serviço de aplicativo do Azure](#)

# Integração contínua e implantação

09/12/2018 • 21 minutes to read • [Edit Online](#)

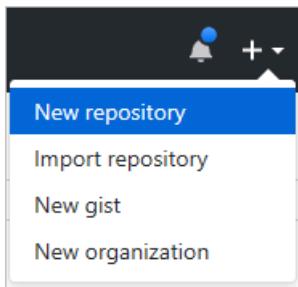
No capítulo anterior, você criou um repositório Git local para o aplicativo de leitor de Feed simples. Neste capítulo, você irá publicar esse código em um repositório do GitHub e construir um pipeline de serviços de DevOps do Azure usando os Pipelines do Azure. O pipeline permite compilações contínuas e implantações do aplicativo. Qualquer confirmação para o repositório GitHub dispara uma compilação e uma implantação em slot de preparo do aplicativo Web do Azure.

Nesta seção, você concluirá as seguintes tarefas:

- Publicar o código do aplicativo no GitHub
- Desconectar-se a implantação do Git local
- Criar uma organização de DevOps do Azure
- Criar um projeto de equipe nos serviços de DevOps do Azure
- Criar uma definição de compilação
- Criar um pipeline de lançamento
- Confirmar alterações no GitHub e implantar automaticamente no Azure
- Examinar o pipeline de Pipelines do Azure

## Publicar o código do aplicativo no GitHub

1. Abra uma janela do navegador e navegue até <https://github.com>.
2. Clique o + lista suspensa no cabeçalho e selecione **novo repositório**:



3. Selecione sua conta na **proprietário** lista suspensa e insira *leitor de feeds simples* no **nome do repositório** caixa de texto.
4. Clique o **criar repositório** botão.
5. Abra o shell de comando do seu computador local. Navegue até o diretório no qual o *leitor de feeds simples* repositório Git é armazenado.
6. Renomear existente *origin* remoto para *upstream*. Execute o seguinte comando:

```
git remote rename origin upstream
```

7. Adicione um novo *origem* remoto que aponta para a sua cópia do repositório no GitHub. Execute o seguinte comando:

```
git remote add origin https://github.com/<GitHub_username>/simple-feed-reader/
```

8. Publica seu repositório Git local para o repositório GitHub recém-criado. Execute o seguinte comando:

```
git push -u origin master
```

9. Abra uma janela do navegador e navegue até [https://github.com/<GitHub\\_username>/simple-feed-reader/](https://github.com/<GitHub_username>/simple-feed-reader/).

Valide que seu código aparece no repositório do GitHub.

## Desconectar-se a implantação do Git local

Remova a implantação Git local com as etapas a seguir. Pipelines do Azure (um serviço de DevOps do Azure) substitui e amplia essa funcionalidade.

1. Abra o [portal do Azure](#) e navegue até a *de preparo* (*mywebapp<unique\_number>/de preparo*) aplicativo Web. O aplicativo Web pode estar localizado rapidamente digitando *preparo* na caixa de pesquisa do portal:



2. Clique em **opções de implantação**. Um novo painel é exibido. Clique em **desconectar** para remover a configuração de controle de código-fonte Git local que foi adicionada no capítulo anterior. Confirme a operação de remoção clicando o **Sim** botão.
3. Navegue até a *mywebapp <unique\_number>* o serviço de aplicativo. Como lembrete, caixa de pesquisa do portal pode ser usada para localizar rapidamente o serviço de aplicativo.
4. Clique em **opções de implantação**. Um novo painel é exibido. Clique em **desconectar** para remover a configuração de controle de código-fonte Git local que foi adicionada no capítulo anterior. Confirme a operação de remoção clicando o **Sim** botão.

## Criar uma organização de DevOps do Azure

1. Abra um navegador e navegue até a [página de criação de organização de DevOps do Azure](#).
2. Digite um nome exclusivo para o **escolher um nome inesquecível** textbox para formar a URL para acessar a sua organização de DevOps do Azure.
3. Selecione o **Git** botão de opção, já que o código é hospedado em um repositório GitHub.
4. Clique no botão **Continue (Continuar)**. Após uma breve espera, uma conta e um projeto de equipe, chamado *MyFirstProject*, são criados.

Host my projects at:

Pick a memorable name [.visualstudio.com](#)

Manage code using:

-  Git
-  Team Foundation Version Control

We will host your projects in Central US location.  
You can share work with other Microsoft users.

[Change details](#)

[Continue](#)

5. Abra o email de confirmação indicando que a organização de DevOps do Azure e o projeto estão prontos para uso. Clique o **inicie seu projeto** botão:

Start your project



Plan, share code, collaborate, track progress, build, test, and deploy applications more efficiently, using Visual Studio, or your own tools

[Start your project](#) [Open in Visual Studio >](#)

6. Um navegador é aberto `<account_name>.visualstudio.com`. Clique o *MyFirstProject* link para começar a configurar o pipeline de DevOps do projeto.

## Configurar o pipeline de Pipelines do Azure

Há três etapas distintas para ser concluída. As etapas nos resultados em um pipeline de DevOps operacional três seções a seguir.

### DevOps do Azure conceda acesso ao repositório do GitHub

1. Expanda o **ou compilar o código de um repositório externo** accordion. Clique o **instalação compilar** botão:

or build code from an external repository

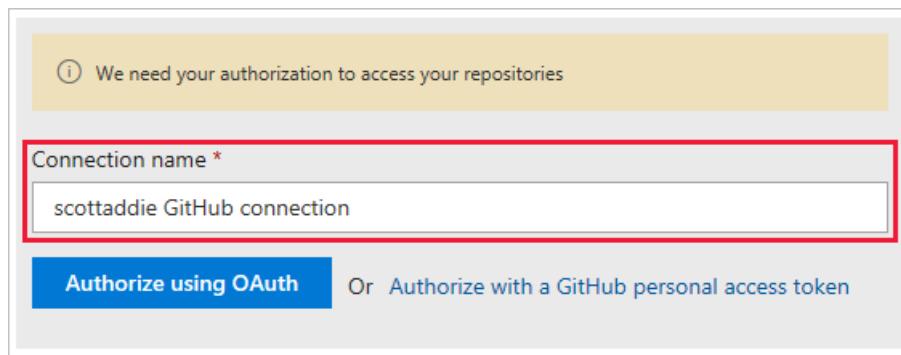
[Setup Build](#)

2. Selecione o **GitHub** opção a **selecionar uma fonte de** seção:

Select a source

VSTS Git	 GitHub	GitHub Enterprise	Subversion	Bitbucket® Cloud	External Git
----------	--------------------------------------------------------------------------------------------	-------------------	------------	------------------	--------------

3. A autorização é necessária antes de DevOps do Azure pode acessar seu repositório do GitHub. Insira < *GitHub\_username* > conexão do GitHub na **nome de Conexão** caixa de texto. Por exemplo:



4. Se a autenticação de dois fatores é ativada em sua conta do GitHub, um token de acesso pessoal é necessário. Clique nesse caso, o **autorizar com um token de acesso pessoal do GitHub** link. Consulte a [instruções oficiais de criação de token de acesso pessoal do GitHub](#) para obter ajuda. Somente o *repositório* escopo das permissões é necessária. Caso contrário, clique o **autorizar usando OAuth** botão.
5. Quando solicitado, entre sua conta do GitHub. Em seguida, selecione a autorização para conceder acesso a sua organização de DevOps do Azure. Se for bem-sucedido, um novo ponto de extremidade de serviço é criado.
6. Clique no botão de reticências ao lado de **repositório** botão. Selecione o < *GitHub\_username* > / *leitor de feeds simples* repositório na lista. Clique o **selecionar** botão.
7. Selecione o mestre ramificar a **branch padrão para compilações de manuais e programadas** lista suspensa. Clique no botão **Continue (Continuar)**. Página seleção de modelo é exibido.

#### Criar a definição de compilação

1. Na página seleção do modelo, insira *ASP.NET Core* na caixa de pesquisa:

The screenshot shows the 'Select a template' search interface. On the left, there's a search bar with 'Select a template' placeholder and an 'Empty pipeline' link. On the right, the search results for 'ASP.NET Core' are shown. A red box highlights the search term 'ASP.NET Core' in the search bar. Below the search bar, there's a 'Others' section with two items: 'ASP.NET Core' (dotnet icon) and 'ASP.NET Core (.NET Framework)' (Visual Studio icon).

2. Os resultados da pesquisa de modelo são exibidos. Passe o mouse sobre o **ASP.NET Core** modelo e clique no **aplicar** botão.
3. O **tarefas** guia da definição de compilação é exibida. Clique na guia **Gatilhos**.
4. Verifique as **habilitar a integração contínua** caixa. Sob o **filtros de ramificação** seção, confirme se o **tipo** lista suspensa é definida como *Include*. Defina as **especificação de Branch** lista suspensa para *mestre*.

Enable continuous integration

Batch changes while a build is in progress

Branch filters

Type

Include

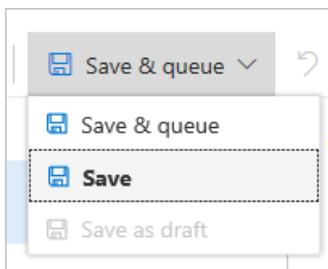
Branch specification

master

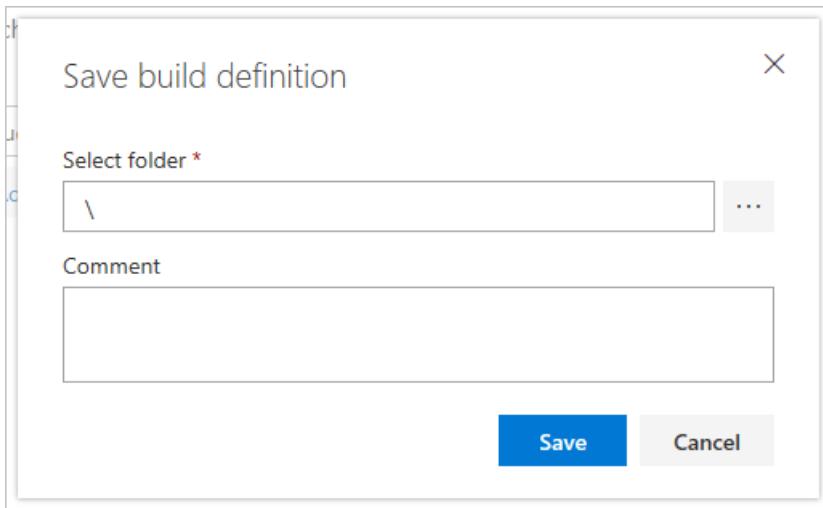
+ Add

Essas configurações fazem com que uma compilação disparar quando qualquer alteração é enviada por push para o *mestre* branch do repositório do GitHub. Integração contínua é testada na [confirmar alterações no GitHub e implantar automaticamente para o Azure](#) seção.

5. Clique o **salvar e enfileirar** botão e, em seguida, selecione o **salvar** opção:



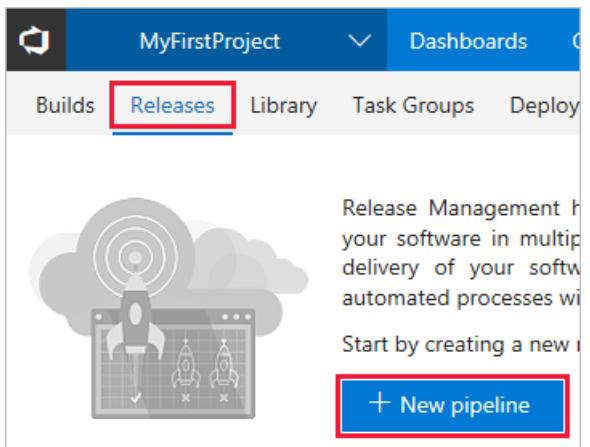
6. A seguinte caixa de diálogo modal será exibida:



Usar a pasta padrão de \ e clique no **salvar** botão.

#### Criar o pipeline de lançamento

1. Clique o **versões** guia de seu projeto de equipe. Clique o **novo pipeline** botão.



O painel de seleção de modelo é exibido.

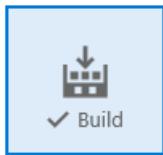
2. Na página seleção do modelo, insira *serviço de aplicativo* na caixa de pesquisa:

3. Os resultados da pesquisa de modelo são exibidos. Passe o mouse sobre o **implantação de serviço de aplicativo do Azure com o Slot** modelo e clique no **aplicar** botão. O **Pipeline** guia do pipeline de lançamento é exibida.

4. Clique o **Add** botão na **artefatos** caixa. O **adicionar um artefato** painel é exibido:

## Add artifact

Source type



GitHub



Team Found...

[3 more artifact types ▾](#)

Project \* ⓘ

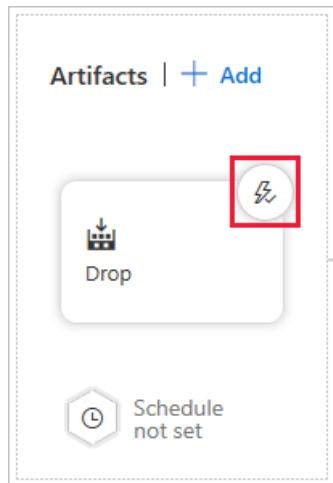
MyFirstProject

Source (Build definition) \* ⓘ

ⓘ This setting is required.

Add

5. Selecione o **construir** lado a lado do **tipo de fonte** seção. Esse tipo permite a vinculação do pipeline de lançamento para a definição de compilação.
6. Selecione *MyFirstProject* da **projeto** lista suspensa.
7. Selecione o nome de definição de compilação *MyFirstProject do ASP.NET Core-CI*, da **fonte (definição de compilação)** lista suspensa.
8. Selecione *mais recente* da **versão padrão** lista suspensa. Essa opção cria os artefatos produzidos pela execução mais recente da definição de compilação.
9. Substitua o texto na **alias de origem** caixa de texto com *Drop*.
10. Clique no botão **Adicionar**. O **artefatos** seção será atualizado para exibir as alterações.
11. Clique no ícone de raio para habilitar implantações contínuas:



Com essa opção habilitada, uma implantação ocorre sempre que uma nova compilação está disponível.

12. Um **gatilho de implantação contínua** painel aparece à direita. Clique no botão de alternância para habilitar o recurso. Não é necessário habilitar o **gatilho de solicitação de Pull**.
13. Clique o **Add** na lista suspensa a **criar filtros de ramificação** seção. Escolha o **ramificação de padrão da definição de compilação** opção. Esse filtro faz com que a versão disparar apenas para uma compilação do

repositório do GitHub *mestre* branch.

14. Clique no botão **Salvar**. Clique o **Okey** botão na resultante **salvar** caixa de diálogo modal.
15. Clique o **ambiente 1** caixa. Uma **ambiente** painel aparece à direita. Alterar o *ambiente 1* texto na **nome do ambiente** caixa de texto *produção*.

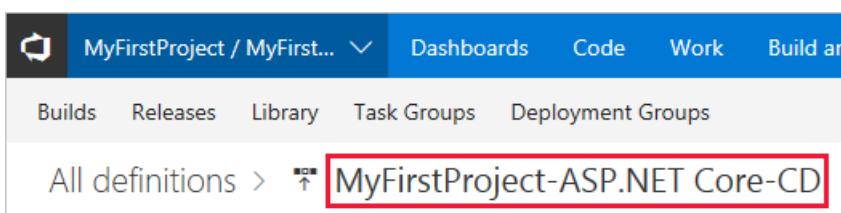


16. Clique o **1 fase, 2 tarefas** link na **produção** caixa:



O **tarefas** guia do ambiente é exibida.

17. Clique o **implantar o serviço de aplicativo do Azure ao Slot** tarefa. Suas configurações aparecem em um painel à direita.
18. Selecione a assinatura do Azure associada com o serviço de aplicativo da **assinatura do Azure** lista suspensa. Depois de selecionadas, clique o **autorizar** botão.
19. Selecione *aplicativo Web* da **tipo de aplicativo** lista suspensa.
20. Selecione *mywebapp / <unique\_number>* da **nome do serviço de aplicativo** lista suspensa.
21. Selecione *AzureTutorial* da **grupo de recursos** lista suspensa.
22. Selecione *preparo* da **Slot** lista suspensa.
23. Clique no botão **Salvar**.
24. Passe o mouse sobre o nome do pipeline de versão padrão. Clique no ícone de lápis para editá-lo. Use *MyFirstProject* do *ASP.NET Core-CD* como o nome.



25. Clique no botão **Salvar**.

## Confirmar alterações no GitHub e implantar automaticamente no Azure

1. Abra *SimpleFeedReader.sln* no Visual Studio.
2. No Gerenciador de soluções, abra *Pages\Index.cshtml*. Alteração `<h2>Simple Feed Reader - V3</h2>` para `<h2>Simple Feed Reader - V4</h2>`.
3. Pressione **Ctrl+Shift+B** para compilar o aplicativo.
4. Confirme o arquivo para o repositório GitHub. Use o **alterações** página no Visual Studio *Team Explorer* guia ou execute o seguinte usando o shell de comando do computador local:

```
git commit -a -m "upgraded to V4"
```

5. Enviar por push a alteração na *mestre* ramificar para o *origem* remoto do seu repositório GitHub:

```
git push origin master
```

A confirmação é exibida no repositório do GitHub *mestre* branch:

The screenshot shows a GitHub repository interface. At the top, there are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. Below this, a commit message is displayed: 'scottaddie upgraded to V4' with a timestamp 'Latest commit bb0d47d 3 minutes ago'.

A compilação é disparada, como integração contínua está habilitada na definição de build **gatilhos** guia:

The screenshot shows the 'Triggers' tab in the Azure DevOps pipeline configuration. Under the 'Continuous integration' section, there is a row for 'scottaddie/simple-feed-reader' with the status 'Enabled'. A checkbox labeled 'Enable continuous integration' is checked and highlighted with a red box.

6. Navegue até a **enfileirado** guia da **Pipelines do Azure > compilações** página nos serviços de DevOps do Azure. Compilação enfileirada mostra a ramificação e confirme que disparou a compilação:

The screenshot shows a table titled 'Queued or running' listing a single pipeline item. The columns are: Name, Definition name, Queue name, Source, Source version, and Date queued. The item listed is '20180614.1' with 'MyFirstProject-ASP.NET Core-Cl' as the definition name, 'Hosted VS2017' as the queue name, 'master' as the source, 'bb0d47d' as the source version, and 'a minute ago' as the date queued.

7. A compilação for bem-sucedida, ocorre uma implantação do Azure. Navegue até o aplicativo no navegador. Observe que o texto "V4" é exibido no título:

The screenshot shows a web application titled "Simple Feed Reader - V4". At the top, there is a header bar with the title. Below it, a form has a label "Enter a feed URL:" followed by a text input field. Underneath the input field is a button labeled "Retrieve Feed". At the bottom of the page, there is a copyright notice: "© 2017 - SimpleFeedReader".

## Examinar o pipeline de Pipelines do Azure

### Definição de compilação

Uma definição de compilação foi criada com o nome *MyFirstProject* do *ASP.NET Core-CI*. Após a conclusão, a compilação produz um *.zip* arquivo, incluindo os ativos a serem publicadas. O pipeline de lançamento implanta esses ativos do Azure.

A definição de compilação **tarefas** guia lista as etapas individuais que está sendo usadas. Há cinco tarefas de compilação.

The screenshot shows the Azure DevOps Pipeline definition. The pipeline consists of the following tasks:

- Process**: Build process
- Get sources**: scottaddie/simple-feed-reader, master
- Phase 1**: Run on agent
  - Restore**: .NET Core
  - Build**: .NET Core
  - Test**: .NET Core
  - Publish**: .NET Core
  - Publish Artifact**: Publish Build Artifacts

1. **Restaure** — executa o `dotnet restore` comando restaurar pacotes do NuGet do aplicativo. O pacote padrão feed usado é nuget.org.
2. **Crie** — executa o `dotnet build --configuration release` comando para compilar o código do aplicativo. Isso `--configuration` opção é usada para produzir uma versão otimizada do código, que é adequado para implantação em um ambiente de produção. Modificar a *BuildConfiguration* variável na definição de compilação **variáveis** guia se, por exemplo, uma configuração de depuração é necessária.
3. **Teste** — executa o

```
dotnet test --configuration release --logger trx --results-directory <local_path_on_build_agent>
```

comando para executar testes de unidade do aplicativo. Testes de unidade são executados em qualquer linguagem c# projeto de correspondência a `**/*Tests/*.csproj` padrão glob. Resultados de teste são salvos em um `trx` arquivo no local especificado pelo `--results-directory` opção. Se qualquer teste falhar, a compilação falhará e não será implantada.

#### NOTE

Para verificar se o trabalho de testes de unidade, modifique `SimpleFeedReader.Tests\Services\NewsServiceTests.cs` propositalmente quebrar um dos testes. Por exemplo, altere `Assert.True(result.Count > 0);` à `Assert.False(result.Count > 0);` no `Returns_News_Stories_Given_Valid_Url` método. Confirme e envie a alteração ao GitHub. A compilação é disparada e falha. O status do pipeline de compilação muda para **falha**. Reverta a alteração, a confirmação e o envio por push novamente. A compilação for bem-sucedida.

4. **Publique** — executa o `dotnet publish --configuration release --output <local_path_on_build_agent>` comando para produzir uma `.zip` arquivo com os artefatos a serem implantados. O `--output` opção especifica o local de publicação do `.zip` arquivo. Se o local é especificado passando um **variável predefinida** denominado `$(build.artifactstagingdirectory)`. Essa variável se expande para um caminho local, como `c:\agent_work\1\a`, no agente de compilação.
5. **Publicar artefato** — Publishes as `.zip` produzido pelo arquivo o **publicar** tarefa. A tarefa aceita o `.zip` local como um parâmetro, que é a variável predefinida do arquivo `$(build.artifactstagingdirectory)`. O `.zip` arquivo é publicado como uma pasta chamada `drop`.

Clique na definição de compilação **resumo** link para exibir um histórico das compilações com a definição:



Na página resultante, clique no link correspondente ao número de build exclusivo:

A screenshot of the Azure DevOps build summary page for build #20180525.1. The page has tabs for 'Summary', 'History', and 'Deleted'. The 'Summary' tab is active. It shows details about the repository (scottaddie/simple-feed-reader), default queue (Hosted VS2017), queue status (Enabled), and last updated by (Scott Addie). The 'Queued &amp; running' section indicates 'No builds queued or running at the moment'. The 'Recently completed' section lists build #20180525.1, which succeeded on master branch by Scott Addie. The 'Analytics' section shows a success rate of 100.00% with 1 build.

Um resumo dessa compilação específica é exibido. Clique o **artefatos** guia e observe o `drop` produzida pela compilação de pasta é listada:

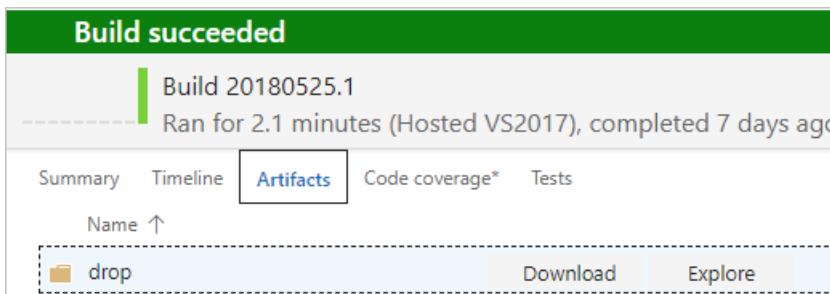
**Build succeeded**

Build 20180525.1  
Ran for 2.1 minutes (Hosted VS2017), completed 7 days ago

Summary Timeline Artifacts Code coverage\* Tests

Name ↑

drop Download Explore



Use o **Baixe e explorar** links para inspecionar os artefatos publicados.

### Pipeline de lançamento

Um pipeline de lançamento foi criado com o nome *MyFirstProject* do *ASP.NET Core-CD*:

Pipeline Tasks Variables Retention Options History

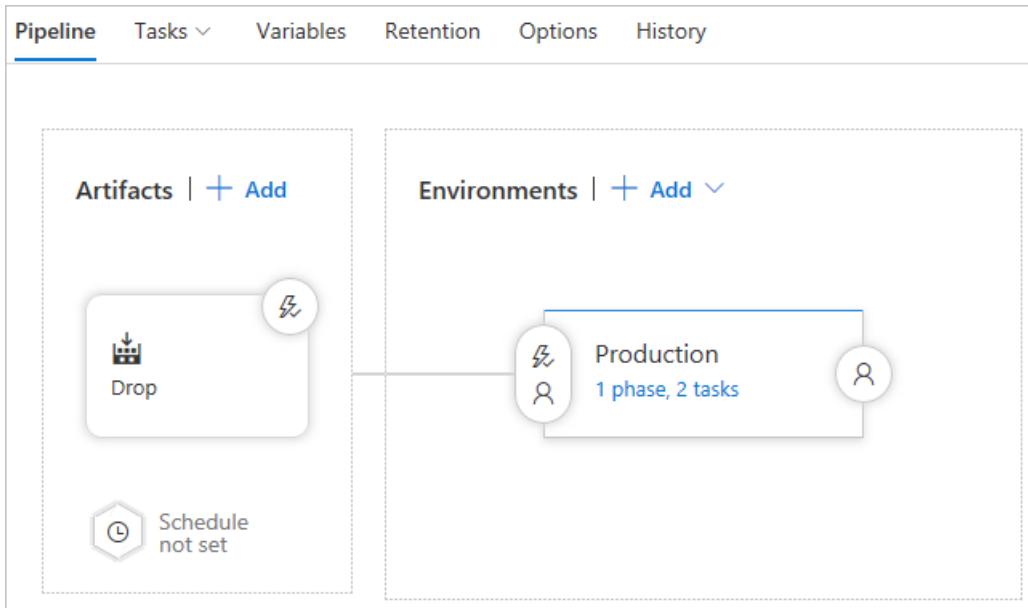
Artifacts | + Add

Drop

Schedule not set

Environments | + Add

Production 1 phase, 2 tasks



Os dois principais componentes do pipeline de lançamento são as **artefatos** e o **ambientes**. Clicando na caixa na **artefatos** seção revela o seguinte painel:

Pipeline Tasks Variables Retention Options History

Artifacts | + Add

Drop

Schedule not set

Environments

Artifact

Build - Drop

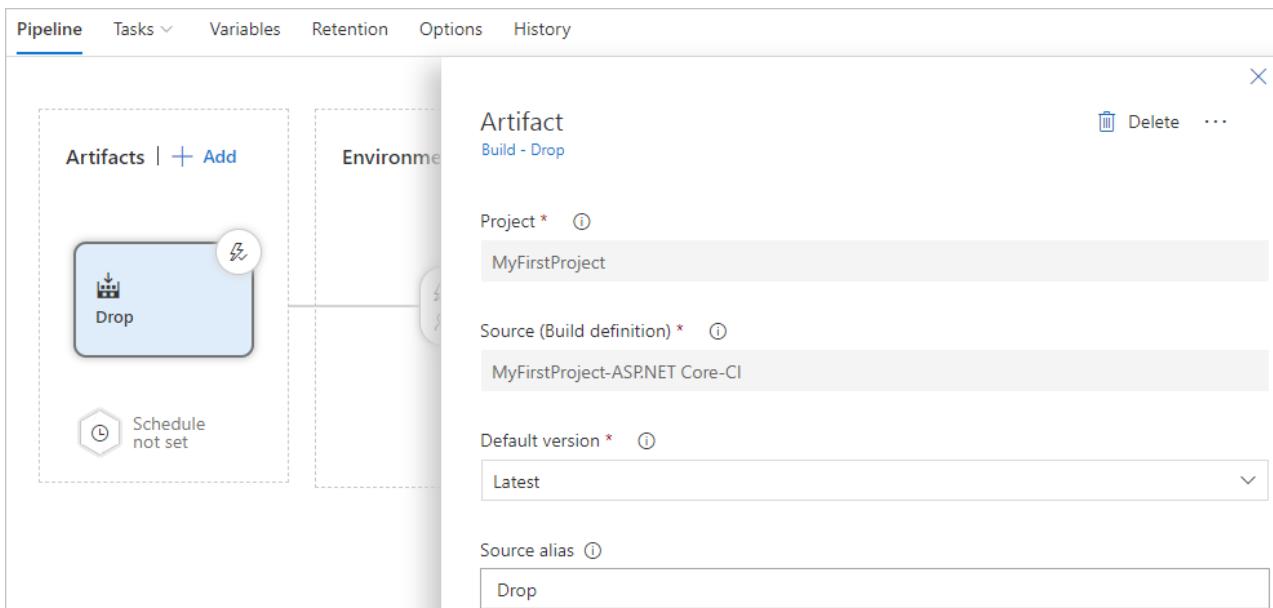
Delete ...

Project \* ⓘ MyFirstProject

Source (Build definition) \* ⓘ MyFirstProject-ASP.NET Core-CD

Default version \* ⓘ Latest

Source alias ⓘ Drop



O **fonte (definição de compilação)** valor representa a definição de compilação ao qual esse pipeline de versão está vinculada. O .zip arquivo produzido por uma execução bem-sucedida da definição de compilação é fornecido para o *produção* ambiente para implantação no Azure. Clique o *1 fase, 2 tarefas* link na *produção* caixa do ambiente para exibir as tarefas de pipeline de lançamento:

O pipeline de lançamento consiste em duas tarefas: *implantar o serviço de aplicativo do Azure ao Slot* e *gerenciar o serviço de aplicativo Azure - Slot de troca*. Clicar a primeira tarefa revela a configuração de tarefa a seguir:

Version	3.*
Display name *	
Deploy Azure App Service to Slot	
Azure subscription *	<a href="#">Manage</a>
Visual Studio Enterprise	<input type="button" value=""/>
App type *	
Web App	
App Service name *	
mywebapp11857	
<input checked="" type="checkbox"/> Deploy to slot	
Resource group *	
AzureTutorial	
Slot *	
staging	

A assinatura do Azure, o tipo de serviço, o nome do aplicativo web, o grupo de recursos e o slot de implantação são definidos na tarefa de implantação. O **pacote ou pasta** caixa de texto contém o .zip caminho do arquivo a ser extraído e implantado para o *preparo* slot do *mywebapp<exclusivo número>* aplicativo web.

Clicar a tarefa de troca de slot revela a configuração de tarefa a seguir:

Azure App Service Manage ⓘ

Version 0.\* ▾

Display name \*

Manage Azure App Service - Slot Swap

Azure subscription \* ⓘ | Manage ↗

Visual Studio Enterprise ▾ ⏪

Action ⓘ

Swap Slots

App Service name \* ⓘ

mywebapp11857

Resource group \* ⓘ

AzureTutorial

Source Slot \* ⓘ

staging

Swap with Production ⓘ

Preserve Vnet ⓘ

A assinatura, grupo de recursos, tipo de serviço, nome do aplicativo web e detalhes do slot de implantação são fornecidas. O **troca com produção** caixa de seleção está marcada. Consequentemente, os bits implantados para o *preparo* slot são trocadas no ambiente de produção.

## Leitura adicional

- [Criar seu primeiro pipeline com o Azure Pipelines](#)
- [Projeto de compilação e .NET Core](#)
- [Implantar um aplicativo web com Pipelines do Azure](#)

# Monitorar e depurar

12/12/2018 • 10 minutes to read • [Edit Online](#)

Tendo implantado o aplicativo e criado um pipeline de DevOps, é importante entender como monitorar e solucionar problemas com o aplicativo.

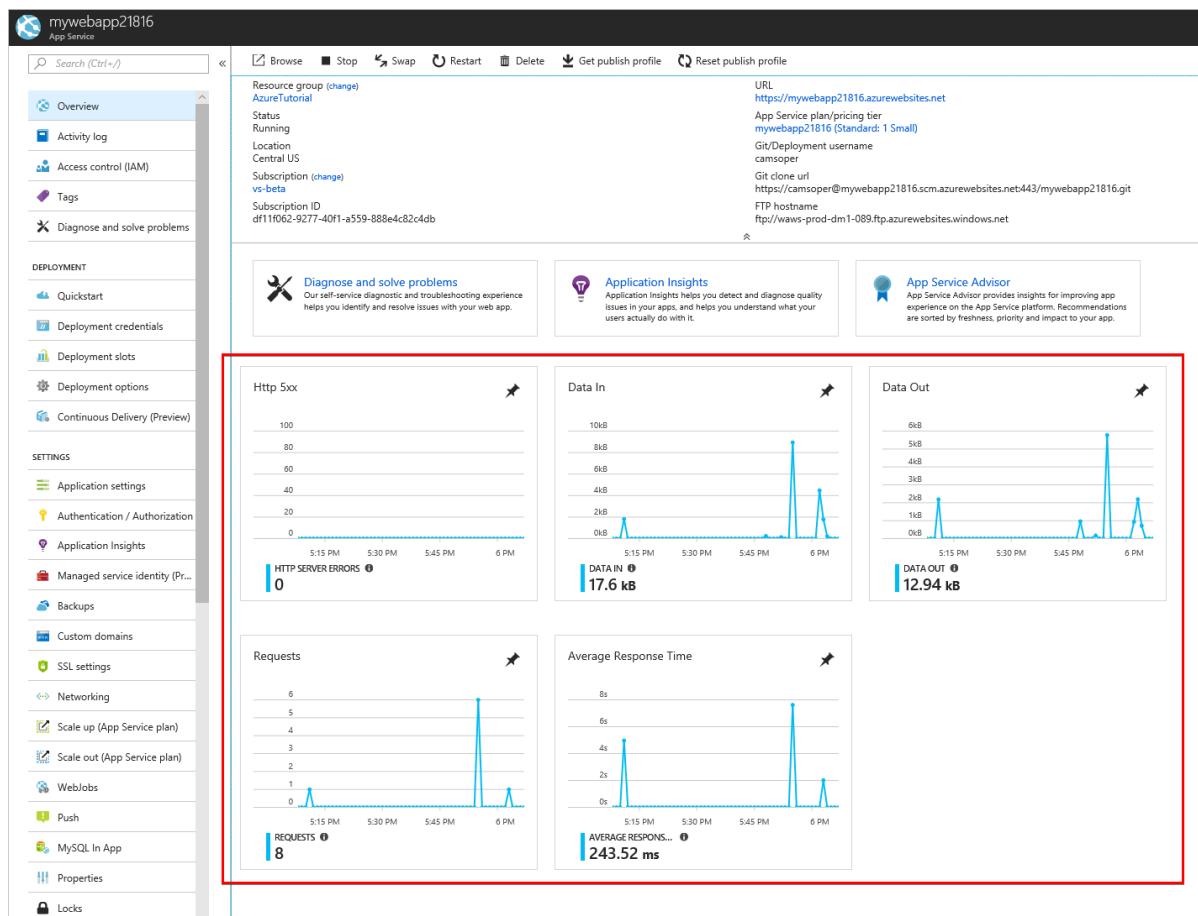
Nesta seção, você concluirá as seguintes tarefas:

- Localizar básicos de monitoramento e solução de problemas de dados no portal do Azure
- Saiba como o Azure Monitor fornece uma análise mais profunda das métricas entre todos os serviços do Azure
- Conectar o aplicativo web com o Application Insights para a criação de perfil de aplicativo
- Ativar o registro em log e saiba onde baixar logs
- Stream logs em tempo real
- Saiba onde configurar alertas
- Saiba mais sobre aplicativos de web do serviço de aplicativo do Azure depuração remotos.

## Solução de problemas e monitoramento básico

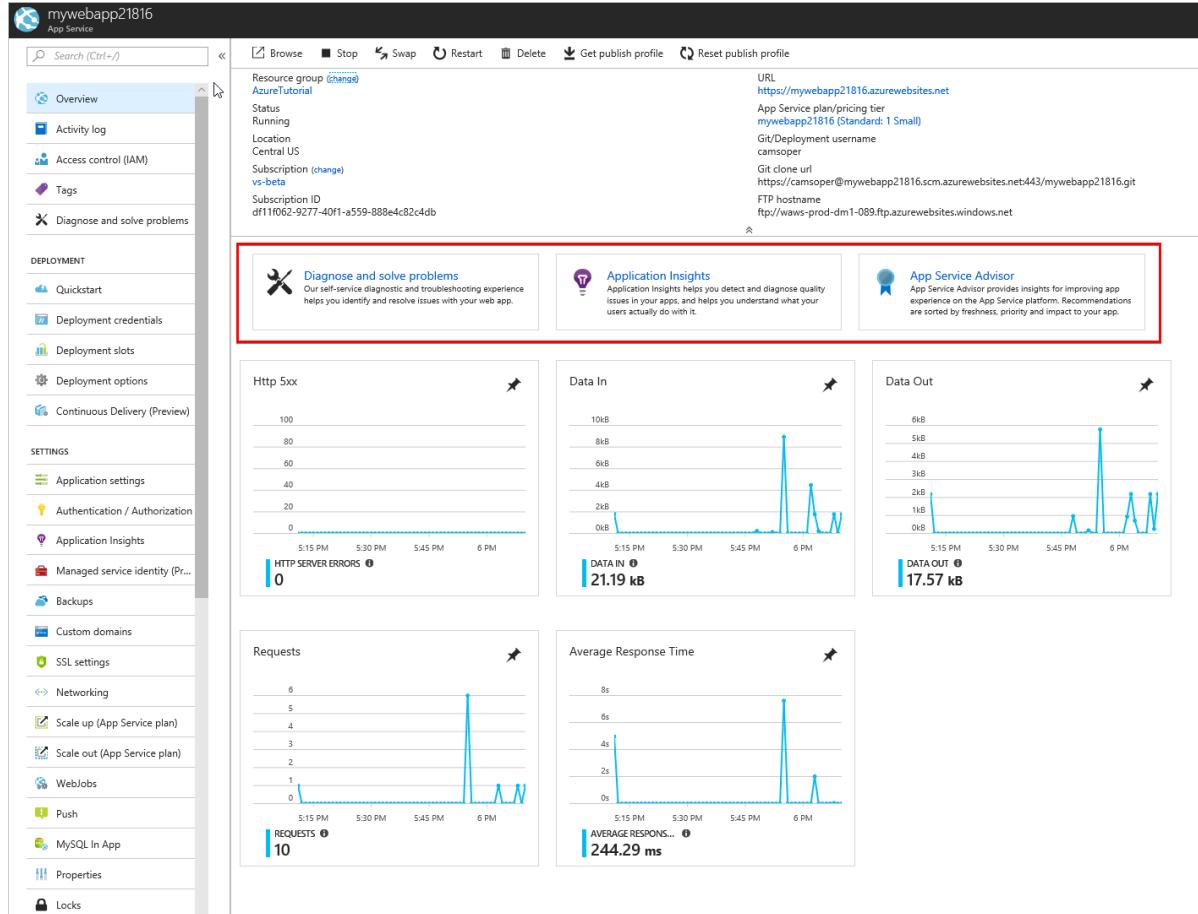
Com facilidade, aplicativos web do serviço de aplicativo são monitorados em tempo real. O portal do Azure processa as métricas em gráficos fáceis de entender e.

1. Abra o [portal do Azure](#), em seguida, navegue até a `mywebapp<unique_number>` o serviço de aplicativo.
2. O **visão geral** guia exibe informações úteis de "instantâneo", incluindo gráficos que exibem as métricas recentes.



- **Http 5xx:** Contagem de erros do lado do servidor, geralmente exceções no código do ASP.NET Core.
- **Dados em:** Entrada de dados que chegam ao seu aplicativo web.
- **Saída de dados:** Saída de dados do seu aplicativo web aos clientes.
- **Solicitações:** Contagem de solicitações HTTP.
- **Tempo médio de resposta:** Tempo médio para o aplicativo web responder às solicitações HTTP.

Várias ferramentas de autoatendimento para otimização e solução de problemas também são encontradas nesta página.



- **Diagnosticar e resolver problemas** é uma solução de problemas de autoatendimento.
- **Application Insights** é para criação de perfil de desempenho e o comportamento do aplicativo e será discutido posteriormente nesta seção.
- **Assistente do serviço de aplicativo** faz recomendações para ajustar sua experiência de aplicativo.

## Monitoramento avançado

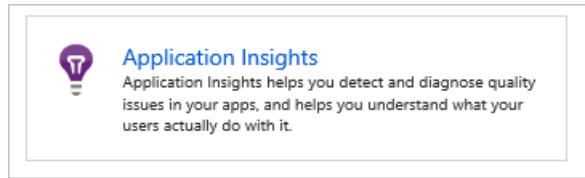
O [Azure Monitor](#) é o serviço centralizado para todas as métricas de monitoramento e configuração de alertas em todos os serviços do Azure. Dentro do Azure Monitor, os administradores podem controlar o desempenho granularmente e identificar tendências. Cada serviço do Azure oferece seu próprio [conjunto de métricas](#) para o Azure Monitor.

## Perfil com o Application Insights

[Application Insights](#) é um serviço do Azure para analisar o desempenho e estabilidade de aplicativos web e como os usuários usá-los. Os dados do Application Insights são mais amplos e aprofundados do que o Azure Monitor. Os dados podem fornecer os desenvolvedores e administradores com informações de chave para aprimorar aplicativos. Application Insights podem ser adicionados a um recurso de serviço de aplicativo do Azure sem alterações de código.

1. Abra o [portal do Azure](#), em seguida, navegue até a `mywebapp<unique_number>` o serviço de aplicativo.

2. Dos visão geral , clique o **Application Insights** lado a lado.



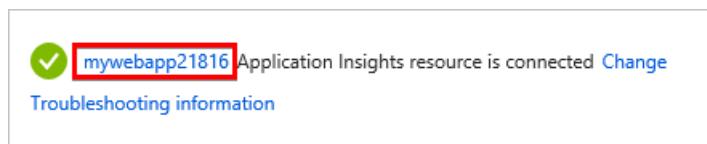
3. Selecione o **criar novo recurso** botão de opção. Use o nome de recurso padrão e selecione o local para o recurso Application Insights. O local não precisa corresponder ao que um aplicativo web.

The screenshot shows the 'Create new resource' dialog for Application Insights. It includes fields for 'New resource name' (set to 'mywebapp21816'), 'Location' (set to 'East US'), and 'Runtime/Framework' (set to 'ASP.NET Core'). The 'Create new resource' option is selected. The 'OK' button at the bottom is highlighted with a red box.

4. Para **tempo de execução/estrutura**, selecione **ASP.NET Core**. Aceite as configurações padrão.

5. Selecione **OK**. Se solicitado a confirmar, selecione**continuar**.

6. Depois que o recurso tiver sido criado, clique no nome do recurso do Application Insights para navegar diretamente até a página do Application Insights.



Como o aplicativo é usado, os dados acumulam. Selecione **Refresh** para recarregar a folha com novos dados.

The screenshot shows the Microsoft Application Insights Overview page for the application 'mywebapp21816'. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, INVESTIGATE (Application map, Smart Detection, Live Metrics Stream, Metrics Explorer, Metrics (preview), Search, Availability, Failures, Performance, Servers, Browser, Workbooks (preview)), and USAGE (PREVIEW) (Users, Sessions, Events, Funnels, User Flows, Retention, Impact, Cohorts). The main area features a purple banner with the message 'Please try new Overview before it becomes the default experience.' Below this is a section titled 'Essentials' with cards for Alerts (0), Live Stream (Click to configure), Users (1), Smart Detection (0 Detections (7d)), Availability (--), and App map. The 'Health' section includes an 'Overview timeline' chart for 'MYWEBAPP21816' showing server response times from 0ms to 2s, with a peak at 1.53s. It also displays page view load time (--), server requests (39), and failed requests (11). The bottom section shows a bar chart titled 'Total of Server Requests by Request Performance' for 'MYWEBAPP21816', detailing the distribution of requests across different performance ranges.

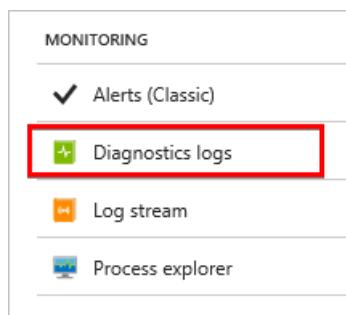
REQUEST PERFORMANCE	TOTAL	% TOTAL
<250ms	17	43.6%
250ms-500ms	8	20.5%
1sec-3sec	6	15.4%
500ms-1sec	4	10.3%
7sec-15sec	2	5.1%
15sec-30sec	1	2.6%
3sec-7sec	1	2.6%

O Application Insights fornece informações úteis do lado do servidor sem nenhuma configuração adicional. Para obter o máximo valor do Application Insights [instrumentar seu aplicativo com o SDK do Application Insights](#). Quando configurado corretamente, o serviço fornece monitoramento de ponta a ponta entre o servidor web e o navegador, incluindo desempenho do lado do cliente. Para obter mais informações, consulte o [documentação do Application Insights](#).

## Registrando em log

Logs de servidor e aplicativo da Web estão desabilitados por padrão no serviço de aplicativo do Azure. Habilite os logs com as seguintes etapas:

1. Abra o [portal do Azure](#) e navegue até a `mywebapp<unique_number>` o serviço de aplicativo.
2. No menu à esquerda, role para baixo até a **monitoramento** seção. Selecione **logs de diagnóstico**.



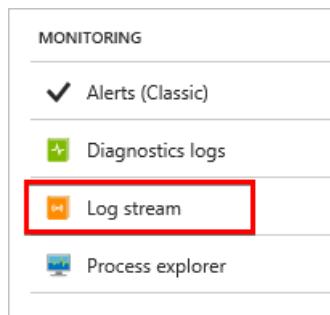
3. Ative **log de aplicativo (Filesystem)**. Se solicitado, clique na caixa para instalar as extensões para habilitar o registro em log no aplicativo web do aplicativo.
4. Definir **log do servidor Web à sistema de arquivos**.
5. Insira o **período de retenção** em dias. Por exemplo, 30.
6. Clique em **Salvar**.

Logs do servidor (serviço de aplicativo) do ASP.NET Core e da web são gerados para o aplicativo web. Eles podem ser baixados usando as informações de FTP/FTPS exibidas. A senha é o mesmo que as credenciais de implantação criadas anteriormente neste guia. Os logs podem ser [transmitido diretamente ao seu computador local com o PowerShell ou CLI do Azure](#). Os logs também podem ser [exibidos no Application Insights](#).

## Streaming de log

Logs do servidor web e de aplicativo podem ser transmitidos em tempo real por meio do portal.

1. Abra o [portal do Azure](#) e navegue até a `mywebapp<unique_number>` o serviço de aplicativo.
2. No menu à esquerda, role para baixo até a **Monitoring** seção e selecione **fluxo de Log**.



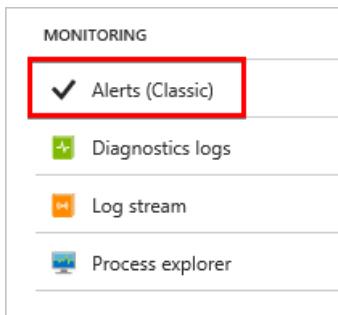
Os logs também podem ser [transmitido por meio da CLI do Azure ou o Azure PowerShell](#), incluindo por meio do Cloud Shell.

## Alertas

O Azure Monitor também fornece [alertas em tempo real](#) com base em métricas, eventos administrativos e outros critérios.

*Observação: No momento, alertas em métricas do aplicativo web só está disponível no serviço alertas (clássico).*

O [alertas de serviço \(clássico\)](#) pode ser encontrada no Azure Monitor ou sob o **monitoramento** seção das configurações de serviço de aplicativo.



## Depuração ao vivo

O serviço de aplicativo do Azure pode ser [depurado remotamente com o Visual Studio](#) quando os logs não fornecer informações suficientes. No entanto, a depuração remota exige o aplicativo a ser compilada com símbolos de depuração. Depuração não deve ser feita em produção, exceto como último recurso.

## Conclusão

Nesta seção, você concluiu as seguintes tarefas:

- Localizar básicos de monitoramento e solução de problemas de dados no portal do Azure
- Saiba como o Azure Monitor fornece uma análise mais profunda das métricas entre todos os serviços do Azure
- Conectar o aplicativo web com o Application Insights para a criação de perfil de aplicativo
- Ativar o registro em log e saiba onde baixar logs
- Stream logs em tempo real
- Saiba onde configurar alertas
- Saiba mais sobre aplicativos de web do serviço de aplicativo do Azure depuração remotos.

## Leitura adicional

- [Solucionar problemas no ASP.NET Core no Serviço de Aplicativo do Azure](#)
- [Referência de erros comuns para o Serviço de Aplicativo do Azure e o IIS com o ASP.NET Core](#)
- [Monitorar o desempenho do aplicativo web do Azure com o Application Insights](#)
- [Habilitar log de diagnósticos para aplicativos Web no Serviço de Aplicativo do Azure](#)
- [Solucionar problemas de um aplicativo Web no Serviço de Aplicativo do Azure usando o Visual Studio](#)
- [Criar alertas de métrica clássicos no Azure Monitor para serviços do Azure - portal do Azure](#)

# Próximas etapas

12/12/2018 • 3 minutes to read • [Edit Online](#)

Neste guia, você criou um pipeline de DevOps para um aplicativo de exemplo do ASP.NET Core. Parabéns! Esperamos que você tenha gostado de aprendizado publicar aplicativos web ASP.NET Core no serviço de aplicativo do Azure e automatizar a integração contínua das alterações.

Além de hospedagem na web e DevOps, o Azure tem uma ampla gama de serviços de plataforma-como um serviço (PaaS) útil para desenvolvedores do ASP.NET Core. Esta seção fornece uma visão geral de alguns dos serviços mais comumente usados.

## Armazenamento e bancos de dados

[Cache redis](#) está disponível como um serviço de cache de dados de alta taxa de transferência e baixa latência. Ele pode ser usado para armazenamento em cache de saída de página, reduzindo as solicitações de banco de dados e fornecendo o estado de sessão do ASP.NET Core em várias instâncias de um aplicativo.

[O armazenamento do Azure](#) é armazenamento em nuvem altamente escalonável do Azure. Os desenvolvedores podem aproveitar [armazenamento de filas](#) para enfileiramento de mensagens confiável, e [armazenamento de tabelas](#) é um repositório de chave-valor NoSQL projetado para rápido desenvolvimento usando conjuntos de dados grandes e semi-estruturados.

[Banco de dados SQL do Azure](#) fornece a funcionalidade de banco de dados relacional conhecida como um serviço usando o mecanismo do Microsoft SQL Server.

[O cosmos DB](#) serviço de banco de dados NoSQL multimodelo, distribuído globalmente. Várias APIs estão disponíveis, incluindo MongoDB, Cassandra e API do SQL (anteriormente chamado de DocumentDB).

## Identidade

[O Azure Active Directory](#) e [Azure Active Directory B2C](#) são ambos os serviços de identidade. O Azure Active Directory foi projetado para cenários empresariais e permite a colaboração do Azure AD B2B (business-to-business), enquanto o Azure Active Directory B2C é pretendidos cenários de negócios para o cliente, incluindo rede social entrar.

## Celular

[Os Hubs de notificação](#) é um mecanismo de notificação por push multiplataforma e dimensionável para enviar rapidamente milhões de mensagens para aplicativos em execução em vários tipos de dispositivos.

## Infraestrutura da Web

[O serviço de contêiner do Azure](#) gerencia seu ambiente Kubernetes hospedado, tornando rápido e fácil de implantar e gerenciar aplicativos em contêineres sem conhecimento de orquestração de contêiner.

[O Azure Search](#) é usado para criar uma solução de pesquisa empresarial sobre conteúdo privada e heterogênea.

[O Service Fabric](#) é uma plataforma de sistemas distribuídos que torna mais fácil empacotar, implantar e gerenciar escalonável e confiáveis microsserviços e contêineres.

# Hospedar o ASP.NET Core no Windows com o IIS

01/02/2019 • 50 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

[Instalar o pacote de hospedagem do .NET Core](#)

## Sistemas operacionais com suporte

Há suporte para os seguintes sistemas operacionais:

- Windows 7 ou posterior
- Windows Server 2008 R2 ou posterior

O [servidor HTTP.sys](#) (anteriormente chamado de WebListener) não funciona em uma configuração de proxy reverso com o IIS. Use o [servidor Kestrel](#).

Para obter mais informações sobre hospedagem no Azure, consulte [Implantar aplicativos ASP.NET Core no Serviço de Aplicativo do Azure](#).

## Plataformas com suporte

Aplicativos publicados para implantação de 32 bits (x86) e 64 bits (x64) têm suporte. Implemente um aplicativo de 32 bits, a menos que o aplicativo:

- Exija o maior espaço de endereço de memória virtual disponível para um aplicativo de 64 bits.
- Exija o maior tamanho de pilha do IIS.
- Tenha dependências nativas de 64 bits.

## Configuração do aplicativo

### Habilitar os componentes de IISIntegration

Um *Program.cs* típico chama [CreateDefaultBuilder](#) para começar a configurar um host:

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
    ...
```

Um *Program.cs* típico chama [CreateDefaultBuilder](#) para começar a configurar um host:

```
public static IWebHost BuildWebHost(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
    ...
```

### Modelo de hospedagem em processo

`CreateDefaultBuilder` chama o método `useIIS` para inicializar o [CoreCLR](#) e hospedar o aplicativo no processo de trabalho do IIS (`w3wp.exe` ou `iisexpress.exe`). Os testes de desempenho indicam que hospedar um aplicativo em processo do .NET Core oferece uma taxa de transferência de solicitação significativamente mais elevada em comparação a hospedar o aplicativo fora do processo e enviar por proxy as solicitações para o servidor [Kestrel](#).

Não há suporte para o modelo de hospedagem em processo para aplicativos ASP.NET Core direcionados ao .NET Framework.

## Modelo de hospedagem de fora do processo

Para a hospedagem fora do processo com o IIS, `CreateDefaultBuilder` configura o servidor **Kestrel** como servidor Web e habilita a integração do IIS ao configurar o caminho base e a porta para o [Módulo do ASP.NET Core](#).

O Módulo do ASP.NET Core gera uma porta dinâmica a ser atribuída ao processo de back-end.

`CreateDefaultBuilder` chama o método [UseIISIntegration](#). `UseIISIntegration` configura o Kestrel para escutar na porta dinâmica no endereço IP do localhost (`127.0.0.1`). Se a porta dinâmica for 1234, o Kestrel escutará em `127.0.0.1:1234`. Essa configuração substitui outras configurações de URL fornecidas por:

- `UseUrls`
- [API de escuta do Kestrel](#)
- [Configuração \(ou opção --urls de linha de comando\)](#)

As chamadas a `UseUrls` ou à API `Listen` do Kestrel não são necessárias ao usar o módulo. Se `UseUrls` ou `Listen` for chamado, o Kestrel escutará nas portas especificadas somente durante a execução do aplicativo sem o IIS.

Para obter mais informações sobre os modelos de hospedagem em processo e fora dele, veja [ASP.NET Core Module](#) (Módulo do ASP.NET Core) e [ASP.NET Core Module configuration reference](#) (Referência de configuração do módulo do ASP.NET Core).

`CreateDefaultBuilder` configura o servidor **Kestrel** como o servidor Web e habilita a integração do IIS, configurando o caminho base e a porta para o [Módulo do ASP.NET Core](#).

O Módulo do ASP.NET Core gera uma porta dinâmica a ser atribuída ao processo de back-end.

`CreateDefaultBuilder` chama o método [UseIISIntegration](#). `UseIISIntegration` configura o Kestrel para escutar na porta dinâmica no endereço IP do localhost (`127.0.0.1`). Se a porta dinâmica for 1234, o Kestrel escutará em `127.0.0.1:1234`. Essa configuração substitui outras configurações de URL fornecidas por:

- `UseUrls`
- [API de escuta do Kestrel](#)
- [Configuração \(ou opção --urls de linha de comando\)](#)

As chamadas a `UseUrls` ou à API `Listen` do Kestrel não são necessárias ao usar o módulo. Se `UseUrls` ou `Listen` for chamado, o Kestrel escutará na porta especificada somente durante a execução do aplicativo sem o IIS.

`CreateDefaultBuilder` configura o servidor **Kestrel** como o servidor Web e habilita a integração do IIS, configurando o caminho base e a porta para o [Módulo do ASP.NET Core](#).

O Módulo do ASP.NET Core gera uma porta dinâmica a ser atribuída ao processo de back-end.

`CreateDefaultBuilder` chama o método [UseIISIntegration](#). `UseIISIntegration` configura o Kestrel para escutar na porta dinâmica no endereço IP do localhost (`localhost`). Se a porta dinâmica for 1234, o Kestrel escutará em `localhost:1234`. Essa configuração substitui outras configurações de URL fornecidas por:

- `UseUrls`
- [API de escuta do Kestrel](#)
- [Configuração \(ou opção --urls de linha de comando\)](#)

As chamadas a `UseUrls` ou à API `Listen` do Kestrel não são necessárias ao usar o módulo. Se `UseUrls` ou `Listen` for chamado, o Kestrel escutará na porta especificada somente durante a execução do aplicativo sem o IIS.

Inclua uma dependência no pacote [Microsoft.AspNetCore.Server.IISIntegration](#) nas dependências do aplicativo. Use o middleware Integração do IIS adicionando o método de extensão [UseIISIntegration](#) ao [WebHostBuilder](#):

```
var host = new WebHostBuilder()
    .UseKestrel()
    .UseIISIntegration()
    ...
```

`UseKestrel` e `UseIISIntegration` são necessários. A chamada do código a `UseIISIntegration` não afeta a portabilidade do código. Se o aplicativo não é executado por trás do IIS (por exemplo, o aplicativo é executado diretamente em Kestrel), `UseIISIntegration` não opera.

O Módulo do ASP.NET Core gera uma porta dinâmica a ser atribuída ao processo de back-end.

`UseIISIntegration` configura o Kestrel para escutar na porta dinâmica no endereço IP do localhost (`localhost`). Se a porta dinâmica for 1234, o Kestrel escutará em `localhost:1234`. Essa configuração substitui outras configurações de URL fornecidas por:

- `UseUrls`
- [Configuração](#) (ou opção `--urls` de linha de comando)

Uma chamada a `UseUrls` não é necessária ao usar o módulo. Se `UseUrls` for chamado, o Kestrel escutará na porta especificada somente durante a execução do aplicativo sem o IIS.

Se `UseUrls` for chamado em um aplicativo do ASP.NET Core 1.0, chame-o **antes** de chamar `UseIISIntegration` para que a porta configurada pelo módulo não seja substituída. Essa ordem de chamada não é necessária com o ASP.NET Core 1.1, porque a configuração do módulo substitui `UseUrls`.

Para saber mais sobre hospedagem, confira [Host no ASP.NET Core](#).

## Opções do IIS

### Modelo de hospedagem em processo

Para configurar opções de IIS, inclua uma configuração de serviço para [IISServerOptions](#) em [ConfigureServices](#). O exemplo a seguir desabilita AutomaticAuthentication:

```
services.Configure<IISServerOptions>(options =>
{
    options.AutomaticAuthentication = false;
});
```

OPÇÃO	PADRÃO	CONFIGURAÇÃO
<code>AutomaticAuthentication</code>	<code>true</code>	Se <code>true</code> , o Servidor do IIS define o <code>HttpContext.User</code> autenticado pela <a href="#">Autenticação do Windows</a> . Se <code>false</code> , o servidor fornecerá apenas uma identidade para <code>HttpContext.User</code> e responderá a desafios quando explicitamente solicitado pelo <code>AuthenticationScheme</code> . A autenticação do Windows deve estar habilitada no IIS para que o <code>AutomaticAuthentication</code> funcione. Para obter mais informações, veja <a href="#">Autenticação do Windows</a> .

OPÇÃO	PADRÃO	CONFIGURAÇÃO
<code>AuthenticationDisplayName</code>	<code>null</code>	Configura o nome de exibição mostrado aos usuários em páginas de logon.

## Modelo de hospedagem de fora do processo

Para configurar opções de IIS, inclua uma configuração de serviço para [IISOptions](#) em [ConfigureServices](#). O exemplo a seguir impede que o aplicativo preencha `HttpContext.Connection.ClientCertificate`:

```
services.Configure<IISOptions>(options =>
{
    options.ForwardClientCertificate = false;
});
```

OPÇÃO	PADRÃO	CONFIGURAÇÃO
<code>AutomaticAuthentication</code>	<code>true</code>	Se <code>true</code> , o middleware de integração do IIS define o <code>HttpContext.User</code> autenticado pela <a href="#">Autenticação do Windows</a> . Se <code>false</code> , o middleware fornecerá apenas uma identidade para <code>HttpContext.User</code> e responderá a desafios quando explicitamente solicitado pelo <code>AuthenticationScheme</code> . A autenticação do Windows deve estar habilitada no IIS para que o <code>AutomaticAuthentication</code> funcione. Saiba mais no tópico <a href="#">Autenticação do Windows</a> .
<code>AuthenticationDisplayName</code>	<code>null</code>	Configura o nome de exibição mostrado aos usuários em páginas de logon.
<code>ForwardClientCertificate</code>	<code>true</code>	Se <code>true</code> e o cabeçalho da solicitação <code>MS-ASPNETCORE-CLIENTCERT</code> estiverem presentes, o <code>HttpContext.Connection.ClientCertificate</code> será populado.

## Servidor proxy e cenários de平衡ador de carga

O Middleware de integração do IIS, que configura Middleware de cabeçalhos encaminhados, e o módulo do ASP.NET Core são configurados para encaminhar o esquema (HTTP/HTTPS) e o endereço IP remoto de onde a solicitação foi originada. Configuração adicional pode ser necessária para aplicativos hospedados atrás de servidores proxy adicionais e平衡adores de carga. Para obter mais informações, veja [Configurar o ASP.NET Core para trabalhar com servidores proxy e平衡adores de carga](#).

## Arquivo web.config

O arquivo `web.config` configura o [Módulo do ASP.NET Core](#). Criando, transformar e publicar o arquivo `Web.config` é tratado por um destino do MSBuild (`_TransformWebConfig`) quando o projeto é publicado. Este destino está presente nos destinos do SDK da Web (`Microsoft.NET.Sdk.Web`). O SDK é definido na parte superior do arquivo de projeto:

```
<Project Sdk="Microsoft.NET.Sdk.Web">
```

Se um arquivo *web.config* não estiver presente no projeto, ele será criado com o *processPath* e os *argumentos* corretos para configurar o [Módulo do ASP.NET Core](#) e será transferido para o [resultado publicado](#).

Se um arquivo *web.config* estiver presente no projeto, ele será transformado com o *processPath* e os *argumentos* corretos para configurar o Módulo do ASP.NET Core e será movido para o resultado publicado. A transformação não altera as definições de configuração do IIS no arquivo.

O arquivo *web.config* pode fornecer configurações adicionais do IIS que controlam módulos ativos do IIS. Para saber mais sobre os módulos do IIS que podem processar solicitações com aplicativos do ASP.NET Core, veja o tópico [Módulos do IIS](#).

Para impedir que o SDK Web transforme o arquivo *web.config*, use a propriedade **<IsTransformWebConfigDisabled>** no arquivo do projeto:

```
<PropertyGroup>
  <IsTransformWebConfigDisabled>true</IsTransformWebConfigDisabled>
</PropertyGroup>
```

Ao impedir que o SDK Web transforme o arquivo, o *processPath* e os *argumentos* devem ser definidos manualmente pelo desenvolvedor. Para obter mais informações, consulte [Referência de configuração do módulo do ASP.NET Core](#).

### Local do arquivo *web.config*

Para configurar o [Módulo do ASP.NET Core](#) corretamente, o arquivo *web.config* deve estar presente no caminho raiz do conteúdo (geralmente, o aplicativo base do caminho) do aplicativo implantado. Esse é o mesmo local que o caminho físico do site fornecido ao IIS. O arquivo *web.config* é necessário na raiz do aplicativo para habilitar a publicação de vários aplicativos usando a Implantação da Web.

Existem arquivos confidenciais no caminho físico do aplicativo, como *<assembly>.runtimeconfig.json*, *<assembly>.xml* (comentários da Documentação XML) e *<assembly>.deps.json*. Quando o arquivo *web.config* estiver presente e o site for iniciado normalmente, o IIS não servirá esses arquivos confidenciais se eles forem solicitados. Se o arquivo *web.config* estiver ausente, nomeado incorretamente ou se não for possível configurar o site para inicialização normal, o IIS poderá servir arquivos confidenciais publicamente.

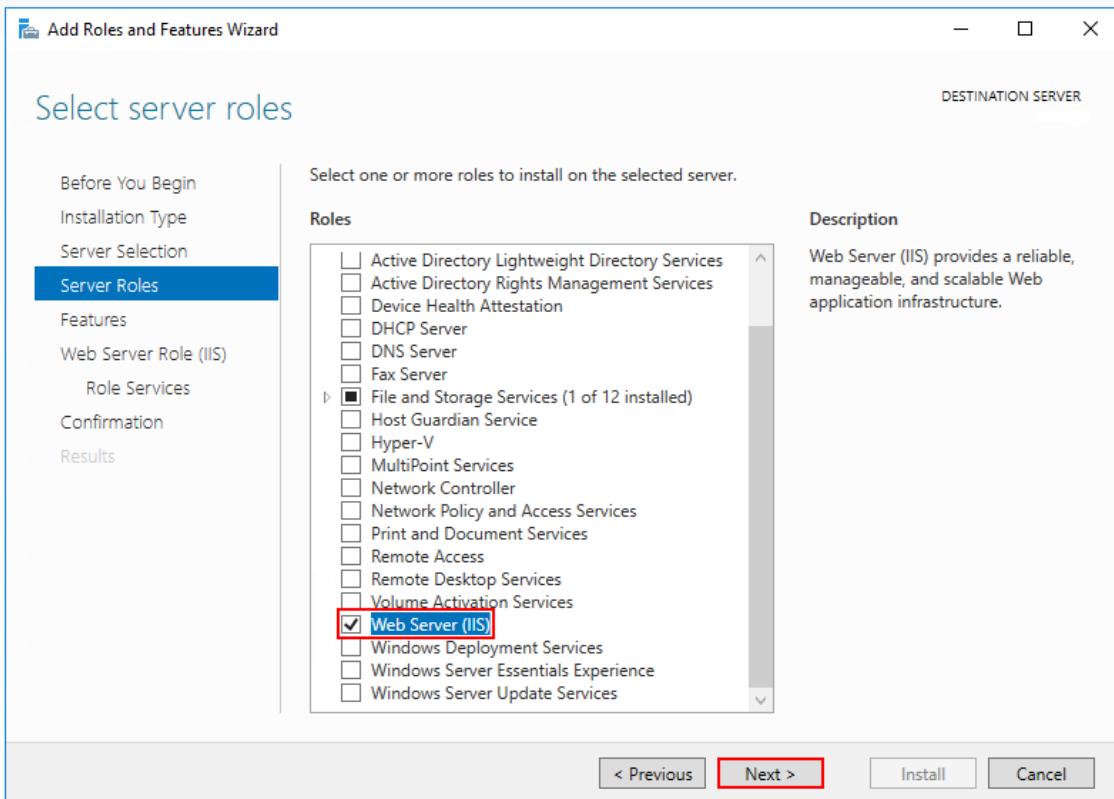
**O arquivo *web.config* deve estar presente na implantação em todos os momentos, nomeado corretamente e ser capaz de configurar o site para inicialização normal. Nunca remova o arquivo *web.config* de uma implantação de produção.**

## Configuração do IIS

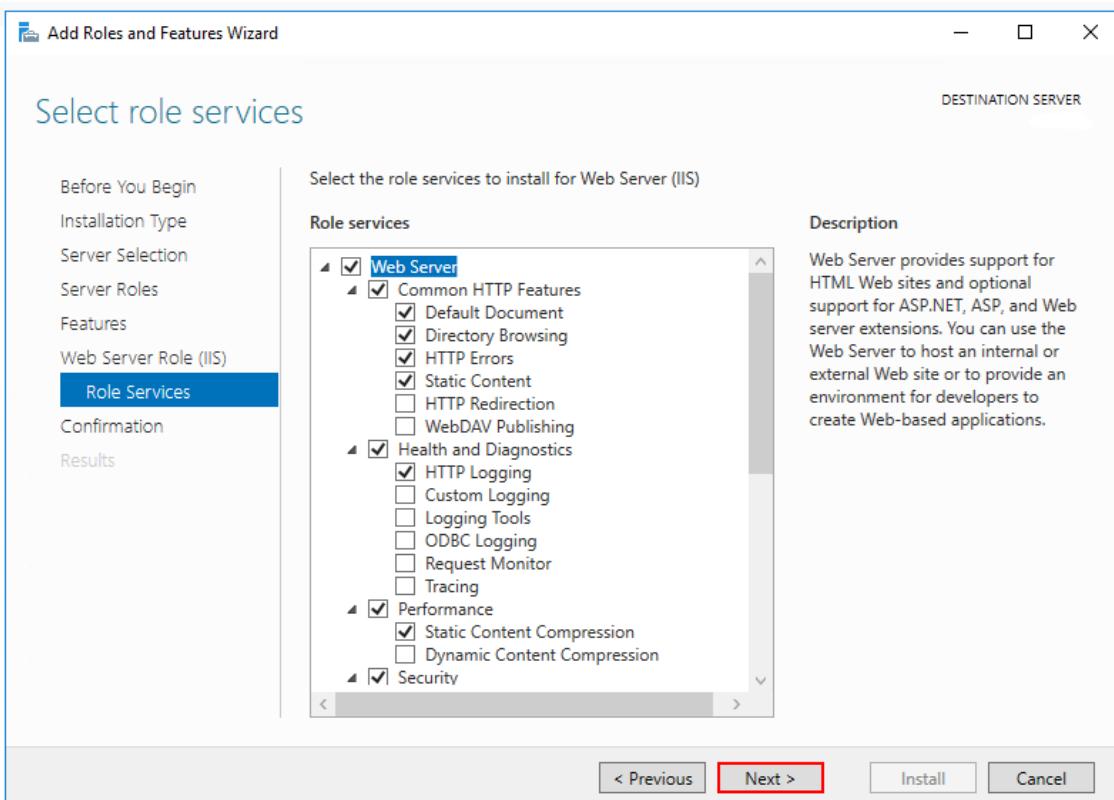
### Sistemas operacionais do Windows Server

Habilite a função **Servidor Web (IIS)** e estabeleça serviços de função.

1. Use o assistente **Adicionar Funções e Recursos** por meio do menu **Gerenciar** ou do link no **Gerenciador do Servidor**. Na etapa **Funções de Servidor**, marque a caixa de **Servidor Web (IIS)**.



2. Após a etapa **Recursos**, a etapa **Serviços de função** é carregada para o servidor Web (IIS). Selecione os serviços de função do IIS desejados ou aceite os serviços de função padrão fornecidos.



### Autenticação do Windows (opcional)

Para habilitar a Autenticação do Windows, expanda os nós a seguir: **Servidor Web > Segurança**.

Selecione o recurso **Autenticação do Windows**. Saiba mais em [Autenticação do Windows](#)

[`<windowsAuthentication>`](#) e [Configurar autenticação do Windows](#).

### WebSockets (opcional)

O WebSockets é compatível com o ASP.NET Core 1.1 ou posterior. Para habilitar WebSockets, expanda os nós a seguir: **Servidor Web > Desenvolvimento de Aplicativo**. Selecione o recurso **Protocolo**

**WebSocket.** Para obter mais informações, consulte [WebSockets](#).

3. Continue para a etapa **Confirmação** para instalar os serviços e a função de servidor Web. Um comando server/IIS restart não será necessário após a instalação da função **Servidor Web (IIS)**.

## Sistemas operacionais Windows de área de trabalho

Habilite o **Console de Gerenciamento do IIS** e os **Serviços na World Wide Web**.

1. Navegue para **Painel de Controle > Programas > Programas e Recursos > Ativar ou desativar recursos do Windows** (lado esquerdo da tela).
2. Abra o nó **Serviços de Informações da Internet**. Abra o nó **Ferramentas de Gerenciamento da Web**.
3. Marque a caixa de **Console de Gerenciamento do IIS**.
4. Marque a caixa de **Serviços na World Wide Web**.
5. Aceite os recursos padrão dos **Serviços na World Wide Web** ou personalize os recursos do IIS.

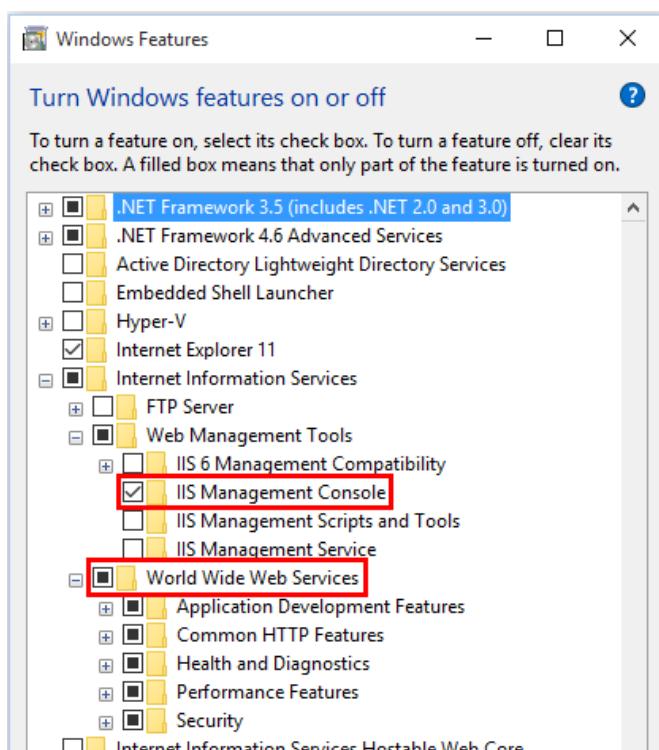
### Autenticação do Windows (opcional)

Para habilitar a Autenticação do Windows, expanda os nós a seguir: **Serviços na World Wide Web > Segurança**. Selecione o recurso **Autenticação do Windows**. Saiba mais em [Autenticação do Windows <windowsAuthentication>](#) e [Configurar autenticação do Windows](#).

### WebSockets (opcional)

O WebSockets é compatível com o ASP.NET Core 1.1 ou posterior. Para habilitar WebSockets, expanda os nós a seguir: **Serviços na World Wide Web > Recursos de Desenvolvimento de Aplicativos**. Selecione o recurso **Protocolo WebSocket**. Para obter mais informações, consulte [WebSockets](#).

6. Se a instalação do IIS exigir uma reinicialização, reinicie o sistema.



## Instalar o pacote de hospedagem do .NET Core

Instale o *pacote de hospedagem do .NET Core* no sistema de hospedagem. O pacote instala o Tempo de Execução .NET Core, a Biblioteca do .NET Core e o [Módulo do ASP.NET Core](#). O módulo permite que aplicativos do ASP.NET Core sejam executados por trás do IIS. Se o sistema não tiver uma conexão com a

Internet, obtenha e instale os [Pacotes redistribuíveis do Microsoft Visual C++ 2015](#) antes de instalar o pacote de hospedagem do .NET Core.

#### IMPORTANT

Se o pacote de hospedagem for instalado antes do IIS, a instalação do pacote deverá ser reparada. Execute o instalador do pacote de hospedagem novamente depois de instalar o IIS.

### Download direto (versão atual)

Baixe o instalador usando o seguinte link:

[Instalador de pacote de hospedagem do .NET Core atual \(download direto\)](#)

### Versões anteriores do instalador

Para obter uma versão anterior do instalador:

1. Navegue até os [arquivos de downloads do .NET](#).
2. Em **.NET Core**, selecione a versão do .NET Core.
3. Na coluna **Executar aplicativos – Tempo de execução**, localize a linha da versão de tempo de execução do .NET Core desejada.
4. Baixe o instalador usando o link **Pacote de hospedagem e de tempo de execução**.

#### WARNING

Alguns instaladores contêm versões de lançamento que atingiram o EOL (fim da vida útil) e não têm mais suporte da Microsoft. Para saber mais, confira a [política de suporte](#).

### Instalar o pacote de hospedagem

1. Execute o instalador no servidor. As seguintes opções estão disponíveis ao executar o instalador em um prompt de comando do administrador:
  - `OPT_NO_ANCM=1` – Ignorar a instalação do Módulo do ASP.NET Core.
  - `OPT_NO_RUNTIME=1` – Ignorar a instalação do tempo de execução do .NET Core.
  - `OPT_NO_SHAREDFX=1` – Ignorar a instalação da Estrutura Compartilhada do ASP.NET (tempo de execução do ASP.NET).
  - `OPT_NO_X86=1` – Ignorar a instalação dos tempos de execução x86. Use essa opção quando você sabe que não hospedará aplicativos de 32 bits. Se houver uma possibilidade de hospedar aplicativos de 32 bits e 64 bits no futuro, não use essa opção e instale ambos os tempos de execução.
2. Reinicie o sistema ou execute **net stop was /y** seguido por **net start w3svc** em um prompt de comando. A reinicialização do IIS identifica uma alteração no CAMINHO do sistema, que é uma variável de ambiente, realizada pelo instalador.

Se o instalador do Pacote de Hospedagem do Windows detectar que o IIS requer uma reinicialização para concluir a instalação, o instalador reiniciará o IIS. Se o instalador disparar uma reinicialização do IIS, todos os pools de aplicativos do IIS e sites serão reiniciados.

#### NOTE

Para obter informações sobre a Configuração Compartilhada do IIS, consulte [Módulo do ASP.NET Core com a Configuração Compartilhada do IIS](#).

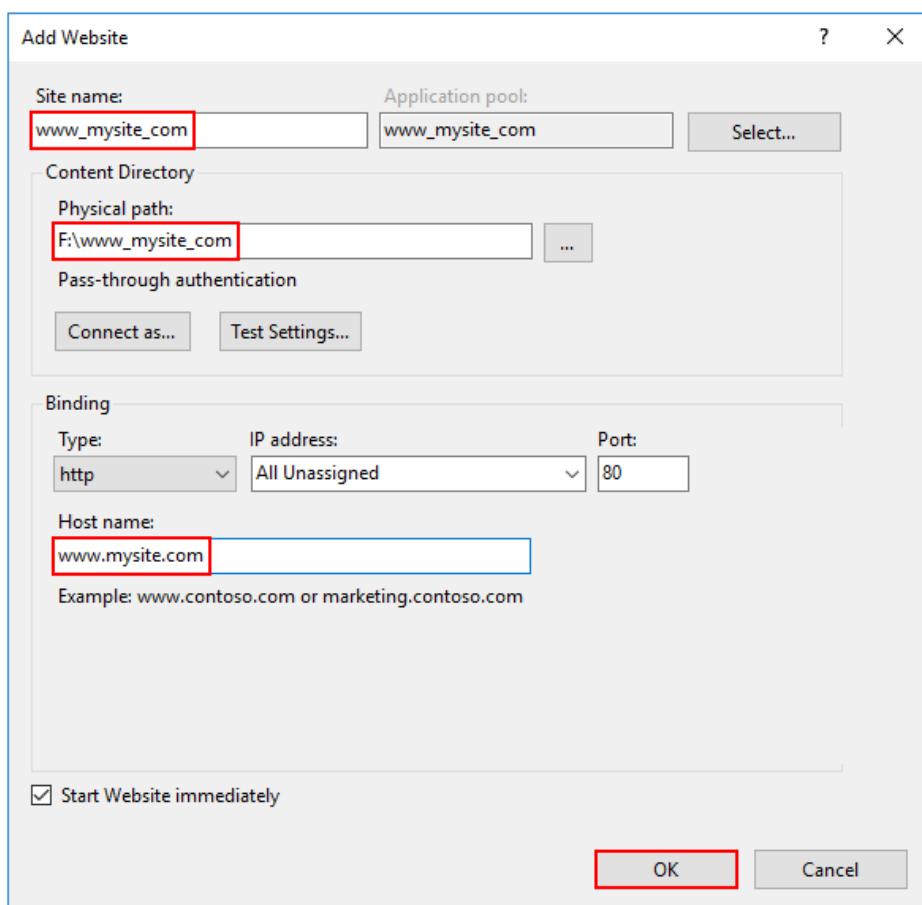
## Instalar a Implantação da Web durante a publicação com o Visual

# Studio

Ao implantar aplicativos para servidores com [Implantação da Web](#), instale a versão mais recente da Implantação da Web no servidor. Para instalar a Implantação da Web, use o [WebPI \(Web Platform Installer\)](#) ou obtenha um instalador diretamente no [Centro de Download da Microsoft](#). O método preferencial é usar o WebPI. O WebPI oferece uma instalação autônoma e uma configuração para provedores de hospedagem.

## Criar o site do IIS

1. No sistema de hospedagem, crie uma pasta para conter arquivos e pastas publicados do aplicativo. O layout de implantação do aplicativo é descrito no tópico [Estrutura de Diretórios](#).
2. No **Gerenciador do IIS**, abra o nó do servidor no painel **Conexões**. Clique com botão direito do mouse na pasta **Sites**. Selecione **Adicionar Site** no menu contextual.
3. Forneça um **Nome do site** e defina o **Caminho físico** como a pasta de implantação do aplicativo. Forneça a configuração **Associação** e crie o site ao selecionar **OK**:



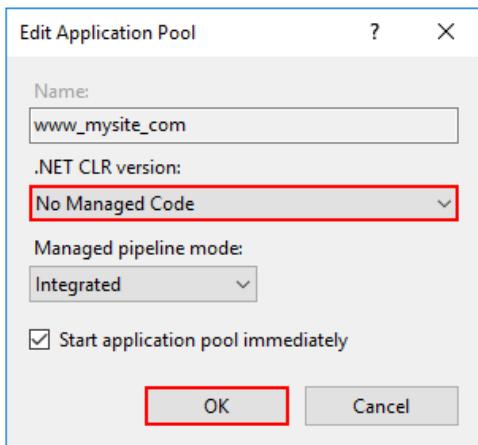
### WARNING

Associações de curinga de nível superior ( `http://*:80/` e `http://+:80` ) **não** devem ser usadas. Associações de curinga de nível superior podem abrir o aplicativo para vulnerabilidades de segurança. Isso se aplica a curingas fortes e fracos. Use nomes de host explícitos em vez de curingas. Associações de curinga de subdomínio (por exemplo, `*.mysub.com` ) não têm esse risco de segurança se você controlar o domínio pai completo (em vez de `*.com` , o qual é vulnerável). Veja [rfc7230 section-5.4](#) para obter mais informações.

4. No nó do servidor, selecione **Pools de Aplicativos**.
5. Clique com o botão direito do mouse no pool de aplicativos do site e selecione **Configurações Básicas** no menu contextual.

6. Na janela **Editar Pool de Aplicativos**, defina a **versão do CLR do .NET** como **Sem Código Gerenciado**:

**Gerenciado:**



O ASP.NET Core é executado em um processo separado e gerencia o tempo de execução. O ASP.NET Core não depende do carregamento do CLR de área de trabalho. Definir a **versão do CLR do .NET** como **Sem Código Gerenciado** é opcional.

7. *ASP.NET Core 2.2 ou posterior:* para uma [implantação autocontida](#) de 64 bits (x64) que usa o [modelo de hospedagem em processo](#), desabilite o pool de aplicativos para processos de 32 bits (x86).

Na barra lateral **Ações** dos **Pools de Aplicativos** do Gerenciador do IIS, selecione **Definir Padrões do Pool de Aplicativos** ou **Configurações Avançadas**. Localize **Habilitar Aplicativos de 32 bits** e defina o valor como `False`. Essa configuração não afeta os aplicativos implantados para a [hospedagem fora do processo](#).

8. Confirme se a identidade do modelo de processo tem as permissões apropriadas.

Se você alterar a identidade padrão do pool de aplicativos (**Modelo de Processo > Identidade**) em **ApplicationPoolIdentity** para outra, verifique se a nova identidade tem as permissões necessárias para acessar a pasta do aplicativo, o banco de dados e outros recursos necessários. Por exemplo, o pool de aplicativos requer acesso de leitura e gravação às pastas nas quais o aplicativo lê e grava os arquivos.

#### **Configuração de Autenticação do Windows (opcional)**

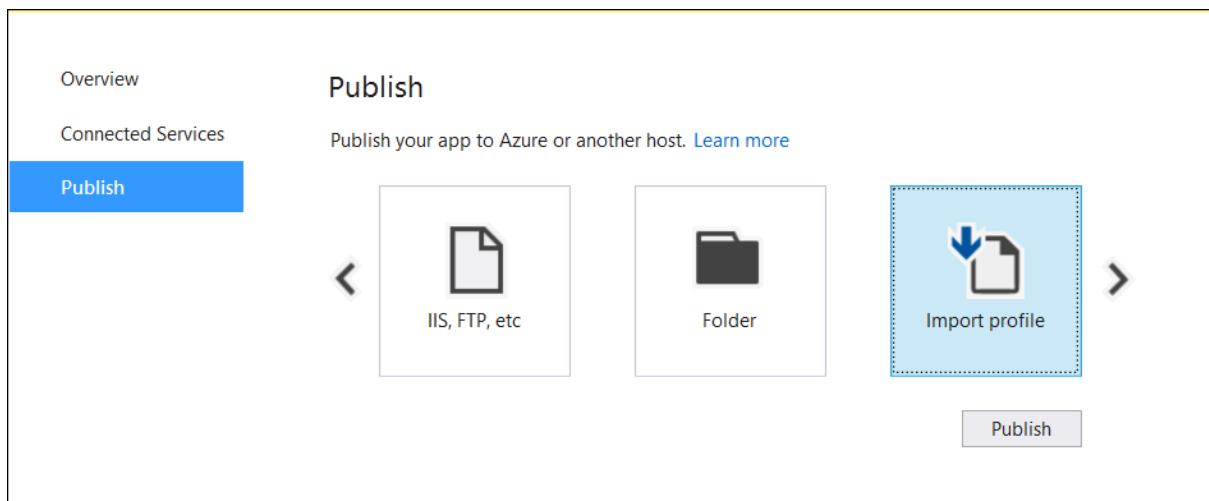
Para saber mais, veja [Configurar a Autenticação do Windows](#).

## Implantar o aplicativo

Implante o aplicativo na pasta criada no sistema de hospedagem. A [Implantação da Web](#) é o mecanismo recomendado para implantação.

#### **Implantação da Web com o Visual Studio**

Consulte o tópico [Perfis de publicação do Visual Studio para implantação de aplicativos ASP.NET Core](#) para saber como criar um perfil de publicação para uso com a Implantação da Web. Se o provedor de hospedagem fornecer um Perfil de Publicação ou o suporte para a criação de um, baixe o perfil e importe-o usando a caixa de diálogo **Publicar** do Visual Studio.



## Implantação da Web fora do Visual Studio

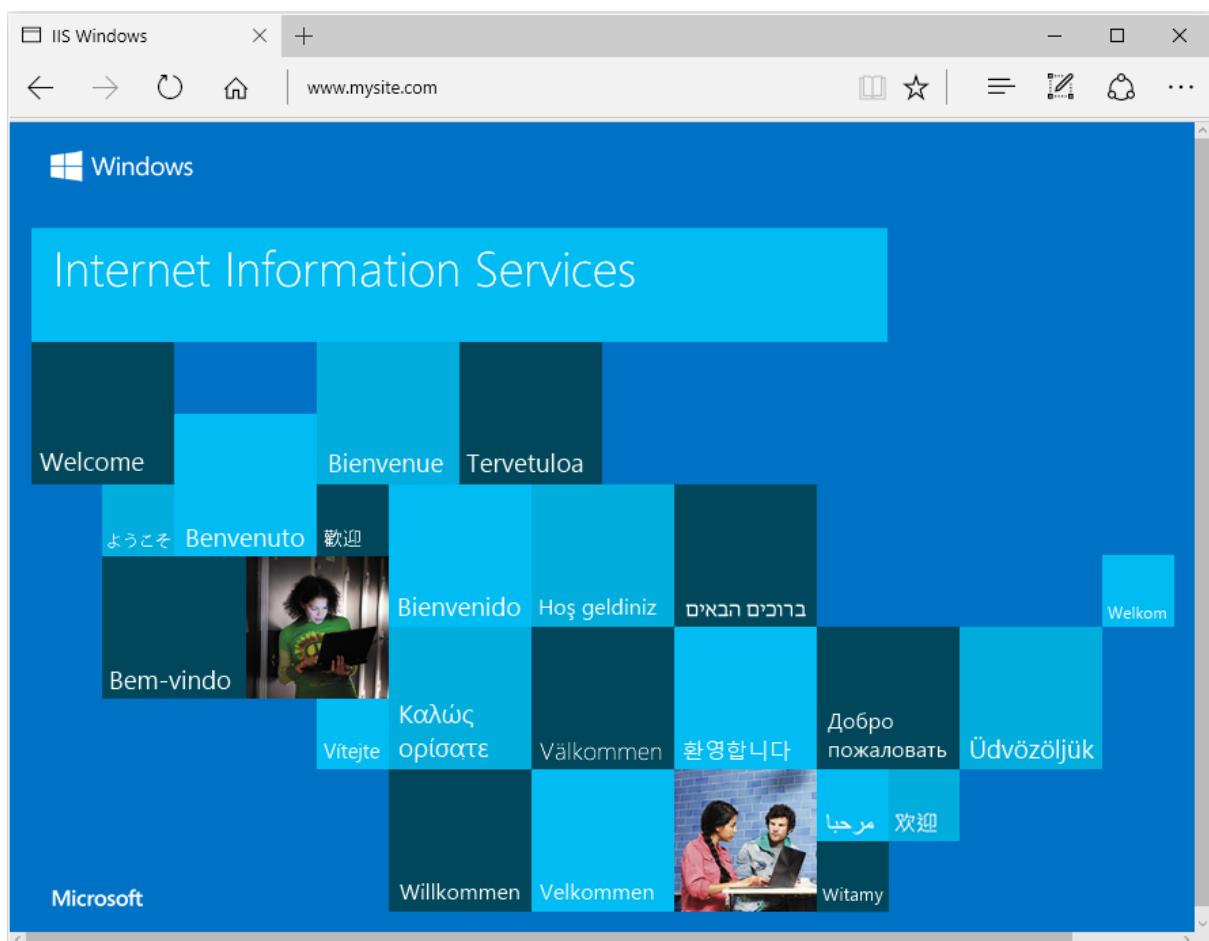
Também é possível usar a [Implantação da Web](#) fora do Visual Studio na linha de comando. Para obter mais informações, consulte [Ferramenta de Implantação da Web](#).

## Alternativas à Implantação da Web

Use qualquer um dos vários métodos para mover o aplicativo para o sistema host, como cópia manual, Xcopy, Robocopy ou PowerShell.

Para obter mais informações sobre a implantação do ASP.NET Core no IIS, consulte a seção [Recursos de implantação para administradores do IIS](#).

## Navegar no site



## Arquivos de implantação bloqueados

Os arquivos na pasta de implantação são bloqueados quando o aplicativo está em execução. Os arquivos bloqueados não podem ser substituídos durante a implantação. Para liberar os arquivos bloqueados em uma implantação, interrompa o pool de aplicativos usando **uma** das abordagens a seguir:

- Use a Implantação da Web e refcrcie `Microsoft.NET.Sdk.Web` no arquivo do projeto. Um arquivo `app_offline.htm` é colocado na raiz do diretório de aplicativo da Web. Quando o arquivo estiver presente, o módulo do ASP.NET Core apenas desligará o aplicativo e servirá o arquivo `app_offline.htm` durante a implantação. Para obter mais informações, consulte [Referência de configuração do módulo do ASP.NET Core](#).
- Manualmente interrompa o pool de aplicativos no Gerenciador do IIS no servidor.
- Use o PowerShell para soltar `app_offline.html` (requer o PowerShell 5 ou posterior):

```
$pathToApp = 'PATH_TO_APP'

# Stop the AppPool
New-Item -Path $pathToApp app_offline.htm

# Provide script commands here to deploy the app

# Restart the AppPool
Remove-Item -Path $pathToApp app_offline.htm
```

## Proteção de dados

A pilha [Proteção de Dados do ASP.NET Core](#) é usada por vários [middlewares](#) ASP.NET Core, incluindo aqueles usados na autenticação. Mesmo se as APIs de proteção de dados não forem chamadas pelo código do usuário, a proteção de dados deverá ser configurada com um script de implantação ou no código do usuário para criar um [repositório de chaves](#) criptográfico persistente. Se a proteção de dados não estiver configurada, as chaves serão mantidas na memória e descartadas quando o aplicativo for reiniciado.

Se o token de autenticação for armazenado na memória quando o aplicativo for reiniciado:

- Todos os tokens de autenticação baseados em cookies serão invalidados.
- Os usuários precisam entrar novamente na próxima solicitação deles.
- Todos os dados protegidos com o token de autenticação não poderão mais ser descriptografados. Isso pode incluir os [tokens CSRF](#) e [cookies TempData do MVC do ASP.NET Core](#).

Para configurar a proteção de dados no IIS para persistir o token de autenticação, use **uma** das seguintes abordagens:

- **Criar chaves de registro de proteção de dados**

As chaves de proteção de dados usadas pelos aplicativos ASP.NET Core são armazenadas no registro externo aos aplicativos. Para persistir as chaves de determinado aplicativo, crie uma chave de registro para o pool de aplicativos.

Para instalações autônomas do IIS não Web Farm, você pode usar o [script Provision-AutoGenKeys.ps1 de Proteção de Dados do PowerShell](#) para cada pool de aplicativos usado com um aplicativo ASP.NET Core. Esse script cria uma chave de registro no registro HKLM que é acessível apenas para a conta do processo de trabalho do pool de aplicativos do aplicativo. As chaves são criptografadas em repouso usando a DPAPI com uma chave para todo o computador.

Em cenários de web farm, um aplicativo pode ser configurado para usar um caminho UNC para armazenar seu token de autenticação de proteção de dados. Por padrão, as chaves de proteção de dados não são criptografadas. Garanta que as permissões de arquivo de o compartilhamento de rede

sejam limitadas à conta do Windows na qual o aplicativo é executado. Um certificado X509 pode ser usado para proteger chaves em repouso. Considere um mecanismo para permitir aos usuários carregar certificados: coloque os certificados no repositório de certificados confiáveis do usuário e certifique-se de que eles estejam disponíveis em todos os computadores nos quais o aplicativo do usuário é executado. Veja [Configurar a proteção de dados do ASP.NET Core](#) para obter detalhes.

- **Configurar o pool de aplicativos do IIS para carregar o perfil do usuário**

Essa configuração está na seção **Modelo de processo** nas **Configurações avançadas** do pool de aplicativos. Defina Carregar perfil do usuário como `True`. Isso armazena as chaves no diretório do perfil do usuário e as protege usando a DPAPI com uma chave específica à conta de usuário usada pelo pool de aplicativos.

- **Use o sistema de arquivos como um repositório de tokens de autenticação**

Ajuste o código do aplicativo para [usar o sistema de arquivos como um repositório de tokens de autenticação](#). Use um certificado X509 para proteger o token de autenticação e verifique se ele é um certificado confiável. Se o certificado for autoassinado, você deverá colocá-lo no repositório Raiz confiável.

Ao usar o IIS em uma web farm:

- Use um compartilhamento de arquivos que pode ser acessado por todos os computadores.
- Implante um certificado X509 em cada computador. Configure a [proteção de dados no código](#).

- **Definir uma política para todo o computador para proteção de dados**

O sistema de proteção de dados tem suporte limitado para a configuração da [política de todo o computador](#) padrão para todos os aplicativos que consomem as APIs de proteção de dados. Para obter mais informações, consulte [Proteção de dados do ASP.NET Core](#).

## Diretórios virtuais

[Diretórios virtuais IIS](#) não são compatíveis com aplicativos ASP.NET Core. Um aplicativo pode ser hospedado como um [subaplicativo](#).

## Subaplicativos

Um aplicativo ASP.NET Core pode ser hospedado como um [subaplicativo IIS \(sub-app\)](#). A parte do caminho do subaplicativo se torna parte da URL raiz do aplicativo.

Um subaplicativo não deve incluir o módulo do ASP.NET Core como um manipulador. Se o módulo for adicionado como um manipulador em um arquivo `web.config` de um subaplicativo, quando tentar procurar o subaplicativo, você receberá um `500.19 Erro Interno do Servidor` que referenciará o arquivo de configuração com falha.

O seguinte exemplo mostra um arquivo `web.config` publicado de um subaplicativo ASP.NET Core:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.webServer>
    <aspNetCore processPath="dotnet"
      arguments=".\\MyApp.dll"
      stdoutLogEnabled="false"
      stdoutLogFile=".\\logs\\stdout" />
  </system.webServer>
</configuration>
```

Ao hospedar um subaplicativo não ASP.NET Core abaixo de um aplicativo ASP.NET Core, remova

explicitamente o manipulador herdado no arquivo `web.config` do subaplicativo:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.webServer>
    <handlers>
      <remove name="aspNetCore" />
    </handlers>
    <aspNetCore processPath="dotnet"
      arguments=".\\MyApp.dll"
      stdoutLogEnabled="false"
      stdoutLogFile=".\\logs\\stdout" />
  </system.webServer>
</configuration>
```

Links de ativos estáticos dentro do subaplicativo devem usar a notação de sinal de til e barra (`~/`). A notação de sinal de til e barra aciona um [Auxiliar de Marca](#) para preceder a base de caminho do subaplicativo ao link relativo renderizado. Para um subaplicativo no `/subapp_path`, uma imagem vinculada com `src="~/image.png"` é renderizada como `src="/subapp_path/image.png"`. O Middleware de Arquivo Estático do aplicativo raiz não processa a solicitação de arquivo estático. A solicitação é processada pelo Middleware de Arquivo Estático do subaplicativo.

Se um atributo de ativo estático `src` for definido como um caminho absoluto (por exemplo, `src="/image.png"`), o link será renderizado sem a base de caminho do subaplicativo. O Middleware de Arquivos Estáticos do aplicativo raiz tenta fornecer o ativo do `webroot` da raiz do aplicativo, que resulta em uma resposta `404 – Não encontrado`, a menos que o ativo estático esteja disponível no aplicativo raiz.

Para hospedar um aplicativo ASP.NET Core como um subaplicativo em outro aplicativo do ASP.NET Core:

1. Estabeleça um pool de aplicativos para o subaplicativo. Defina a [versão do CLR do .NET](#) como **Sem Código Gerenciado**.
2. Adicione o site raiz no Gerenciador do IIS com o subaplicativo em uma pasta no site raiz.
3. Clique com o botão direito do mouse na pasta do subaplicativo no Gerenciador do IIS e selecione **Converter em aplicativo**.
4. Na caixa de diálogo **Adicionar Aplicativo**, use o botão **Selecionar** no **Pool de Aplicativos** para atribuir o pool de aplicativos que você criou ao subaplicativo. Selecione **OK**.

A atribuição de um pool de aplicativos separado para o subaplicativo é um requisito ao usar o modelo de hospedagem em processo.

Para obter mais informações sobre o modelo de hospedagem em processo e como configurar o módulo do ASP.NET Core, confira [Módulo do ASP.NET Core](#) e [Módulo do ASP.NET Core](#).

## Configuração do IIS com `web.config`

A configuração do IIS é influenciada pela seção `<system.webServer>` do `web.config` para cenários do IIS que são funcionais para aplicativos ASP.NET Core com o Módulo do ASP.NET Core. Por exemplo, a configuração do IIS é funcional para a compactação dinâmica. Se o IIS for configurado no nível do servidor para usar a compactação dinâmica, o elemento `<urlCompression>` no arquivo `web.config` do aplicativo pode desabilitá-la para um aplicativo do ASP.NET Core.

Para saber mais, veja a [referência de configuração do `<system.webServer>`](#), a [referência de configuração do Módulo do ASP.NET Core](#) e [Módulos do IIS com o ASP.NET Core](#). Para definir variáveis de ambiente para aplicativos individuais executados em pools de aplicativos isolados (compatível com o IIS 10.0+ ou posterior), veja a seção `comando AppCmd.exe` do tópico [Variáveis de ambiente <environmentVariables>](#) na documentação de referência do IIS.

## Seções de configuração de web.config

As seções de configuração de aplicativos ASP.NET 4.x em *web.config* não são usadas por aplicativos ASP.NET Core para configuração:

- `<system.web>`
- `<appSettings>`
- `<connectionStrings>`
- `<location>`

Aplicativos ASP.NET Core são configurados para usar outros provedores de configuração. Para obter mais informações, consulte [Configuração](#).

## Pools de aplicativos

O isolamento do pool de aplicativos é determinado pelo modelo de hospedagem:

- Hospedagem em processo – Aplicativos são necessários para execução em pools de aplicativos separados.
- Hospedagem fora do processo – É recomendável isolar os aplicativos uns dos outros, executando cada aplicativo em seu próprio pool de aplicativos.

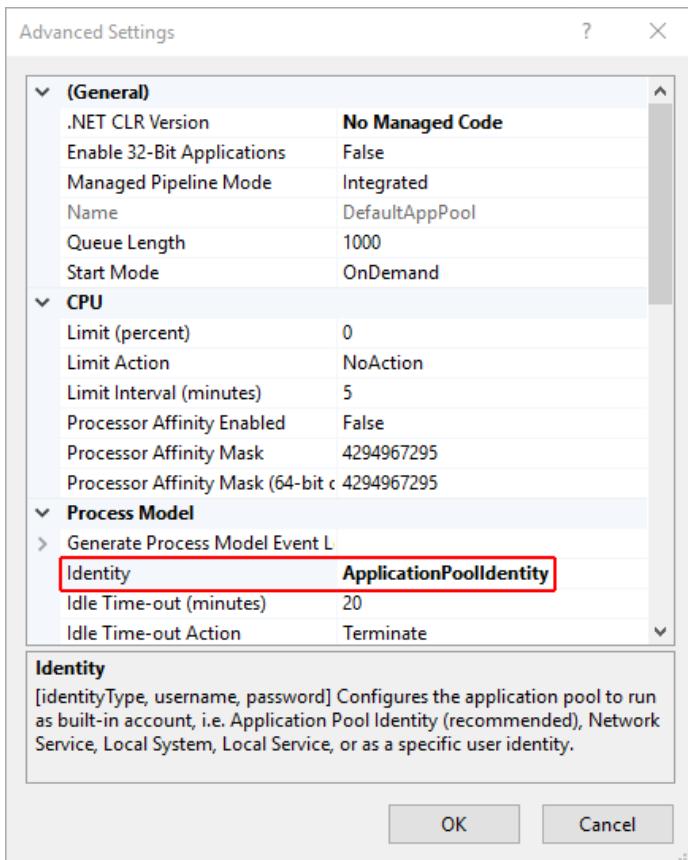
A caixa de diálogo **Adicionar Site** do IIS usa como padrão um único pool de aplicativos por aplicativo.

Quando um **Nome de site** é fornecido, o texto é transferido automaticamente para a caixa de texto **Pool de aplicativos**. Um novo pool de aplicativos é criado usando o nome do site quando você adicionar o site.

Ao hospedar vários sites em um servidor, é recomendável isolar os aplicativos uns dos outros, executando cada aplicativo em seu próprio pool de aplicativo. A caixa de diálogo **Adicionar Site** do IIS usa como padrão essa configuração. Quando um **Nome de site** é fornecido, o texto é transferido automaticamente para a caixa de texto **Pool de aplicativos**. Um novo pool de aplicativos é criado usando o nome do site quando você adicionar o site.

## Identidade do pool de aplicativos

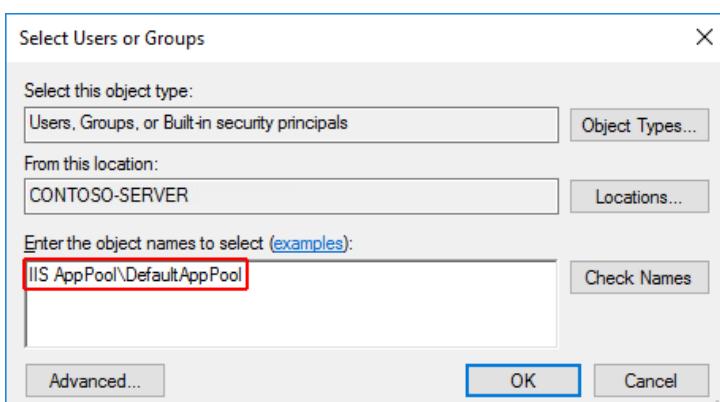
Uma conta de identidade do pool de aplicativos permite executar um aplicativo em uma conta exclusiva sem a necessidade de criar e gerenciar domínios ou contas locais. No IIS 8.0 ou posterior, o WAS (Processo de trabalho do administrador) do IIS cria uma conta virtual com o nome do novo pool de aplicativos e executa os processos de trabalho do pool de aplicativos nesta conta por padrão. No Console de Gerenciamento do IIS, em **Configurações avançadas** do pool de aplicativos, verifique se a **Identidade** é definida para usar **ApplicationPoolIdentity**:



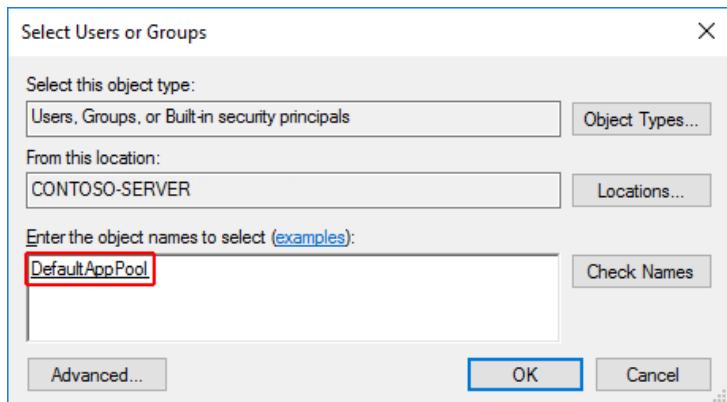
O processo de gerenciamento do IIS cria um identificador seguro com o nome do pool de aplicativos no Sistema de segurança do Windows. Os recursos podem ser protegidos usando essa identidade. No entanto, essa identidade não é uma conta de usuário real e não será mostrada no Console de Gerenciamento de Usuários do Windows.

Se o processo de trabalho do IIS requerer acesso elevado ao aplicativo, modifique a ACL (lista de controle de acesso) do diretório que contém o aplicativo:

1. Abra o Windows Explorer e navegue para o diretório.
2. Clique com o botão direito do mouse no diretório e selecione **Propriedades**.
3. Na guia **Segurança**, selecione o botão **Editar** e, em seguida, no botão **Adicionar**.
4. Clique no botão **Locais** e verifique se o sistema está selecionado.
5. Insira **IIS AppPool\<nome\_pool\_aplicativos>** na área **Inserir os nomes de objeto a serem selecionados**. Selecione o botão **Verificar Nomes**. Para o **DefaultAppPool**, verifique os nomes usando **IIS AppPool\DefaultAppPool**. Quando o botão **Verificar Nomes** é selecionado, um valor de **DefaultAppPool** é indicado na área de nomes de objeto. Não é possível inserir o nome do pool de aplicativos diretamente na área de nomes de objeto. Use o formato **IIS AppPool\<nome\_pool\_aplicativos>** ao verificar o nome do objeto.



6. Selecione **OK**.



7. As permissões de leitura & execução devem ser concedidas por padrão. Forneça permissões adicionais conforme necessário.

O acesso também pode ser concedido por meio de um prompt de comando usando a ferramenta **ICACLS**. Usando o *DefaultAppPool* como exemplo, o comando a seguir é usado:

```
ICACLS C:\sites\MyWebApp /grant "IIS AppPool\DefaultAppPool":F
```

Para saber mais, veja o tópico [icacls](#).

## Compatibilidade com HTTP/2

O [HTTP/2](#) é compatível com ASP.NET Core nos seguintes cenários de implantação de IIS:

- Em processo
  - Windows Server 2016/Windows 10 ou posterior; IIS 10 ou posterior
  - Conexão TLS 1.2 ou posterior
- Fora do processo
  - Windows Server 2016/Windows 10 ou posterior; IIS 10 ou posterior
  - Conexões de servidor de borda voltadas para o público usam HTTP/2, mas a conexão de proxy reverso para o [servidor Kestrel](#) usa HTTP/1.1.
  - Conexão TLS 1.2 ou posterior

Para uma implantação em processo quando uma conexão HTTP/2 for estabelecida, o [HttpRequest.Protocol](#) relatará `HTTP/2`. Para uma implantação fora de processo quando uma conexão HTTP/2 for estabelecida, o [HttpRequest.Protocol](#) relatará `HTTP/1.1`.

Para saber mais sobre os modelos de hospedagem em processo e fora de processo, confira o tópico [Módulo do ASP.NET Core](#) e o [Módulo do ASP.NET Core](#).

O [HTTP/2](#) é compatível com implantações fora de processo que cumprem os seguintes requisitos básicos:

- Windows Server 2016/Windows 10 ou posterior; IIS 10 ou posterior
- Conexões de servidor de borda voltadas para o público usam HTTP/2, mas a conexão de proxy reverso para o [servidor Kestrel](#) usa HTTP/1.1.
- Estrutura de destino: não se aplica a implantações fora de processo, visto que a conexão HTTP/2 é manipulada inteiramente pelo IIS.
- Conexão TLS 1.2 ou posterior

Se uma conexão HTTP/2 for estabelecida, [HttpRequest.Protocol](#) relatará `HTTP/1.1`.

O HTTP/2 está habilitado por padrão. As conexões retornarão para HTTP/1.1 se uma conexão HTTP/2 não for

estabelecida. Para obter mais informações sobre a configuração de HTTP/2 com implantações do IIS, consulte [HTTP/2 no IIS](#).

## Solicitações de simulação do CORS

*Esta seção só se aplica a aplicativos ASP.NET Core com o .NET Framework como destino.*

Para um aplicativo ASP.NET Core com o .NET Framework como destino, as solicitações OPTIONS não são passadas para o aplicativo por padrão no IIS. Confira como configurar os manipuladores IIS do aplicativo no *Web.config* para passar as solicitações OPTIONS em [Habilitar solicitações entre origens na ASP.NET Web API 2: Como funciona o CORS](#).

## Recursos de implantação para administradores do IIS

Conheça o ISS detalhadamente na documentação do IIS.

[Documentação do ISS](#)

Saiba mais sobre os modelos de implantação de aplicativos do .NET Core.

[Implantação de aplicativos do .NET Core](#)

Saiba como o módulo do ASP.NET Core permite que o servidor Web Kestrel use o IIS ou o IIS Express como um servidor proxy reverso.

[Módulo do ASP.NET Core](#)

Saiba como configurar o módulo do ASP.NET Core para hospedar aplicativos do ASP.NET Core.

[Referência de configuração do Módulo do ASP.NET Core](#)

Saiba mais sobre a estrutura do diretório de aplicativos publicados do ASP.NET Core.

[Estrutura de diretórios](#)

Descubra módulos ativos e inativos do IIS para aplicativos do ASP.NET Core e como gerenciar os módulos do IIS.

[Módulos do IIS](#)

Saiba como diagnosticar problemas com as implantações do ISS dos aplicativos do ASP.NET Core.

[Solução de problemas](#)

Distinga erros comuns ao hospedar aplicativos do ASP.NET Core no IIS.

[Referência de erros comuns para o Serviço de Aplicativo do Azure e o IIS](#)

## Recursos adicionais

- [Solucionar problemas de projetos do ASP.NET Core](#)
- [Introdução ao ASP.NET Core](#)
- [O site oficial da IIS da Microsoft](#)
- [Biblioteca de conteúdo técnico do Windows Server](#)
- [HTTP/2 no IIS](#)

# Módulo do ASP.NET Core

30/01/2019 • 43 minutes to read • [Edit Online](#)

Por [Tom Dykstra](#), [Rick Strahl](#), [Chris Ross](#), [Rick Anderson](#), [Sourabh Shirhatti](#), [Justin Kotalik](#) e [Luke Latham](#)

O módulo do ASP.NET Core é um módulo nativo do IIS que se conecta ao pipeline do IIS para:

- Hoster um aplicativo ASP.NET Core dentro do processo de trabalho do IIS (`w3wp.exe`), chamado o [modelo de hospedagem no processo](#).
- Encaminhar solicitações da Web a um aplicativo ASP.NET Core de back-end que executa o [servidor Kestrel](#), chamado o [modelo de hospedagem de fora do processo](#).

Versões do Windows compatíveis:

- Windows 7 ou posterior
- Windows Server 2008 R2 ou posterior

Ao fazer uma hospedagem em processo, o módulo usa uma implementação de servidor em processo do IIS, chamado Servidor HTTP do IIS (`IISHttpServer`).

Ao hospedar de fora do processo, o módulo só funciona com o Kestrel. O módulo é incompatível com [HTTP.sys](#).

## Modelos de hospedagem

### Modelo de hospedagem em processo

Para configurar um aplicativo para hospedagem em processo, adicione a propriedade `<AspNetCoreHostingModel>` ao arquivo de projeto do aplicativo com um valor de `InProcess` (a hospedagem de fora do processo é definida com `OutOfProcess`):

```
<PropertyGroup>
  <AspNetCoreHostingModel>InProcess</AspNetCoreHostingModel>
</PropertyGroup>
```

Não há suporte para o modelo de hospedagem em processo para aplicativos ASP.NET Core direcionados ao .NET Framework.

Se a propriedade `<AspNetCoreHostingModel>` não estiver presente no arquivo, o valor padrão será `OutOfProcess`.

As seguintes características se aplicam ao hospedar em processo:

- O Servidor HTTP do IIS (`IISHttpServer`) é usado, em vez do servidor [Kestrel](#).
- O [requestTimeout atributo](#) não se aplica à hospedagem em processo.
- Não há suporte para o compartilhamento do pool de aplicativos entre aplicativos. Use um pool de aplicativos por aplicativo.
- Ao usar a [Implantação da Web](#) ou inserir manualmente um [arquivo app\\_offline.htm na implantação](#), o aplicativo talvez não seja desligado imediatamente se houver uma conexão aberta. Por exemplo, uma conexão websocket pode atrasar o desligamento do aplicativo.

- A arquitetura (número de bit) do aplicativo e o tempo de execução instalado (x64 ou x86) devem corresponder à arquitetura do pool de aplicativos.
- Se a configuração manual do host do aplicativo com `WebHostBuilder` (não usando `CreateDefaultBuilder`) e o aplicativo não forem executados diretamente no servidor Kestrel (auto-hospedado), chame `UseKestrel` antes de chamar `UseIISIntegration`. Se a ordem for invertida, a inicialização do host falhará.
- As desconexões do cliente são detectadas. O token de cancelamento `HttpContext.RequestAborted` é cancelado quando o cliente se desconecta.
- `GetCurrentDirectory` retorna o diretório de trabalho do processo iniciado pelo IIS em vez de o diretório do aplicativo (por exemplo, `C:\Windows\System32\inetsrv` para `w3wp.exe`).

Para obter o código de exemplo que define o diretório atual do aplicativo, confira a classe `CurrentDirectoryHelpers`. Chame o método `SetCurrentDirectory`. As chamadas seguintes a `GetCurrentDirectory` fornecem o diretório do aplicativo.

### **Modelo de hospedagem de fora do processo**

Para configurar um aplicativo para hospedagem fora do processo, use qualquer uma das abordagens a seguir no arquivo de projeto:

- não especifique a propriedade `<AspNetCoreHostingModel>`. Se a propriedade `<AspNetCoreHostingModel>` não estiver presente no arquivo, o valor padrão será `OutOfProcess`.
- Defina o valor da propriedade `<AspNetCoreHostingModel>` como `OutOfProcess` (a hospedagem no processo é definida com `InProcess`):

```
<PropertyGroup>
  <AspNetCoreHostingModel>OutOfProcess</AspNetCoreHostingModel>
</PropertyGroup>
```

O servidor `Kestrel` é usado, em vez do servidor HTTP do IIS (`IISHttpServer`).

### **Alterações no modelo de hospedagem**

Se a configuração `hostingModel` for alterada no arquivo `web.config` (explicado na seção [Configuração a Web.config](#)), o módulo reciclará o processo de trabalho do IIS.

Para o IIS Express, o módulo não recicla o processo de trabalho, mas em vez disso, dispara um desligamento normal do processo atual do IIS Express. A próxima solicitação para o aplicativo gera um novo processo do IIS Express.

### **Nome do processo**

`Process.GetCurrentProcess().ProcessName` relata `w3wp / iisexpress` (em processo) ou `dotnet` (fora do processo).

O Módulo do ASP.NET Core é um módulo nativo do IIS que se conecta ao pipeline do IIS para encaminhar solicitações da Web para aplicativos ASP.NET Core de back-end.

Versões do Windows compatíveis:

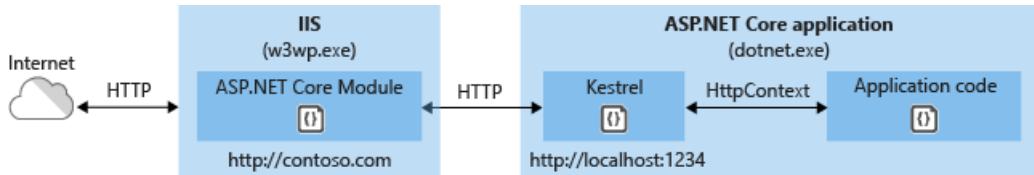
- Windows 7 ou posterior
- Windows Server 2008 R2 ou posterior

O módulo só funciona com o Kestrel. O módulo é incompatível com `HTTP.sys`.

Como os aplicativos ASP.NET Core são executados em um processo separado do processo de trabalho do IIS, o módulo também realiza o gerenciamento de processos. O módulo inicia o

processo para o aplicativo ASP.NET Core quando a primeira solicitação chega e reinicia o aplicativo quando ele falha. Isso é basicamente o mesmo comportamento que o dos aplicativos ASP.NET 4.x que são executados dentro do processo do IIS e são gerenciados pelo [WAS \(Serviço de Ativação de Processos do Windows\)](#).

O diagrama a seguir ilustra a relação entre o IIS, o Módulo do ASP.NET Core e um aplicativo:



As solicitações chegam da Web para o driver do HTTP.sys no modo kernel. O driver roteia as solicitações ao IIS na porta configurada do site, normalmente, a 80 (HTTP) ou a 443 (HTTPS). O módulo encaminha as solicitações ao Kestrel em uma porta aleatória do aplicativo, que não seja a porta 80 ou 443.

O módulo especifica a porta por meio de uma variável de ambiente na inicialização e o middleware de integração do IIS configura o servidor para escutar em `http://localhost:{port}`. Outras verificações são executadas e as solicitações que não se originam do módulo são rejeitadas. O módulo não é compatível com encaminhamento de HTTPS, portanto, as solicitações são encaminhadas por HTTP, mesmo se recebidas pelo IIS por HTTPS.

Depois que o Kestrel coleta a solicitação do módulo, a solicitação é enviada por push ao pipeline do middleware do ASP.NET Core. O pipeline do middleware manipula a solicitação e a passa como uma instância de `HttpContext` para a lógica do aplicativo. O middleware adicionado pela integração do IIS atualiza o esquema, o IP remoto e pathbase para encaminhar a solicitação para o Kestrel. A resposta do aplicativo é retornada ao IIS, que a retorna por push para o cliente HTTP que iniciou a solicitação.

Muitos módulos nativos, como a Autenticação do Windows, permanecem ativos. Para saber mais sobre módulos do IIS ativos com o Módulo do ASP.NET Core, confira [Módulos do IIS com o ASP.NET Core](#).

O Módulo do ASP.NET Core também pode:

- Definir variáveis de ambiente para o processo de trabalho.
- Registrar a saída StdOut no armazenamento de arquivo para a solução de problemas de inicialização.
- Encaminhar tokens de autenticação do Windows.

## Como instalar e usar o Módulo do ASP.NET Core

Para obter instruções sobre como instalar e usar o Módulo do ASP.NET Core, confira [Hospedar o ASP.NET Core no Windows com o IIS](#).

## Configuração com web.config

O Módulo do ASP.NET Core está configurado com a seção `aspNetCore` do nó `system.webServer` no arquivo `web.config` do site.

O seguinte arquivo `web.config` é publicado para uma [implantação dependente de estrutura](#) e configura o Módulo do ASP.NET Core para manipular solicitações de site:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <location path=". " inheritInChildApplications="false">
        <system.webServer>
            <handlers>
                <add name="aspNetCore" path="*" verb="*" modules="AspNetCoreModuleV2"
resourceType="Unspecified" />
            </handlers>
            <aspNetCore processPath="dotnet"
                        arguments=".\\MyApp.dll"
                        stdoutLogEnabled="false"
                        stdoutLogFile=".\\logs\\stdout"
                        hostingModel="InProcess" />
        </system.webServer>
    </location>
</configuration>
```

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <system.webServer>
        <handlers>
            <add name="aspNetCore" path="*" verb="*" modules="AspNetCoreModule"
resourceType="Unspecified" />
        </handlers>
        <aspNetCore processPath="dotnet"
                    arguments=".\\MyApp.dll"
                    stdoutLogEnabled="false"
                    stdoutLogFile=".\\logs\\stdout" />
    </system.webServer>
</configuration>
```

O seguinte *web.config* é publicado para uma [implantação autossuficiente](#):

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <location path=". " inheritInChildApplications="false">
        <system.webServer>
            <handlers>
                <add name="aspNetCore" path="*" verb="*" modules="AspNetCoreModuleV2"
resourceType="Unspecified" />
            </handlers>
            <aspNetCore processPath=".\\MyApp.exe"
                        stdoutLogEnabled="false"
                        stdoutLogFile=".\\logs\\stdout"
                        hostingModel="InProcess" />
        </system.webServer>
    </location>
</configuration>
```

A propriedade [InheritInChildApplications](#) é definida como `false` para indicar que as configurações especificadas no elemento [`<location>`](#) não são herdadas por aplicativos que residem em um subdiretório do aplicativo.

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.webServer>
    <handlers>
      <add name="aspNetCore" path="*" verb="*" modules="AspNetCoreModule"
resourceType="Unspecified" />
    </handlers>
    <aspNetCore processPath=".\\MyApp.exe"
      stdoutLogEnabled="false"
      stdoutLogFile=".\\logs\\stdout" />
  </system.webServer>
</configuration>

```

Quando um aplicativo é implantado no [Serviço de Aplicativo do Azure](#), o caminho `stdoutLogFile` é definido para `\\?\%home%\LogFiles\stdout`. O caminho salva logs de stdout para a pasta `LogFiles`, que é um local criado automaticamente pelo serviço.

Para saber mais sobre a configuração de subaplicativos do IIS, confira [Hospedar o ASP.NET Core no Windows com o IIS](#).

### Atributos do elemento `aspNetCore`

ATRIBUTO	DESCRIÇÃO	PADRÃO
<code>arguments</code>	<p>Atributo de cadeia de caracteres opcional.</p> <p>Argumentos para o executável especificado em <code>processPath</code>.</p>	
<code>disableStartupErrorPage</code>	<p>Atributo booleano opcional.</p> <p>Se for true, a página <b>502.5 – Falha do Processo</b> será suprimida e a página de código de status 502, configurada no <code>web.config</code>, terá precedência.</p>	<code>false</code>
<code>forwardWindowsAuthToken</code>	<p>Atributo booleano opcional.</p> <p>Se for true, o token será encaminhado para o processo filho escutando em <code>%ASPNETCORE_PORT%</code> como um cabeçalho 'MS-ASPNETCORE-WINAUTHTOKEN' por solicitação. É responsabilidade desse processo chamar <code>CloseHandle</code> nesse token por solicitação.</p>	<code>true</code>

ATRIBUTO	DESCRIÇÃO	PADRÃO
<code>hostingModel</code>	<p>Atributo de cadeia de caracteres opcional.</p> <p>Especifica o modelo de hospedagem como em processo (<code>InProcess</code>) ou de fora do processo (<code>OutOfProcess</code>).</p>	<code>OutOfProcess</code>
<code>processesPerApplication</code>	<p>Atributo inteiro opcional.</p> <p>Especifica o número de instâncias do processo especificado na configuração <b>processPath</b> que pode ser ativada por aplicativo.</p> <p>†Para hospedagem em processo, o valor está limitado a <code>1</code>.</p>	Padrão: <code>1</code> Mín.: <code>1</code> Máx.: <code>100</code> +
<code>processPath</code>	<p>Atributo de cadeia de caracteres obrigatório.</p> <p>Caminho para o executável que inicia um processo que escuta solicitações HTTP.</p> <p>Caminhos relativos são compatíveis. Se o caminho começa com <code>.</code>, o caminho é considerado relativo à raiz do site.</p>	
<code>rapidFailsPerMinute</code>	<p>Atributo inteiro opcional.</p> <p>Especifica o número de vezes que o processo especificado em <b>processPath</b> pode falhar por minuto. Se esse limite for excedido, o módulo interromperá a inicialização do processo pelo restante do minuto.</p> <p>Sem suporte com hospedagem padrão.</p>	Padrão: <code>10</code> Mín.: <code>0</code> Máx.: <code>100</code>

ATRIBUTO	DESCRIÇÃO	PADRÃO
<code>requestTimeout</code>	<p>Atributo de intervalo de tempo opcional.</p> <p>Especifica a duração para qual o Módulo do ASP.NET Core aguarda uma resposta do processo que escuta em <code>%ASPNETCORE_PORT%</code>.</p> <p>Em versões do Módulo do ASP.NET Core que acompanham a versão do ASP.NET Core 2.1 ou posterior, o <code>requestTimeout</code> é especificado em horas, minutos e segundos.</p> <p>Não se aplica à hospedagem em processo. Para a hospedagem em processo, o módulo aguarda o aplicativo processar a solicitação.</p>	Padrão: <code>00:02:00</code> Mín.: <code>00:00:00</code> Máx.: <code>360:00:00</code>
<code>shutdownTimeLimit</code>	<p>Atributo inteiro opcional.</p> <p>Duração em segundos que o módulo espera para o executável desligar normalmente quando o arquivo <code>app_offline.htm</code> é detectado.</p>	Padrão: <code>10</code> Mín.: <code>0</code> Máx.: <code>600</code>
<code>startupTimeLimit</code>	<p>Atributo inteiro opcional.</p> <p>Duração em segundos que o módulo espera para o arquivo executável iniciar um processo escutando na porta. Se esse tempo limite é excedido, o módulo encerra o processo. O módulo tentará reiniciar o processo quando ele receber uma nova solicitação e continuará a tentar reiniciar o processo em solicitações subsequentes de entrada, a menos que o aplicativo falhe em iniciar um número de vezes igual a <code>rapidFailsPerMinute</code> no último minuto sem interrupção.</p> <p>Um valor de 0 (zero) <b>não</b> é considerado um tempo limite infinito.</p>	Padrão: <code>120</code> Mín.: <code>0</code> Máx.: <code>3600</code>

ATRIBUTO	DESCRIÇÃO	PADRÃO
<code>stdoutLogEnabled</code>	Atributo booleano opcional.  Se for true, <b>stdout</b> e <b>stderr</b> para o processo especificado em <b>processPath</b> serão redirecionados para o arquivo especificado em <b>stdoutLogFile</b> .	<code>false</code>
<code>stdoutLogFile</code>	Atributo de cadeia de caracteres opcional.  Especifica o caminho relativo ou absoluto para o qual <b>stdout</b> e <b>stderr</b> do processo especificado em <b>processPath</b> são registrados em log. Os caminhos relativos são relativos à raiz do site. Qualquer caminho começando com <code>.</code> é relativo à raiz do site e todos os outros caminhos são tratados como caminhos absolutos. Todas as pastas fornecidas no caminho são criadas pelo módulo quando o arquivo de log é criado. Usando delimitadores de sublinhado, um carimbo de data/hora, uma ID de processo e a extensão de arquivo ( <i>.log</i> ) são adicionados ao último segmento do caminho <b>stdoutLogFile</b> . Se <code>.\logs\stdout</code> é fornecido como um valor, um log de exemplo stdout é salvo como <i>stdout_20180205194132_1934.log</i> na pasta <i>logs</i> quando salvos em 5/2/2018, às 19:41:32, com uma ID de processo de 1934.	<code>aspnetcore-stdout</code>
ATRIBUTO	DESCRIÇÃO	PADRÃO
<code>arguments</code>	Atributo de cadeia de caracteres opcional.  Argumentos para o executável especificado em <b>processPath</b> .	

ATRIBUTO	DESCRIÇÃO	PADRÃO
<code>disableStartUpErrorPage</code>	Atributo booleano opcional.  Se for true, a página <b>502.5 – Falha do Processo</b> será suprimida e a página de código de status 502, configurada no <code>web.config</code> , terá precedência.	<code>false</code>
<code>forwardWindowsAuthToken</code>	Atributo booleano opcional.  Se for true, o token será encaminhado para o processo filho escutando em <code>%ASPNETCORE_PORT%</code> como um cabeçalho 'MS-ASPNETCORE-WINAUTHTOKEN' por solicitação. É responsabilidade desse processo chamar <code>CloseHandle</code> nesse token por solicitação.	<code>true</code>
<code>processesPerApplication</code>	Atributo inteiro opcional.  Especifica o número de instâncias do processo especificado na configuração <b>processPath</b> que pode ser ativada por aplicativo.	Padrão: <code>1</code> Mín.: <code>1</code> Máx.: <code>100</code>
<code>processPath</code>	Atributo de cadeia de caracteres obrigatório.  Caminho para o executável que inicia um processo que escuta solicitações HTTP. Caminhos relativos são compatíveis. Se o caminho começa com <code>.</code> , o caminho é considerado relativo à raiz do site.	
<code>rapidFailsPerMinute</code>	Atributo inteiro opcional.  Especifica o número de vezes que o processo especificado em <b>processPath</b> pode falhar por minuto. Se esse limite for excedido, o módulo interromperá a inicialização do processo pelo restante do minuto.	Padrão: <code>10</code> Mín.: <code>0</code> Máx.: <code>100</code>

ATRIBUTO	DESCRIÇÃO	PADRÃO
<code>requestTimeout</code>	<p>Atributo de intervalo de tempo opcional.</p> <p>Especifica a duração para qual o Módulo do ASP.NET Core aguarda uma resposta do processo que escuta em %ASPNETCORE_PORT%.</p> <p>Em versões do Módulo do ASP.NET Core que acompanham a versão do ASP.NET Core 2.1 ou posterior, o <code>requestTimeout</code> é especificado em horas, minutos e segundos.</p>	Padrão: <code>00:02:00</code> Mín.: <code>00:00:00</code> Máx.: <code>360:00:00</code>
<code>shutdownTimeLimit</code>	<p>Atributo inteiro opcional.</p> <p>Duração em segundos que o módulo espera para o executável desligar normalmente quando o arquivo <code>app_offline.htm</code> é detectado.</p>	Padrão: <code>10</code> Mín.: <code>0</code> Máx.: <code>600</code>
<code>startupTimeLimit</code>	<p>Atributo inteiro opcional.</p> <p>Duração em segundos que o módulo espera para o arquivo executável iniciar um processo escutando na porta. Se esse tempo limite é excedido, o módulo encerra o processo. O módulo tentará reiniciar o processo quando ele receber uma nova solicitação e continuará a tentar reiniciar o processo em solicitações subsequentes de entrada, a menos que o aplicativo falhe em iniciar um número de vezes igual a <code>rapidFailsPerMinute</code> no último minuto sem interrupção.</p> <p>Um valor de 0 (zero) <b>não</b> é considerado um tempo limite infinito.</p>	Padrão: <code>120</code> Mín.: <code>0</code> Máx.: <code>3600</code>

ATRIBUTO	DESCRIÇÃO	PADRÃO
<code>stdoutLogEnabled</code>	Atributo booleano opcional.  Se for true, <b>stdout</b> e <b>stderr</b> para o processo especificado em <b>processPath</b> serão redirecionados para o arquivo especificado em <b>stdoutLogFile</b> .	<code>false</code>
<code>stdoutLogFile</code>	Atributo de cadeia de caracteres opcional.  Especifica o caminho relativo ou absoluto para o qual <b>stdout</b> e <b>stderr</b> do processo especificado em <b>processPath</b> são registrados em log. Os caminhos relativos são relativos à raiz do site. Qualquer caminho começando com <code>.</code> é relativo à raiz do site e todos os outros caminhos são tratados como caminhos absolutos. As pastas fornecidas no caminho devem existir para que o módulo crie o arquivo de log. Usando delimitadores de sublinhado, um carimbo de data/hora, uma ID de processo e a extensão de arquivo ( <i>.log</i> ) são adicionados ao último segmento do caminho <b>stdoutLogFile</b> . Se <code>.\logs\stdout</code> é fornecido como um valor, um log de exemplo stdout é salvo como <i>stdout_20180205194132_1934.log</i> na pasta <i>logs</i> quando salvos em 5/2/2018, às 19:41:32, com uma ID de processo de 1934.	<code>aspnetcore-stdout</code>

ATRIBUTO	DESCRIÇÃO	PADRÃO
<code>arguments</code>	Atributo de cadeia de caracteres opcional.  Argumentos para o executável especificado em <b>processPath</b> .	

ATRIBUTO	DESCRIÇÃO	PADRÃO
<code>disableStartUpErrorPage</code>	Atributo booleano opcional.  Se for true, a página <b>502.5 – Falha do Processo</b> será suprimida e a página de código de status 502, configurada no <i>web.config</i> , terá precedência.	<code>false</code>
<code>forwardWindowsAuthToken</code>	Atributo booleano opcional.  Se for true, o token será encaminhado para o processo filho escutando em %ASPNETCORE_PORT% como um cabeçalho 'MS-ASPNETCORE-WINAUTHTOKEN' por solicitação. É responsabilidade desse processo chamar CloseHandle nesse token por solicitação.	<code>true</code>
<code>processesPerApplication</code>	Atributo inteiro opcional.  Especifica o número de instâncias do processo especificado na configuração <b>processPath</b> que pode ser ativada por aplicativo.	Padrão: <code>1</code> Mín.: <code>1</code> Máx.: <code>100</code>
<code>processPath</code>	Atributo de cadeia de caracteres obrigatório.  Caminho para o executável que inicia um processo que escuta solicitações HTTP. Caminhos relativos são compatíveis. Se o caminho começa com <code>.</code> , o caminho é considerado relativo à raiz do site.	
<code>rapidFailsPerMinute</code>	Atributo inteiro opcional.  Especifica o número de vezes que o processo especificado em <b>processPath</b> pode falhar por minuto. Se esse limite for excedido, o módulo interromperá a inicialização do processo pelo restante do minuto.	Padrão: <code>10</code> Mín.: <code>0</code> Máx.: <code>100</code>

ATRIBUTO	DESCRIÇÃO	PADRÃO
<code>requestTimeout</code>	<p>Atributo de intervalo de tempo opcional.</p> <p>Especifica a duração para qual o Módulo do ASP.NET Core aguarda uma resposta do processo que escuta em <code>%ASPNETCORE_PORT%</code>.</p> <p>Em versões do Módulo ASP.NET Core que acompanham a versão do ASP.NET Core 2.0 ou anterior, o <code>requestTimeout</code> deve ser especificado somente em minutos inteiros, caso contrário, ele assume o valor padrão de 2 minutos.</p>	Padrão: <code>00:02:00</code> Mín.: <code>00:00:00</code> Máx.: <code>360:00:00</code>
<code>shutdownTimeLimit</code>	<p>Atributo inteiro opcional.</p> <p>Duração em segundos que o módulo espera para o executável desligar normalmente quando o arquivo <code>app_offline.htm</code> é detectado.</p>	Padrão: <code>10</code> Mín.: <code>0</code> Máx.: <code>600</code>
<code>startupTimeLimit</code>	<p>Atributo inteiro opcional.</p> <p>Duração em segundos que o módulo espera para o arquivo executável iniciar um processo escutando na porta. Se esse tempo limite é excedido, o módulo encerra o processo. O módulo tentará reiniciar o processo quando ele receber uma nova solicitação e continuará a tentar reiniciar o processo em solicitações subsequentes de entrada, a menos que o aplicativo falhe em iniciar um número de vezes igual a <code>rapidFailsPerMinute</code> no último minuto sem interrupção.</p>	Padrão: <code>120</code> Mín.: <code>0</code> Máx.: <code>3600</code>
<code>stdoutLogEnabled</code>	<p>Atributo booleano opcional.</p> <p>Se for true, <code>stdout</code> e <code>stderr</code> para o processo especificado em <code>processPath</code> serão redirecionados para o arquivo especificado em <code>stdoutLogFile</code>.</p>	<code>false</code>

ATRIBUTO	DESCRIÇÃO	PADRÃO
<code>stdoutLogFile</code>	<p>Atributo de cadeia de caracteres opcional.</p> <p>Especifica o caminho relativo ou absoluto para o qual <b>stdout</b> e <b>stderr</b> do processo especificado em <b>processPath</b> são registrados em log. Os caminhos relativos são relativos à raiz do site. Qualquer caminho começando com <code>.</code> é relativo à raiz do site e todos os outros caminhos são tratados como caminhos absolutos. As pastas fornecidas no caminho devem existir para que o módulo crie o arquivo de log. Usando delimitadores de sublinhado, um carimbo de data/hora, uma ID de processo e a extensão de arquivo (<i>.log</i>) são adicionados ao último segmento do caminho <b>stdoutLogFile</b>. Se <code>.\logs\stdout</code> é fornecido como um valor, um log de exemplo stdout é salvo como <i>stdout_20180205194132_1934.log</i> na pasta <i>logs</i> quando salvos em 5/2/2018, às 19:41:32, com uma ID de processo de 1934.</p>	<code>aspnetcore-stdout</code>

## Definindo variáveis de ambiente

Variáveis de ambiente podem ser especificadas para o processo no atributo `processPath`. Especificar uma variável de ambiente com o elemento filho `<environmentVariable>` de um elemento de coleção `<environmentVariables>`. Variáveis de ambiente definidas nesta seção têm precedência sobre variáveis de ambiente do sistema.

Variáveis de ambiente podem ser especificadas para o processo no atributo `processPath`. Especificar uma variável de ambiente com o elemento filho `<environmentVariable>` de um elemento de coleção `<environmentVariables>`.

### WARNING

As variáveis de ambiente definidas nesta seção são conflitantes com as variáveis de ambiente do sistema definidas com o mesmo nome. Quando a variável de ambiente é definida no arquivo *web.config* e no nível do sistema do Windows, o valor do arquivo *web.config* fica anexado ao valor da variável de ambiente do sistema (por exemplo, `ASPNETCORE_ENVIRONMENT: Development;Development`), o que impede a inicialização do aplicativo.

O exemplo a seguir define duas variáveis de ambiente. `ASPNETCORE_ENVIRONMENT` configura o

ambiente do aplicativo para `Development`. Um desenvolvedor pode definir esse valor temporariamente no arquivo `web.config` para forçar o carregamento da [Página de Exceções do Desenvolvedor](#) ao depurar uma exceção de aplicativo. `CONFIG_DIR` é um exemplo de uma variável de ambiente definida pelo usuário, em que o desenvolvedor escreveu código que lê o valor de inicialização para formar um caminho no qual carregar o arquivo de configuração do aplicativo.

```
<aspNetCore processPath="dotnet"
    arguments=".\\MyApp.dll"
    stdoutLogEnabled="false"
    stdoutLogFile="\\?\%home%\LogFiles\stdout"
    hostingModel="InProcess">
<environmentVariables>
    <environmentVariable name="ASPNETCORE_ENVIRONMENT" value="Development" />
    <environmentVariable name="CONFIG_DIR" value="f:\\application_config" />
</environmentVariables>
</aspNetCore>
```

```
<aspNetCore processPath="dotnet"
    arguments=".\\MyApp.dll"
    stdoutLogEnabled="false"
    stdoutLogFile="\\?\%home%\LogFiles\stdout">
<environmentVariables>
    <environmentVariable name="ASPNETCORE_ENVIRONMENT" value="Development" />
    <environmentVariable name="CONFIG_DIR" value="f:\\application_config" />
</environmentVariables>
</aspNetCore>
```

#### NOTE

Em vez de configurar o ambiente diretamente no `web.config`, você pode incluir a propriedade `<EnvironmentName>` no perfil de publicação (`pubxml`) ou no perfil de projeto. Esta abordagem define o ambiente no arquivo `web.config` quando o projeto é publicado:

```
<PropertyGroup>
    <EnvironmentName>Development</EnvironmentName>
</PropertyGroup>
```

#### WARNING

Defina a variável de ambiente apenas `ASPNETCORE_ENVIRONMENT` para `Development` em servidores de preparo e de teste que não estão acessíveis a redes não confiáveis, tais como a Internet.

## app\_offline.htm

Se um arquivo com o nome `app_offline.htm` é detectado no diretório raiz de um aplicativo, o Módulo do ASP.NET Core tenta desligar normalmente o aplicativo e parar o processamento de solicitações de entrada. Se o aplicativo ainda está em execução após o número de segundos definido em `shutdownTimeLimit`, o Módulo do ASP.NET Core encerra o processo em execução.

Enquanto o arquivo `app_offline.htm` estiver presente, o Módulo do ASP.NET Core responderá às solicitações enviando o conteúdo do arquivo `app_offline.htm`. Quando o arquivo `app_offline.htm` é removido, a próxima solicitação inicia o aplicativo.

Ao usar o modelo de hospedagem de fora do processo, talvez o aplicativo não desligue imediatamente se houver uma conexão aberta. Por exemplo, uma conexão websocket pode

atrasar o desligamento do aplicativo.

## Página de erro de inicialização

A hospedagem em processo e fora do processo produzem páginas de erro personalizadas quando falham ao iniciar o aplicativo.

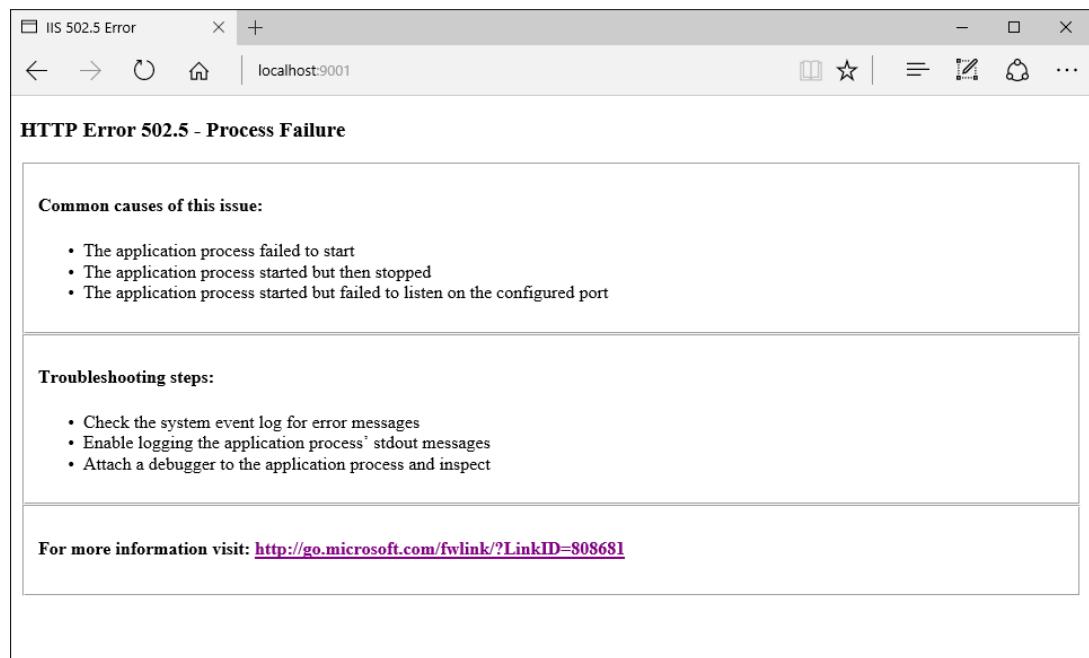
Se o Módulo do ASP.NET Core falhar ao encontrar o manipulador de solicitação em processo ou fora do processo, uma página de código de status *500.0 – falha ao carregar manipulador no processo/fora do processo* será exibida.

No caso da hospedagem em processo, se o módulo do ASP.NET Core falhar ao iniciar o aplicativo, uma página de código de status *500.30 – falha ao iniciar* será exibida.

Para hospedagem fora do processo, se o Módulo do ASP.NET Core falhar ao iniciar o processo de back-end ou se o processo de back-end iniciar, mas falhar ao escutar na porta configurada, uma página de código de status *502.5 – falha no processo* será exibida.

Para suprimir essa página e reverter para a página de código de status 5xx padrão do IIS, use o atributo `disableStartUpErrorPage`. Para obter mais informações sobre como configurar mensagens de erro personalizadas, veja [Erros HTTP <httpErrors>](#).

Se o Módulo do ASP.NET Core falhar ao iniciar o processo de back-end ou se o processo de back-end iniciar, mas falhar ao escutar na porta configurada, uma página de código de status *502.5 – falha no processo* será exibida. Para omitir esta página e reverter para a página de código de status 502 padrão do IIS, use o atributo `disableStartUpErrorPage`. Para obter mais informações sobre como configurar mensagens de erro personalizadas, veja [Erros HTTP <httpErrors>](#).



## Criação de log e redirecionamento

O Módulo do ASP.NET Core redireciona as saídas de console `stdout` e `stderr` para o disco se os atributos `stdoutLogEnabled` e `stdoutLogFile` do elemento `aspNetCore` forem definidos. Todas as pastas no caminho `stdoutLogFile` são criadas pelo módulo quando o arquivo de log é criado. O pool de aplicativos deve ter acesso de gravação ao local em que os logs foram gravados (use `IIS AppPool\<app_pool_name>` para fornecer permissão de gravação).

Logs não sofrem rotação, a menos que ocorra a reciclagem/reinicialização do processo. É

responsabilidade do hoster limitar o espaço em disco consumido pelos logs.

Usar o log de stdout é recomendado apenas para solucionar problemas de inicialização do aplicativo. Não use o log de stdout para fins gerais de registro em log do aplicativo. Para registro em log de rotina em um aplicativo ASP.NET Core, use uma biblioteca de registro em log que limita o tamanho do arquivo de log e realiza a rotação de logs. Para obter mais informações, veja [provedores de log de terceiros](#).

Uma extensão de arquivo e um carimbo de data/hora são adicionados automaticamente quando o arquivo de log é criado. O nome do arquivo de log é composto por meio do acréscimo do carimbo de data/hora, da ID do processo e da extensão de arquivo (.log) para o último segmento do caminho `stdoutLogFile` (normalmente `stdout`), delimitados por sublinhados. Se o caminho `stdoutLogFile` termina com `stdout`, um log para um aplicativo com um PID de 1934, criado em 5/2/2018 às 19:42:32, tem o nome de arquivo `stdout_20180205194132_1934.log`.

Se `stdoutEnabled` for falso, os erros que ocorrerem na inicialização do aplicativo serão capturados e emitidos no log de eventos até 30 KB. Após a inicialização, todos os logs adicionais são descartados.

O elemento `aspNetCore` de exemplo a seguir configura o registro em log de stdout para um aplicativo hospedado no Serviço de Aplicativo do Azure. Um caminho local ou um caminho de compartilhamento de rede é aceitável para o registro em log local. Confirme se a identidade do usuário AppPool tem permissão para gravar no caminho fornecido.

```
<aspNetCore processPath="dotnet"
            arguments=".\\MyApp.dll"
            stdoutEnabled="true"
            stdoutLogFile="\\?\%home%\LogFiles\stdout"
            hostingModel="InProcess">
</aspNetCore>
```

```
<aspNetCore processPath="dotnet"
            arguments=".\\MyApp.dll"
            stdoutEnabled="true"
            stdoutLogFile="\\?\%home%\LogFiles\stdout">
</aspNetCore>
```

## Logs de diagnóstico avançados

O Módulo do ASP.NET Core é configurável para fornecer logs de diagnóstico avançados.

Adicione o elemento `<handlerSettings>` ao elemento `<aspNetCore>` no `web.config`. A definição de `debugLevel` como `TRACE` expõe uma fidelidade maior de informações de diagnóstico:

```
<aspNetCore processPath="dotnet"
            arguments=".\\MyApp.dll"
            stdoutEnabled="false"
            stdoutLogFile="\\?\%home%\LogFiles\stdout"
            hostingModel="InProcess">
<handlerSettings>
    <handlerSetting name="debugFile" value="aspnetcore-debug.log" />
    <handlerSetting name="debugLevel" value="FILE,TRACE" />
</handlerSettings>
</aspNetCore>
```

Os valores do nível de depuração (`debugLevel`) podem incluir o nível e a localização.

Níveis (na ordem do menos para o mais detalhado):

- ERROR
- WARNING
- INFO
- TRACE

Locais (vários locais são permitidos):

- CONSOLE
- EVENTLOG
- FILE

As configurações do manipulador também podem ser fornecidas por meio de variáveis de ambiente:

- `ASPNETCORE_MODULE_DEBUG_FILE` – caminho para o arquivo de log de depuração. (Padrão: `aspnetcore-debug.log`)
- `ASPNETCORE_MODULE_DEBUG` – configuração do nível de depuração.

#### **WARNING**

**Não** deixe o log de depuração habilitado na implantação por mais tempo que o necessário para solucionar um problema. O tamanho do log não é limitado. Deixar o log de depuração habilitado pode esgotar o espaço em disco disponível e causar falha no servidor ou no serviço de aplicativo.

Veja [Configuração com web.config](#) para obter um exemplo do elemento `aspNetCore` no arquivo `web.config`.

## A configuração de proxy usa o protocolo HTTP e um token de emparelhamento

*Só se aplica à hospedagem de fora do processo.*

O proxy criado entre o Módulo do ASP.NET Core e o Kestrel usa o protocolo HTTP. O uso de HTTP é uma otimização de desempenho na qual o tráfego entre o módulo e o Kestrel ocorre em um endereço de loopback fora do adaptador de rede. Não há nenhum risco de interceptação do tráfego entre o módulo e o Kestrel em um local fora do servidor.

Um token de emparelhamento é usado para assegurar que as solicitações recebidas pelo Kestrel foram transmitidas por proxy pelo IIS e que não são provenientes de outra origem. O token de emparelhamento é criado e definido em uma variável de ambiente (`ASPNETCORE_TOKEN`) pelo módulo. O token de emparelhamento também é definido em um cabeçalho (`MS-ASPNETCORE-TOKEN`) em cada solicitação com proxy. O Middleware do IIS verifica cada solicitação recebida para confirmar se o valor de cabeçalho do token de emparelhamento corresponde ao valor da variável de ambiente. Se os valores do token forem incompatíveis, a solicitação será registrada em log e rejeitada. A variável de ambiente do token de emparelhamento e o tráfego entre o módulo e o Kestrel não são acessíveis em um local fora do servidor. Sem saber o valor do token de emparelhamento, um invasor não pode enviar solicitações que ignoram a verificação no Middleware do IIS.

## Módulo do ASP.NET Core com uma configuração do IIS compartilhada

O instalador do módulo do ASP.NET Core é executado com os privilégios da conta de **SISTEMA**. Já que a conta de sistema local não tem permissão para modificar o caminho do compartilhamento usado pela configuração compartilhada de IIS, o instalador experimenta um erro de acesso negado ao tentar definir as configurações de módulo em *applicationHost.config* no compartilhamento. Ao usar uma configuração compartilhada de IIS, siga estas etapas:

1. Desabilite a configuração compartilhada de IIS.
2. Execute o instalador.
3. Exportar o arquivo *applicationHost.config* atualizado para o compartilhamento.
4. Reabilite a Configuração Compartilhada do IIS.

## Versão do módulo e logs do instalador do pacote de hospedagem

Para determinar a versão do Módulo do ASP.NET Core instalado:

1. No sistema de hospedagem, navegue até %windir%\System32\inetsrv.
2. Localize o arquivo *aspnetcore.dll*.
3. Clique com o botão direito do mouse no arquivo e selecione **Propriedades** no menu contextual.
4. Selecione a guia **Detalhes**. A **Versão do arquivo** e a **Versão do produto** representam a versão instalada do módulo.

Os logs de instalador do pacote de hospedagem para o módulo são encontrados em C:\Usuários\%UserName%\AppData\Local\Temp. O arquivo é nomeado *dd\_DotNetCoreWinSrvHosting\_<carimbo de data/hora>\_000\_AspNetCoreModule\_x64.log*.

## Locais dos arquivos de módulo, de esquema e de configuração

### Módulo

#### IIS (x86/amd64):

- %windir%\System32\inetsrv\aspnetcore.dll
- %windir%\SysWOW64\inetsrv\aspnetcore.dll
- %ProgramFiles%\IIS\Asp.Net Core Module\V2\aspnetcorev2.dll
- %ProgramFiles(x86)%\IIS\Asp.Net Core Module\V2\aspnetcorev2.dll

#### IIS Express (x86/amd64):

- %ProgramFiles%\IIS Express\aspnetcore.dll
- %ProgramFiles(x86)%\IIS Express\aspnetcore.dll
- %ProgramFiles%\IIS Express\Asp.Net Core Module\V2\aspnetcorev2.dll
- %ProgramFiles(x86)%\IIS Express\Asp.Net Core Module\V2\aspnetcorev2.dll

### Esquema

#### IIS

- %windir%\System32\inetsrv\config\schema\aspnetcore\_schema.xml
- %windir%\System32\inetsrv\config\schema\aspnetcore\_schema\_v2.xml

## IIS Express

- %ProgramFiles%\IIS Express\config\schema\aspnetcore\_schema.xml
- %ProgramFiles%\IIS Express\config\schema\aspnetcore\_schema\_v2.xml

## Configuração

### IIS

- %windir%\System32\inetsrv\config\applicationHost.config

## IIS Express

- Visual Studio: {APPLICATION ROOT}\.vs\config\applicationHost.config
- *iisexpress.exe* CLI:  
%USERPROFILE%\Documents\IISExpress\config\applicationhost.config

Os arquivos podem ser encontrados pesquisando por *aspnetcore* no arquivo *applicationHost.config*.

## Recursos adicionais

- [Hospedar o ASP.NET Core no Windows com o IIS](#)
- [Repositório do GitHub do Módulo do ASP.NET Core \(origem de referência\)](#)
- [Módulos do IIS com o ASP.NET Core](#)

# Suporte ao IIS no tempo de desenvolvimento no Visual Studio para ASP.NET Core

10/01/2019 • 6 minutes to read • [Edit Online](#)

Por [Sourabh Shirhatti](#) e [Luke Latham](#)

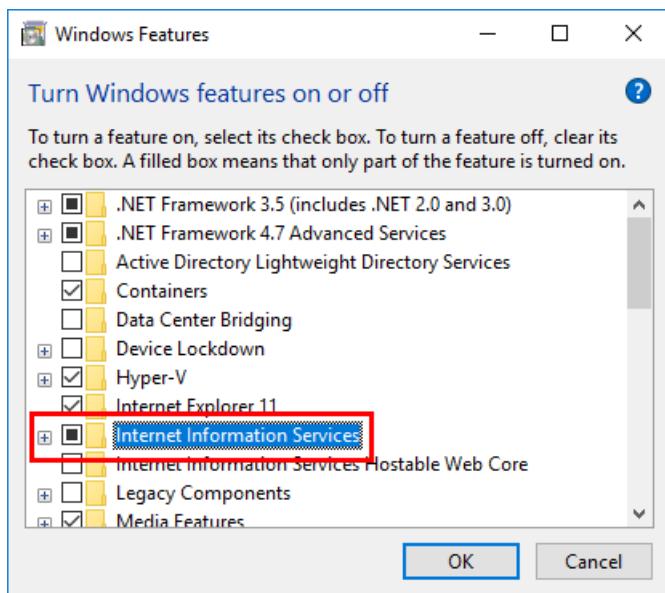
Este artigo descreve o suporte do [Visual Studio](#) a depuração de aplicativos do ASP.NET Core em execução por trás do IIS no Windows Server. Este tópico orienta você sobre como habilitar esse recurso e configurar um projeto.

## Pré-requisitos

- [Visual Studio para Windows](#)
- Carga de trabalho **ASP.NET e desenvolvimento para a Web**
- Carga de trabalho de **desenvolvimento multiplataforma do .NET Core**
- Certificado de segurança X.509

## Habilitar o IIS

1. Navegue para **Painel de Controle > Programas > Programas e Recursos > Ativar ou desativar recursos do Windows** (lado esquerdo da tela).
2. Selecione a caixa de seleção **Serviços de Informações da Internet**.



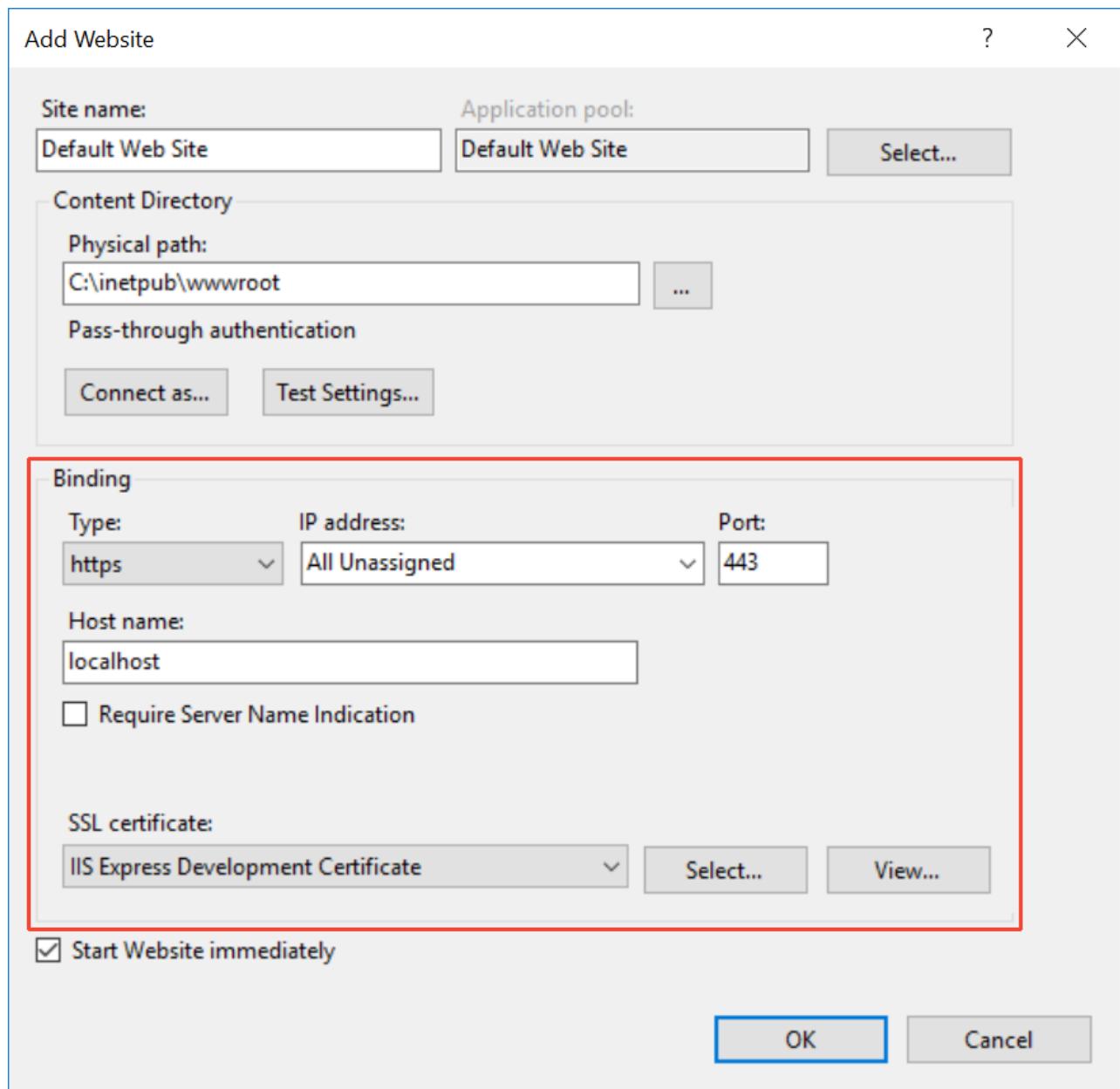
A instalação do IIS pode exigir uma reinicialização do sistema.

## Configurar o IIS

O IIS deve ter um site configurado com o seguinte:

- Um nome do host que corresponda ao nome do host da URL do perfil de inicialização do aplicativo.
- Associação para a porta 443, com um certificado atribuído.

Por exemplo, o **Nome do host** para um site adicionado é definido como "localhost" (o perfil de inicialização também usará "localhost" posteriormente neste tópico). A porta é definida para "443" (HTTPS). O **Certificado de Desenvolvimento do IIS Express** é atribuído ao site, mas nenhum certificado válido funciona:

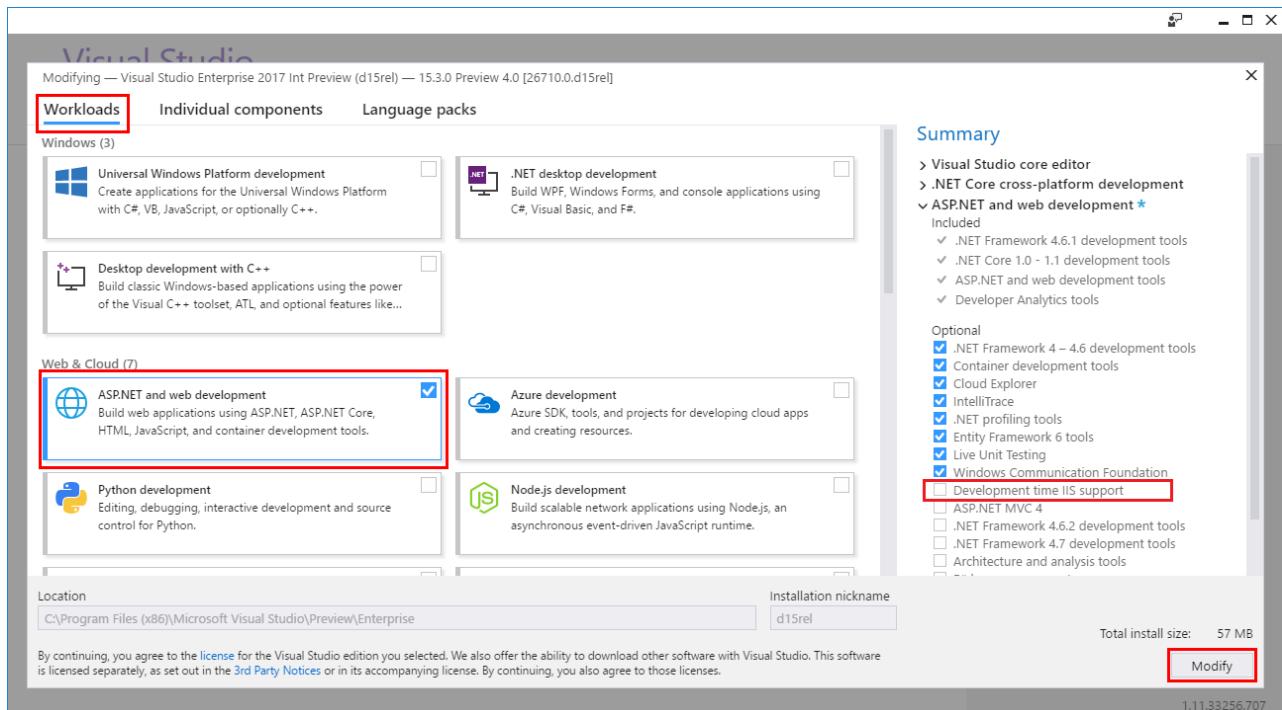


Se a instalação do IIS já tiver um **site da Web padrão** com um nome do host que corresponde ao nome do host da URL do perfil de inicialização do aplicativo:

- Adicione uma associação de porta para a porta 443 (HTTPS).
- Atribua um certificado válido para o site.

## Habilitar o suporte ao IIS no tempo de desenvolvimento no Visual Studio

1. Inicie o Instalador do Visual Studio.
2. Selecione o componente **Suporte ao IIS no tempo de desenvolvimento**. O componente está listado como opcional no painel **Resumo** para a carga de trabalho **Desenvolvimento Web e ASP.NET**. O componente instala o [Módulo do ASP.NET Core](#), que é um módulo nativo do IIS necessário para executar aplicativos ASP.NET Core com o IIS.

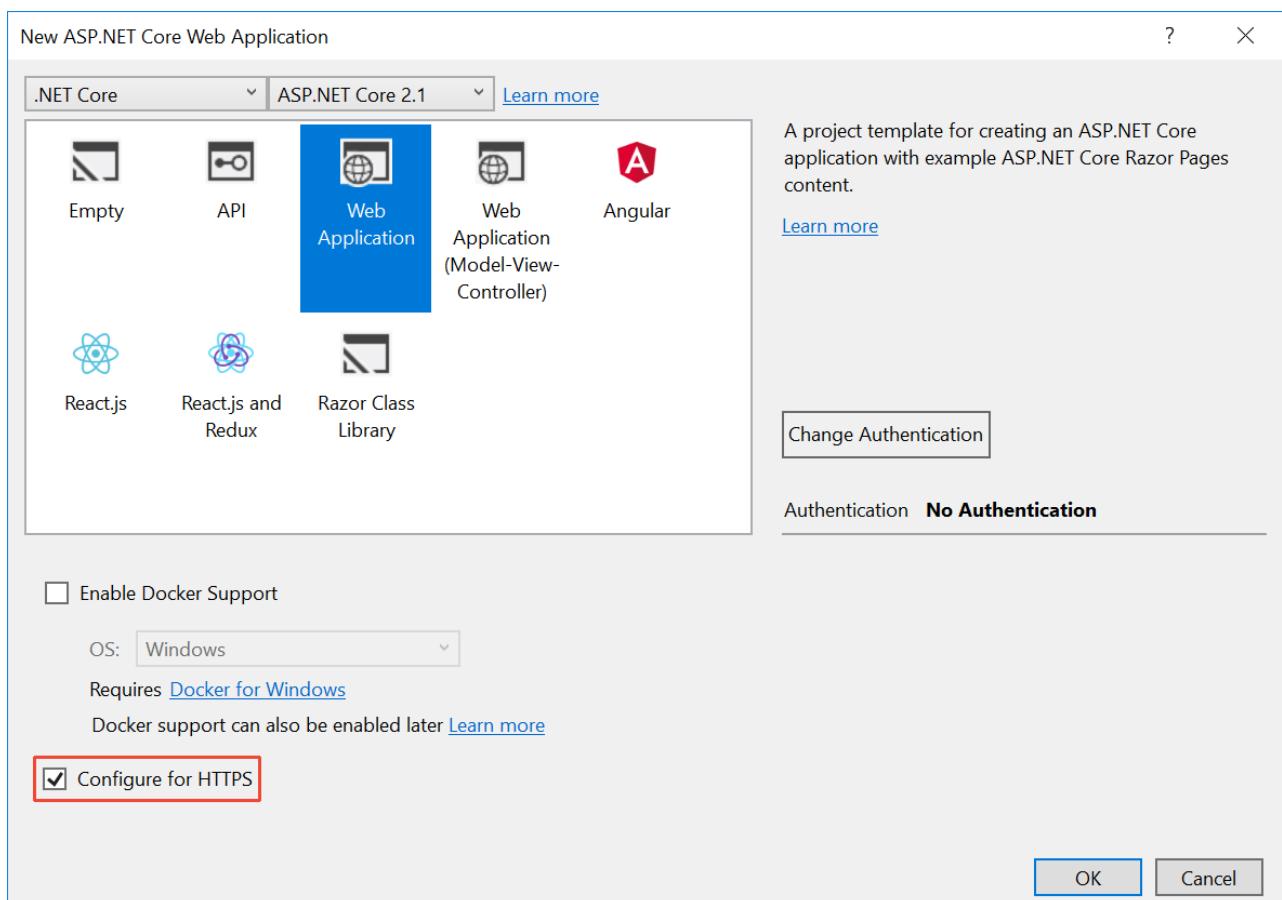


## Configurar o projeto

### Redirecionamento para HTTPS

Para um novo projeto, selecione a caixa de seleção para **Configurar para HTTPS** na janela **Novo Aplicativo**

#### Web ASP.NET Core:



Em um projeto existente, use o middleware de redirecionamento para HTTPS no `Startup.Configure` chamando o método de extensão `UseHttpsRedirection`:

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseCookiePolicy();

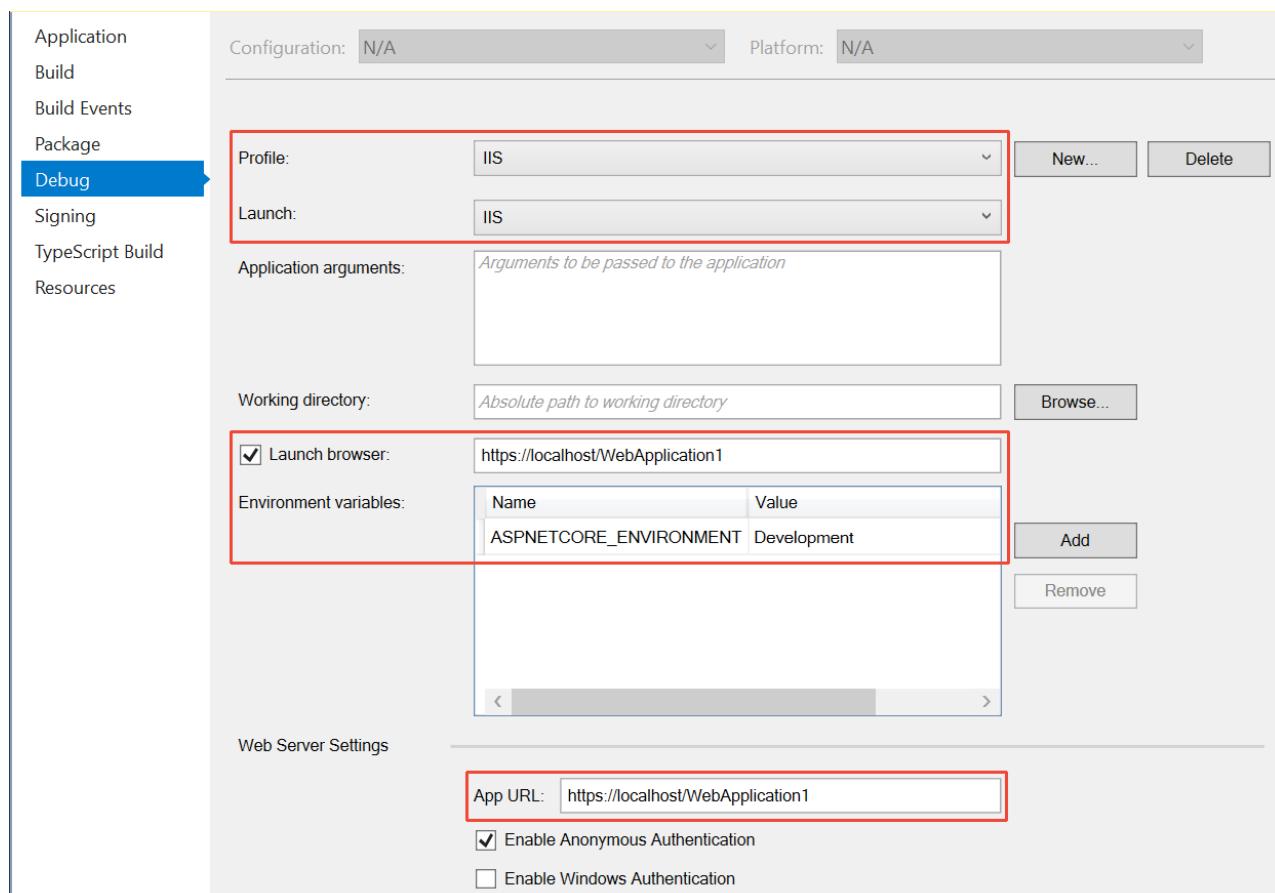
    app.UseMvc();
}

```

## Perfil de inicialização do IIS

Crie um novo perfil de inicialização para adicionar suporte ao IIS no tempo de desenvolvimento:

1. Para **Perfil**, selecione o botão **Novo**. Nomeie o perfil "IIS" na janela pop-up. Selecione **OK** para criar o perfil.
2. Para a configuração **Iniciar**, selecione **IIS** da lista.
3. Selecione a caixa de seleção **Iniciar navegador** e forneça a URL de ponto de extremidade. Use o protocolo **HTTPS**. Este exemplo usa `https://localhost/WebApplication1`.
4. Na seção **Variáveis de ambiente**, selecione o botão **Adicionar**. Fornecer uma variável de ambiente com uma chave `ASPNETCORE_ENVIRONMENT` e um valor `Development`.
5. Na área **Configurações do Servidor Web**, defina a **URL do Aplicativo**. Este exemplo usa `https://localhost/WebApplication1`.
6. Salve o perfil.



Como alternativa, adicione manualmente um perfil de inicialização para o arquivo [launchSettings.json](#) no aplicativo:

```
{  
    "iisSettings": {  
        "windowsAuthentication": false,  
        "anonymousAuthentication": true,  
        "iis": {  
            "applicationUrl": "https://localhost/WebApplication1",  
            "sslPort": 0  
        }  
    },  
    "profiles": {  
        "IIS": {  
            "commandName": "IIS",  
            "launchBrowser": true,  
            "launchUrl": "https://localhost/WebApplication1",  
            "environmentVariables": {  
                "ASPNETCORE_ENVIRONMENT": "Development"  
            }  
        }  
    }  
}
```

## Executar o projeto

No Visual Studio:

- Confirme se a lista suspenso de configuração de compilação está definida para **Depurar**.
- Defina o botão Executar para o perfil do **IIS** e selecione o botão para iniciar o aplicativo.



O Visual Studio poderá solicitar uma reinicialização se não estiver executando como administrador. Se solicitado, reinicie o Visual Studio.

Se for usado um certificado de desenvolvimento não confiável, o navegador poderá exigir a criação de uma exceção para o certificado não confiável.

### NOTE

A depuração de uma configuração de Compilação de versão com [Apenas Meu Código](#) e otimizações de compilador resulta em uma experiência inadequada. Por exemplo, os pontos de interrupção não são atingidos.

## Recursos adicionais

- [Hospedar o ASP.NET Core no Windows com o IIS](#)
- [Introdução ao Módulo do ASP.NET Core](#)
- [Referência de configuração do Módulo do ASP.NET Core](#)
- [Importar HTTPS](#)

# Módulos do IIS com o ASP.NET Core

21/01/2019 • 11 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

Alguns dos módulos nativos do IIS e todos os módulos gerenciados do IIS não são capazes de processar solicitações para aplicativos do ASP.NET Core. Em muitos casos, o ASP.NET Core oferece uma alternativa aos cenários abordados pelos módulos gerenciados e nativos do IIS.

## Módulos nativos

A tabela indica os módulos IIS nativos que funcionam com aplicativos ASP.NET Core e o Módulo do ASP.NET Core.

MÓDULO	FUNCIONAL COM OS APLICATIVOS DO ASP.NET CORE	OPÇÃO DO ASP.NET CORE
<b>Autenticação anônima</b> <code>AnonymousAuthenticationModule</code>	Sim	
<b>Autenticação básica</b> <code>BasicAuthenticationModule</code>	Sim	
<b>Autenticação de mapeamento de certificação de cliente</b> <code>CertificateMappingAuthenticationModule</code>	Sim	
<b>CGI</b> <code>CgiModule</code>	Não	
<b>Validação da configuração</b> <code>ConfigurationValidationModule</code>	Sim	
<b>Erros HTTP</b> <code>CustomErrorModule</code>	Não	Middleware de páginas de código de status
<b>Registro em log personalizado</b> <code>CustomLoggingModule</code>	Sim	
<b>Documento padrão</b> <code>DefaultDocumentModule</code>	Não	Middleware de arquivos padrão
<b>Autenticação Digest</b> <code>DigestAuthenticationModule</code>	Sim	
<b>Pesquisa no Diretório</b> <code>DirectoryListingModule</code>	Não	Middleware de navegação no diretório
<b>Compactação dinâmica</b> <code>DynamicCompressionModule</code>	Sim	Middleware de compactação de resposta

MÓDULO	FUNCIONAL COM OS APLICATIVOS DO ASP.NET CORE	OPÇÃO DO ASP.NET CORE
<b>Rastreamento</b> FailedRequestsTracingModule	Sim	Registro em log do ASP.NET Core
<b>Cache de arquivo</b> FileCacheModule	Não	Middleware de Cache de Resposta
<b>Cache HTTP</b> HttpCacheModule	Não	Middleware de Cache de Resposta
<b>Log HTTP</b> HttpLoggingModule	Sim	Registro em log do ASP.NET Core
<b>Redirecionamento de HTTP</b> HttpRedirectionModule	Sim	Middleware de regravação de URL
<b>Autenticação de mapeamento de certificado do cliente IIS</b> IISCertificateMappingAuthenticationModule	Sim	
<b>Restrições de IP e domínio</b> IpRestrictionModule	Sim	
<b>Filtros ISAPI</b> IsapiFilterModule	Sim	Middleware
<b>ISAPI</b> IsapiModule	Sim	Middleware
<b>Supporte de protocolo</b> ProtocolSupportModule	Sim	
<b>Filtragem de Solicitações</b> RequestFilteringModule	Sim	Middleware de regravação de URL IRule
<b>Monitor de Solicitações</b> RequestMonitorModule	Sim	
<b>Regravação de URL<sup>†</sup></b> RewriteModule	Sim	Middleware de regravação de URL
<b>Inclusões do lado do servidor</b> ServerSideIncludeModule	Não	
<b>Compactação estática</b> StaticCompressionModule	Não	Middleware de compactação de resposta
<b>Conteúdo Estático</b> StaticFileModule	Não	Middleware de arquivos estáticos
<b>Cache de token</b> TokenCacheModule	Sim	

MÓDULO	FUNCIONAL COM OS APLICATIVOS DO ASP.NET CORE	OPÇÃO DO ASP.NET CORE
<b>Cache de URI</b> UriCacheModule	Sim	
<b>Autorização de URL</b> UrlAuthorizationModule	Sim	Identidade do ASP.NET Core
<b>Autenticação do Windows</b> WindowsAuthenticationModule	Sim	

<sup>†</sup>Os tipos de correspondência `isFile` e `isDirectory` do módulo de regravação da URL não funcionam com aplicativos do ASP.NET Core, devido a alterações na [estrutura de diretórios](#).

## Módulos gerenciados

Os módulos gerenciados *não* funcionam com aplicativos do ASP.NET Core hospedados quando a versão do .NET CLR do pool de aplicativos está definido como **Sem Código Gerenciado**. O ASP.NET Core oferece alternativas de middleware em vários casos.

MÓDULO	OPÇÃO DO ASP.NET CORE
AnonymousIdentification	
DefaultAuthentication	
FileAuthorization	
FormsAuthentication	Middleware de autenticação de cookie
OutputCache	Middleware de Cache de Resposta
Perfil	
RoleManager	
ScriptModule-4.0	
Session	Middleware de sessão
UrlAuthorization	
UrlMappingsModule	Middleware de regravação de URL
UrlRoutingModule-4.0	Identidade do ASP.NET Core
WindowsAuthentication	

## Alterações de aplicativo do Gerenciador do IIS

Ao usar o Gerenciador do IIS para definir as configurações, o arquivo `web.config` do aplicativo é alterado. Ao implantar um aplicativo e incluir `web.config`, todas as alterações feitas com o Gerenciador do IIS são

substituídas pelo arquivo `web.config` implantado. Se forem feitas alterações para o arquivo `web.config` do servidor, copie o arquivo `web.config` atualizado no servidor para o projeto local imediatamente.

## Desabilitando módulos do IIS

Se um módulo do IIS é configurado no nível do servidor que deve ser desabilitado para um aplicativo, uma adição ao arquivo `web.config` do aplicativo pode desabilitar o módulo. Deixe o módulo no lugar e desative-o usando uma definição de configuração (se disponível) ou remova o módulo do aplicativo.

### Desativação do módulo

Muitos módulos oferecem uma configuração que permite que eles sejam desabilitados sem remover o módulo do aplicativo. Essa é a maneira mais simples e rápida de desativar um módulo. Por exemplo, o módulo de redirecionamento de HTTP pode ser desabilitado com o elemento `<httpRedirect>` em `web.config`:

```
<configuration>
  <system.webServer>
    <httpRedirect enabled="false" />
  </system.webServer>
</configuration>
```

Para obter mais informações sobre como desabilitar módulos com definições de configuração, siga os links na seção [Elementos Filho de IIS `<system.webServer>`](#).

### Remoção do módulo

Se optar pela remoção de um módulo com uma configuração em `web.config`, desbloqueie o módulo e desbloqueie a seção `<modules>` de `web.config` primeiro:

1. Desbloqueie o módulo no nível do servidor. Selecione o servidor do IIS na barra lateral **Conexões** do Gerenciador do IIS. Abra os **Módulos** na área **IIS**. Selecione o módulo na lista. Na barra lateral **Ações** à direita, selecione **Desbloquear**. Se a entrada de ação para o módulo aparece como **Bloquear**, o módulo já está desbloqueado e nenhuma ação é necessária. Desbloqueie todos os módulos que você planeja remover de `web.config` posteriormente.
2. Implantar o aplicativo sem uma seção `<modules>` em `web.config`. Se um aplicativo é implantado com um `web.config` que contém a seção `<modules>` sem ter desbloqueado a seção primeiro no Gerenciador do IIS, o Configuration Manager gera uma exceção ao tentar desbloquear a seção. Portanto, implante o aplicativo sem uma seção `<modules>`.
3. Desbloqueie a seção `<modules>` de `web.config`. Na barra lateral **Conexões**, selecione o site em **Sites**. Na área **Gerenciamento**, abra o **Editor de Configuração**. Use os controles de navegação para selecionar a seção `system.webServer/modules`. Na barra lateral **Ações** à direita, selecione para **Desbloquear** a seção. Se a entrada de ação para a seção do módulo aparece como **Bloquear Seção**, a seção do módulo já está desbloqueada e nenhuma ação é necessária.
4. Adicione uma seção `<modules>` ao arquivo `web.config` local do aplicativo com um elemento `<remove>` para remover o módulo do aplicativo. Adicione vários elementos `<remove>` para remover vários módulos. Se alterações a `web.config` forem feitas no servidor, faça imediatamente as mesmas alterações no arquivo `web.config` do projeto localmente. Remover um módulo usando essa abordagem não afeta o uso do módulo com outros aplicativos no servidor.

```
<configuration>
  <system.webServer>
    <modules>
      <remove name="MODULE_NAME" />
    </modules>
  </system.webServer>
</configuration>
```

Para adicionar ou remover módulos para IIS Express usando o *web.config*, modifique o *applicationHost.config* para desbloquear a seção `<modules>`:

1. Abra `{APPLICATION ROOT}\.vs\config\applicationhost.config`.
2. Localize o elemento `<section>` para módulos do IIS e a altere `overrideModeDefault` de `Deny` para `Allow`:

```
<section name="modules"
  allowDefinition="MachineToApplication"
  overrideModeDefault="Allow" />
```

3. Localize a seção `<location path="" overrideMode="Allow"><system.webServer><modules>`. Para todos os módulos que você deseja remover, defina `lockItem` de `true` para `false`. No exemplo a seguir, o módulo CGI é desbloqueado:

```
<add name="CgiModule" lockItem="false" />
```

4. Após a seção `<modules>` e módulos individuais serem desbloqueados, você pode adicionar ou remover módulos do IIS usando o arquivo *web.config* do aplicativo para executar o aplicativo no IIS Express.

Um módulo do IIS também pode ser removido com *Appcmd.exe*. Forneça o `MODULE_NAME` e `APPLICATION_NAME` no comando:

```
Appcmd.exe delete module MODULE_NAME /app.name:APPLICATION_NAME
```

Por exemplo, remova o `DynamicCompressionModule` do site da Web padrão:

```
%windir%\system32\inetsrv\appcmd.exe delete module DynamicCompressionModule /app.name:"Default Web Site"
```

## Configuração do módulo mínimo

Os únicos módulos necessários para executar um aplicativo ASP.NET Core são o módulo de Autenticação Anônima e o Módulo do ASP.NET Core.

O módulo de cache de URI (`uriCacheModule`) permite configuração de site, do IIS para o cache, no nível da URL. Sem esse módulo, o IIS deve ler e analisar a configuração em cada solicitação, mesmo quando a mesma URL é solicitada repetidamente. Analisar a configuração em cada solicitação resulta em uma perda de desempenho significativa. *Embora o módulo de cache de URI não seja estritamente necessário para que um aplicativo ASP.NET Core hospedado seja executado, é recomendável que o módulo de cache de URI seja habilitado para todas as implantações do ASP.NET Core.*

O módulo de cache HTTP (`HttpCacheModule`) implementa o cache de saída do IIS e também a lógica para gravar itens no cache do HTTP.sys. Sem esse módulo, o conteúdo não é mais armazenado em cache no modo kernel e perfis de cache são ignorados. Remover o módulo de cache HTTP geralmente tem um efeito adverso sobre o

desempenho e o uso de recursos. Embora o módulo de cache HTTP não seja estritamente necessário para que um aplicativo ASP.NET Core hospedado seja executado, é recomendável que o módulo de cache de HTTP seja habilitado para todas as implantações do ASP.NET Core.

## Recursos adicionais

- [Hospedar o ASP.NET Core no Windows com o IIS](#)
- [Introdução às arquiteturas do IIS: módulos no IIS](#)
- [Visão geral de módulos do IIS](#)
- [Personalizando funções e módulos do IIS 7.0](#)
- [IIS `<system.webServer>`](#)

# Solucionar problemas do ASP.NET Core no IIS

10/01/2019 • 19 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

Este artigo fornece instruções sobre como diagnosticar um problema de inicialização do aplicativo ASP.NET Core ao hospedar com [IIS \(Serviços de Informações da Internet\)](#). As informações neste artigo se aplicam à hospedagem em IIS no Windows Server e Windows Desktop.

No Visual Studio, um projeto do ASP.NET Core usa por padrão a hospedagem do [IIS Express](#) durante a depuração. Uma *502.5 – falha de processo* ou uma *500.30 – falha de inicialização* que ocorre ao depurar localmente pode ser solucionada usando as recomendações presentes neste tópico.

No Visual Studio, um projeto do ASP.NET Core usa por padrão a hospedagem do [IIS Express](#) durante a depuração. Uma *502.5 – Falha de Processo* que ocorre ao depurar localmente pode ser solucionada usando as recomendações presentes neste tópico.

Tópicos adicionais de solução de problemas:

## [Solucionar problemas no ASP.NET Core no Serviço de Aplicativo do Azure](#)

Embora o Serviço de Aplicativo use o [Módulo do ASP.NET Core](#) e o IIS para hospedar aplicativos, veja o tópico dedicado para obter instruções específicas para o Serviço de Aplicativo.

## [Tratar erros no ASP.NET Core](#)

Descubra como tratar erros em aplicativos do ASP.NET Core durante o desenvolvimento em um sistema local.

## [Aprenda a depurar usando o Visual Studio](#)

Este tópico apresenta os recursos do depurador do Visual Studio.

## [Depurar com o Visual Studio Code](#)

Saiba mais sobre o suporte de depuração interno do Visual Studio Code.

# Erros de inicialização do aplicativo

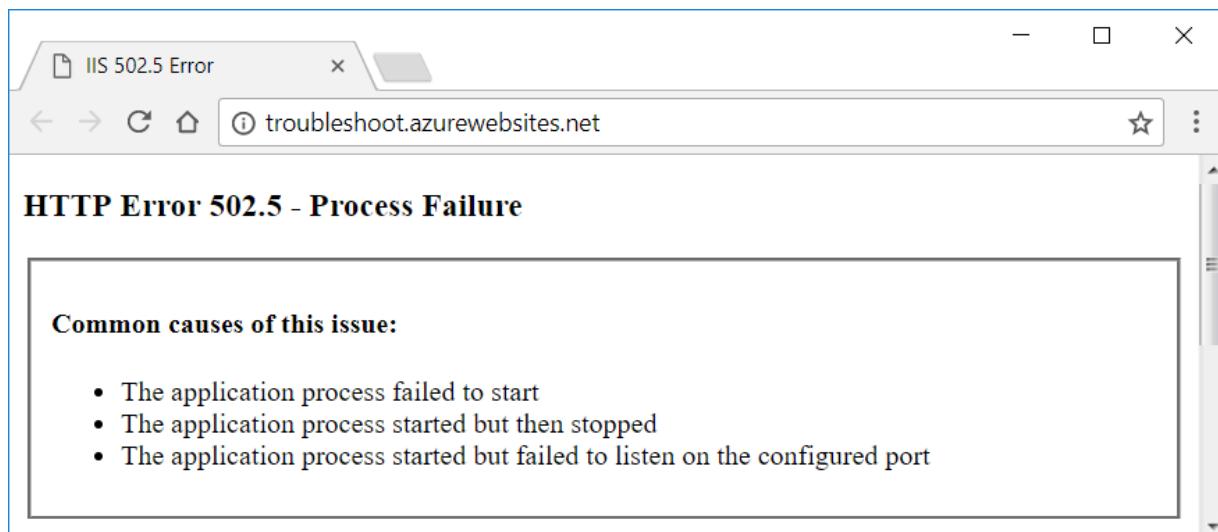
## **502.5 Falha de processo**

O processo de trabalho falha. O aplicativo não foi iniciado.

O Módulo do ASP.NET Core tenta iniciar o processo dotnet de back-end, mas falha ao iniciar. A causa de uma falha de inicialização do processo geralmente pode ser determinada com base em entradas no [Log de Eventos do Aplicativo](#) e no [log de stdout do Módulo do ASP.NET Core](#).

Uma condição de falha comum é o aplicativo configurado incorretamente, direcionado a uma versão da estrutura compartilhada do ASP.NET Core que não está presente. Verifique quais versões da estrutura compartilhada do ASP.NET Core estão instaladas no computador de destino.

A página do erro *502.5 – Falha no Processo* é retornada quando um erro de configuração de hospedagem ou do aplicativo faz com que o processo de trabalho falhe:



## 500.30 Falha de inicialização em processo

O processo de trabalho falha. O aplicativo não foi iniciado.

O Módulo do ASP.NET Core tenta iniciar o CLR do .NET Core em processo, mas falha ao iniciar. A causa de uma falha de inicialização do processo geralmente pode ser determinada com base em entradas no [Log de Eventos do Aplicativo](#) e no [log de stdout do Módulo do ASP.NET Core](#).

Uma condição de falha comum é o aplicativo configurado incorretamente, direcionado a uma versão da estrutura compartilhada do ASP.NET Core que não está presente. Verifique quais versões da estrutura compartilhada do ASP.NET Core estão instaladas no computador de destino.

## 500.0 Falha de carregamento de manipulador em processo

O processo de trabalho falha. O aplicativo não foi iniciado.

O Módulo do ASP.NET Core falha ao encontrar o CLR do .NET Core e o manipulador de solicitação em processo (`aspnetcorev2_inprocess.dll`). Verifique se:

- O aplicativo destina-se ao pacote NuGet [Microsoft.AspNetCore.Server.IIS](#) ou ao [metapacote Microsoft.AspNetCore.App](#).
- A versão da estrutura compartilhada do ASP.NET Core a que o aplicativo se destina está instalada no computador de destino.

## 500.0 Falha de carregamento de manipulador fora de processo

O processo de trabalho falha. O aplicativo não foi iniciado.

O Módulo do ASP.NET Core falha ao encontrar o manipulador de solicitações de hospedagem de fora do processo. Verifique se a `aspnetcorev2_outofprocess.dll` está presente em uma subpasta próxima a `aspnetcorev2.dll`.

## 500 Erro Interno do Servidor

O aplicativo é iniciado, mas um erro impede o servidor de atender à solicitação.

Esse erro ocorre no código do aplicativo durante a inicialização ou durante a criação de uma resposta. A resposta poderá não conter nenhum conteúdo, ou a resposta poderá ser exibida como um *500 – Erro Interno do Servidor* no navegador. O Log de Eventos do Aplicativo geralmente indica que o aplicativo iniciou normalmente. Da perspectiva do servidor, isso está correto. O aplicativo foi iniciado, mas não é capaz de gerar uma resposta válida. [Execute o aplicativo em um prompt de comando](#) no servidor ou [habilite o log de stdout do Módulo do ASP.NET Core](#) para solucionar o problema.

### Falha ao iniciar o aplicativo (ErrorCode '0x800700c1')

```
EventID: 1010
Source: IIS AspNetCore Module V2
Failed to start application '/LM/W3SVC/6/ROOT/', ErrorCode '0x800700c1'.
```

O aplicativo falhou ao ser iniciado porque o assembly do aplicativo (*.dll*) não pôde ser carregado.

Esse erro ocorre quando há uma incompatibilidade de número de bits entre o aplicativo publicado e o processo w3wp/iisexpress.

Confirme se a configuração de 32 bits do pool de aplicativos está correta:

1. Selecione o pool de aplicativos no **Pools de Aplicativos** do Gerenciador do IIS.
2. Selecione **Configurações Avançadas** em **Editar Pool de Aplicativos** no painel **Ações**.
3. Defina **Habilitar Aplicativos de 32 bits**:
  - Se estiver implantando um aplicativo de 32 bits (x86), defina o valor como **True**.
  - Se estiver implantando um aplicativo de 64 bits (x64), defina o valor como **False**.

### Redefinição de conexão

Se um erro ocorrer após os cabeçalhos serem enviados, será tarde demais para o servidor enviar um **500 – Erro Interno do Servidor** no caso de um erro ocorrer. Isso geralmente acontece quando ocorre um erro durante a serialização de objetos complexos para uma resposta. Esse tipo de erro é exibida como um erro de *redefinição de conexão* no cliente. O [Log de aplicativo](#) pode ajudar a solucionar esses tipos de erros.

## Limites de inicialização padrão

O Módulo do ASP.NET Core está configurado com um *startupTimeLimit* padrão de 120 segundos. Quando deixado no valor padrão, um aplicativo pode levar até dois minutos para iniciar antes que uma falha do processo seja registrada em log pelo módulo. Para obter informações sobre como configurar o módulo, veja [Atributos do elemento `aspNetCore`](#).

## Solucionar problemas de inicialização do aplicativo

### Habilitar o log de depuração do Módulo do ASP.NET Core

Adicione as seguintes configurações de manipulador ao arquivo *web.config* do aplicativo para habilitar os logs de depuração do Módulo do ASP.NET Core:

```
<aspNetCore ...>
  <handlerSettings>
    <handlerSetting name="debugLevel" value="file" />
    <handlerSetting name="debugFile" value="c:\temp\ancm.log" />
  </handlerSettings>
</aspNetCore>
```

Confirme se o caminho especificado para o log existe e se a identidade do pool de aplicativos tem permissões de gravação no local.

### Log de Eventos do Aplicativo

Acesse o Log de Eventos do Aplicativo:

1. Abra o menu Iniciar, procure **Visualizador de Eventos** e, em seguida, selecione o aplicativo **Visualizador de Eventos**.
2. No **Visualizador de Eventos**, abra o nó **Logs do Windows**.
3. Selecione **Aplicativo** para abrir o Log de Eventos do Aplicativo.
4. Procure erros associados ao aplicativo com falha. Os erros têm um valor *Módulo AspNetCore do IIS* ou

Módulo AspNetCore do IIS Express na coluna Origem.

## Execute o aplicativo em um prompt de comando

Muitos erros de inicialização não produzem informações úteis no Log de Eventos do Aplicativo. Você pode encontrar a causa de alguns erros ao executar o aplicativo em um prompt de comando no sistema de hospedagem.

### Implantação dependente de estrutura

Se o aplicativo é uma [implantação dependente de estrutura](#):

1. Em um prompt de comando, navegue até a pasta de implantação e execute o aplicativo, executando o assembly do aplicativo com `dotnet.exe`. No comando a seguir, substitua o nome do assembly do aplicativo por `<assembly_name>`: `dotnet .\<assembly_name>.dll`.
2. A saída do console do aplicativo, mostrando eventuais erros, é gravada na janela do console.
3. Se os erros ocorrerem ao fazer uma solicitação para o aplicativo, faça uma solicitação para o host e a porta em que o Kestrel escuta. Usando o host e a porta padrão, faça uma solicitação para `http://localhost:5000/`. Se o aplicativo responde normalmente no endereço do ponto de extremidade do Kestrel, a probabilidade de o problema estar relacionado à configuração de hospedagem é maior e, de estar relacionado ao aplicativo, menor.

### Implantação autocontida

Se o aplicativo é uma [implantação autossuficiente](#):

1. Em um prompt de comando, navegue até a pasta de implantação e execute o arquivo executável do aplicativo. No comando a seguir, substitua o nome do assembly do aplicativo por `<assembly_name>`: `<assembly_name>.exe`.
2. A saída do console do aplicativo, mostrando eventuais erros, é gravada na janela do console.
3. Se os erros ocorrerem ao fazer uma solicitação para o aplicativo, faça uma solicitação para o host e a porta em que o Kestrel escuta. Usando o host e a porta padrão, faça uma solicitação para `http://localhost:5000/`. Se o aplicativo responde normalmente no endereço do ponto de extremidade do Kestrel, a probabilidade de o problema estar relacionado à configuração de hospedagem é maior e, de estar relacionado ao aplicativo, menor.

## Log de stdout do Módulo do ASP.NET Core

Para habilitar e exibir logs de stdout:

1. Navegue até a pasta de implantação do site no sistema de hospedagem.
2. Se a pasta `logs` não estiver presente, crie-a. Para obter instruções sobre como habilitar o MSBuild para criar a pasta `logs` na implantação automaticamente, veja o tópico [Estrutura de diretórios](#).
3. Edite o arquivo `web.config`. Defina `stdoutLogEnabled` para `true` e altere o caminho `stdoutLogFile` para apontar para a pasta `logs` (por exemplo, `.\logs\stdout`). `stdout` no caminho é o prefixo do nome do arquivo de log. Uma extensão de arquivo, uma ID do processo e um carimbo de data/hora são adicionados automaticamente quando o log é criado. Usando `stdout` como o prefixo do nome do arquivo, um arquivo de log típico é nomeado `stdout_20180205184032_5412.log`.
4. Verifique se a identidade do pool de aplicativos tem permissões de gravação para a pasta `logs`.
5. Salve o arquivo `web.config` atualizado.
6. Faça uma solicitação ao aplicativo.
7. Navegue até a pasta `logs`. Localize e abra o log de stdout mais recente.
8. Estude o log em busca de erros.

### IMPORTANT

Desabilite o registro em log de stdout quando a solução de problemas for concluída.

1. Edite o arquivo `web.config`.
2. Defina `stdoutLogEnabled` para `false`.
3. Salve o arquivo.

#### WARNING

Falha ao desabilitar o log de `stdout` pode levar a falhas de aplicativo ou de servidor. Não há limites para o tamanho do arquivo de log ou para o número de arquivos de log criados.

Para registro em log de rotina em um aplicativo ASP.NET Core, use uma biblioteca de registro em log que limita o tamanho do arquivo de log e realiza a rotação de logs. Para obter mais informações, veja [provedores de log de terceiros](#).

## Habilitar a página de exceção do desenvolvedor

A variável de ambiente `ASPNETCORE_ENVIRONMENT` pode ser adicionada ao `web.config` para executar o aplicativo no ambiente de desenvolvimento. Desde que o ambiente não seja substituído na inicialização do aplicativo por `UseEnvironment` no compilador do host, definir a variável de ambiente permite que a [Página de Exceções do Desenvolvedor](#) apareça quando o aplicativo é executado.

```
<aspNetCore processPath="dotnet"
            arguments=".\\MyApp.dll"
            stdoutLogEnabled="false"
            stdoutLogFile=".\\logs\\stdout"
            hostingModel="InProcess">
  <environmentVariables>
    <environmentVariable name="ASPNETCORE_ENVIRONMENT" value="Development" />
  </environmentVariables>
</aspNetCore>
```

```
<aspNetCore processPath="dotnet"
            arguments=".\\MyApp.dll"
            stdoutLogEnabled="false"
            stdoutLogFile=".\\logs\\stdout">
  <environmentVariables>
    <environmentVariable name="ASPNETCORE_ENVIRONMENT" value="Development" />
  </environmentVariables>
</aspNetCore>
```

Configurar a variável de ambiente para `ASPNETCORE_ENVIRONMENT` só é recomendado para servidores de preparo e de teste que não estejam expostos à Internet. Remova a variável de ambiente do arquivo `web.config` após a solução de problemas. Para obter informações sobre como definir variáveis de ambiente no `web.config`, confira [Elemento filho environmentVariables de aspNetCore](#).

## Erros de inicialização comuns

Consulte [Referência de erros comuns para o Serviço de Aplicativo do Azure e o IIS com o ASP.NET Core](#). A maioria dos problemas comuns que impedem a inicialização do aplicativo é abordada no tópico de referência.

## Obter dados de um aplicativo

Se um aplicativo for capaz de responder às solicitações, obtenha as solicitações, conexão e dados adicionais do aplicativo que usar o middleware embutido de terminal. Para saber mais e obter um código de exemplo, consulte [Solucionar problemas de projetos do ASP.NET Core](#).

## Aplicativo lento ou travando

Quando um aplicativo responde lentamente ou trava em uma solicitação, obtenha e analise um [arquivo de despejo](#). Arquivos de despejo podem ser obtidos usando qualquer uma das ferramentas a seguir:

- [ProcDump](#)
- [DebugDiag](#)
- WinDbg: [Baixar as ferramentas de depuração para Windows, Depuração usando o WinDbg](#)

## Depuração remota

Veja [Depuração remota do ASP.NET Core em um computador de IIS remoto no Visual Studio 2017](#) na documentação do Visual Studio.

## Informações do aplicativo

O [Application Insights](#) fornece telemetria de aplicativos hospedados pelo IIS, incluindo recursos de relatório e de registro de erros em log. O Application Insights só pode relatar erros ocorridos depois que o aplicativo é iniciado quando os recursos de registro em log do aplicativo se tornam disponíveis. Para obter mais informações, veja [Application Insights para ASP.NET Core](#).

## Orientações adicionais

Algumas vezes um aplicativo em funcionamento falha imediatamente após atualizar o SDK do .NET Core nas versões de pacote ou de computador de desenvolvimento no aplicativo. Em alguns casos, pacotes incoerentes podem interromper um aplicativo ao executar atualizações principais. A maioria desses problemas pode ser corrigida seguindo estas instruções:

1. Exclua as pastas *bin* e *obj*.
2. Limpe os caches de pacote em `%UserProfile%\.nuget\packages` e `%LocalAppData%\Nuget\v3-cache`.
3. Restaure e recompile o projeto.
4. Confirme que a implantação anterior no servidor foi completamente excluída antes de reimplantar o aplicativo.

### TIP

Uma maneira prática de se limpar caches do pacote é executar `dotnet nuget locals all --clear` de um prompt de comando.

Também é possível limpar os caches de pacote usando a ferramenta [nuget.exe](#) e executando o comando `nuget locals all -clear`. [nuget.exe](#) não é uma instalação fornecida com o sistema operacional Windows Desktop e devem ser obtidos separadamente do [site do NuGet](#).

## Recursos adicionais

- [Solucionar problemas de projetos do ASP.NET Core](#)
- [Tratar erros no ASP.NET Core](#)
- [Referência de erros comuns para o Serviço de Aplicativo do Azure e o IIS com o ASP.NET Core](#)
- [Módulo do ASP.NET Core](#)
- [Solucionar problemas no ASP.NET Core no Serviço de Aplicativo do Azure](#)

# Referência de erros comuns para o Serviço de Aplicativo do Azure e o IIS com o ASP.NET Core

21/01/2019 • 27 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

Este tópico oferece conselhos de solução de problemas para erros comuns ao hospedar aplicativos ASP.NET Core no Serviço de Aplicativos do Azure e no IIS.

Colete as seguintes informações:

- Comportamento do navegador (código de status e mensagem de erro)
- Entradas do Log de Eventos do Aplicativo
  - Serviço de Aplicativo do Azure – Confira [Solucionar problemas no ASP.NET Core no Serviço de Aplicativo do Azure](#).
  - IIS
    1. Selecione **Iniciar** no menu **Windows**, digite *Visualizador de Eventos* e pressione **Enter**.
    2. Após o **Visualizador de Eventos** ser aberto, expanda **Logs do Windows > Aplicativo** na barra lateral.
- Entradas do log de depuração e stdout do Módulo do ASP.NET Core
  - Serviço de Aplicativo do Azure – Confira [Solucionar problemas no ASP.NET Core no Serviço de Aplicativo do Azure](#).
  - IIS – Siga as instruções nas seções [Criação de log e redirecionamento](#) e [Logs de diagnóstico avançados](#) do tópico Módulo do ASP.NET Core.

Compare as informações do erro para os erros comuns a seguir. Se uma correspondência for encontrada, siga o aviso de solução de problemas.

A lista de erros neste tópico não é exaustiva. Se você encontrar um erro não listado aqui, abra um novo problema usando o botão **Comentários sobre o Conteúdo** na parte inferior deste tópico com instruções detalhadas sobre como reproduzir o erro.

## IMPORTANT

### Versões prévias do ASP.NET Core com o Serviço de Aplicativo do Azure

Versões prévias do ASP.NET Core não são implantadas para o Serviço de Aplicativo do Azure por padrão. Para hospedar um aplicativo que usa uma versão prévia do ASP.NET Core, veja [Implantar versão prévia do ASP.NET Core para o Serviço de Aplicativo do Azure](#).

## O instalador não pode obter os Pacotes Redistribuíveis do VC++

- **Exceção do instalador:** 0x80072efd –OU– 0x80072f76 – erro não especificado
- **Exceção do log do instalador:** Erro 0x80072efd –OU– 0x80072f76: Falha ao executar o pacote EXE

+O log está localizado em

*C:\Users\{USER}\AppData\Local\Temp\dd\_DotNetCoreWinSvrHosting\_{TIMESTAMP}.log.*

Solução de problemas:

Se o sistema não tiver acesso à Internet durante a [instalação do pacote de hospedagem do .NET Core](#), essa exceção ocorrerá quando o instalador for impedido de obter os *Pacotes Redistribuíveis do Microsoft Visual C++ 2015*. Obtenha um instalador do [Centro de Download da Microsoft](#). Se o instalador falhar, o servidor poderá não receber o tempo de execução do .NET Core necessário para hospedar uma **FDD (implantação dependente de estrutura)**. Se estiver hospedando uma FDD, confirme se o tempo de execução está instalado em **Programas e Recursos** ou **Aplicativos e recursos**. Se um tempo de execução específico for necessário, baixe o tempo de execução dos [Arquivos de Download do .NET](#) e instale-o no sistema. Depois de instalar o tempo de execução, reinicie o sistema ou o IIS executando **net stop was /y** seguido por **net start w3svc** em um prompt de comando.

## O upgrade do sistema operacional removeu o Módulo do ASP.NET Core de 32 bits

**Log do Aplicativo:** A DLL do Módulo **C:\WINDOWS\system32\inetsrv\aspnetcore.dll** falhou ao ser carregada. Os dados são o erro.

Solução de problemas:

Arquivos que não são do sistema operacional no diretório **C:\Windows\SysWOW64\inetsrv** não são preservados durante um upgrade do sistema operacional. Se o Módulo do ASP.NET Core estiver instalado antes de uma atualização do sistema operacional e, em seguida, qualquer pool de aplicativos for executado no modo de 32 bits após uma atualização do sistema operacional, esse problema será encontrado. Após um upgrade do sistema operacional, repare o Módulo do ASP.NET Core. Veja [Instalar o pacote de Hospedagem do .NET Core](#). Selecione **Reparar** ao executar o instalador.

## Um aplicativo x86 é implantado, mas o pool de aplicativos não está habilitado para aplicativos de 32 bits

- **Navegador:** Erro HTTP 500.30 – Falha de início no processo do ANCM
- **Log do Aplicativo:** Aplicativo '/LM/W3SVC/5/ROOT' com raiz física '{PATH}' atingiu uma exceção gerenciada inesperada, código de exceção = '0xe0434352'. Verifique os logs de stderr para obter mais informações. Aplicativo '/LM/W3SVC/5/ROOT' com raiz física '{PATH}' falhou ao carregar o clr e o aplicativo gerenciado. O thread de trabalho do CLR foi encerrado prematuramente
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log é criado, mas vazio.
- **Log de depuração do módulo do ASP.NET Core:** HRESULT com falha retornou: 0x8007023e

Esse cenário é interceptado pelo SDK ao publicar um aplicativo autocontido. O SDK produzirá um erro se o RID não coincidir com o destino da plataforma (por exemplo, RID `win10-x64` com `<PlatformTarget>x86</PlatformTarget>` no arquivo de projeto).

Solução de problemas:

Para uma implantação dependente da estrutura x86 (`<PlatformTarget>x86</PlatformTarget>`), habilite o pool de aplicativos de IIS para aplicativos de 32 bits. No Gerenciador do IIS, abra as **Configurações Avançadas** do pool de aplicativos e defina **Habilitar Aplicativos de 32 Bits** como **Verdadeiro**.

## Conflitos de plataforma com o RID

- **Navegador:** Erro HTTP 502.5 – falha do processo
- **Log do Aplicativo:** O aplicativo 'MACHINE/WEBROOT/APPHOST/{ASSEMBLY}' com raiz física 'C:{PATH}' falhou ao iniciar o processo com a linha de comando '"C:{PATH}{ASSEMBLY}.{exe|dll}"', ErrorCode = '0x80004005 : ff.'

- **Log de stdout do Módulo do ASP.NET Core:** Exceção sem tratamento:  
System.BadImageFormatException: Não foi possível carregar arquivo ou o assembly '{ASSEMBLY}.dll'.  
Foi feita uma tentativa de carregar um programa com um formato incorreto.

Solução de problemas:

- Confirme se o aplicativo é executado localmente no Kestrel. Uma falha do processo pode ser o resultado de um problema no aplicativo. Para obter mais informações, confira [Solução de problemas \(IIS\)](#) ou [Solução de problemas \(Serviço de Aplicativo do Azure\)](#).
- Se essa exceção ocorrer para uma implantação dos Aplicativos do Azure ao fazer upgrade de um aplicativo e implantar assemblies mais recentes, exclua manualmente todos os arquivos da implantação anterior. Assemblies incompatíveis remanescentes podem resultar em uma exceção `System.BadImageFormatException` durante a implantação de um aplicativo atualizado.

## Ponto de extremidade de URI incorreto ou site interrompido

- **Navegador:** ERR\_CONNECTION\_REFUSED –OU– Não é possível estabelecer conexão
- **Log do Aplicativo:** Nenhuma entrada
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log não é criado.
- **Log de depuração do módulo do ASP.NET Core:** O arquivo de log não é criado.

Solução de problemas:

- Confirme se o ponto de extremidade do URI correto para o aplicativo está sendo usado. Verifique as associações.
- Confirme que o site do IIS não está no estado *Parado*.

## Recursos do servidor CoreWebEngine ou W3SVC desabilitados

**Exceção do Sistema Operacional:** Os recursos CoreWebEngine e W3SVC do IIS 7.0 devem ser instalados para usar o Módulo do ASP.NET Core.

Solução de problemas:

Confirme que a função e os recursos apropriados estão habilitados. Consulte [Configuração do IIS](#).

## Caminho físico do site incorreto ou aplicativo ausente

- **Navegador:** 403 Proibido – acesso negado –OU– 403.14 Proibido – o servidor Web está configurado para não listar o conteúdo deste diretório.
- **Log do Aplicativo:** Nenhuma entrada
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log não é criado.
- **Log de depuração do módulo do ASP.NET Core:** O arquivo de log não é criado.

Solução de problemas:

Confira as **Configurações Básicas** no site do IIS e a pasta do aplicativo físico. Confirme que o aplicativo está na pasta no **Caminho físico** do site do IIS.

## Função incorreta, Módulo do ASP.NET Core Não Instalado ou

## permissões incorretas

- **Navegador:** 500.19 Erro interno do servidor – a página solicitada não pode ser acessada porque os dados de configuração relacionados da página são inválidos. **-OU-** Esta página não pode ser exibida
- **Log do Aplicativo:** Nenhuma entrada
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log não é criado.
- **Log de depuração do módulo do ASP.NET Core:** O arquivo de log não é criado.

Solução de problemas:

- Confirme que você habilitou a função apropriada. Consulte [Configuração do IIS](#).
- Abra **Programas e Recursos** ou **Aplicativos e Recursos** e confirme se a **Hospedagem do Windows Server** está instalada. Se a **Hospedagem do Windows Server** não estiver presente na lista de programas instalados, baixe e instale o Pacote de Hospedagem do .NET Core.

[Instalador de pacote de hospedagem do .NET Core atual \(download direto\)](#)

Para obter mais informações, confira [Instalar o pacote de hospedagem do .NET Core](#).

- Verifique se o **Pool de aplicativos > Modelo de processo > Identidade** está definido como **ApplicationPoolIdentity** ou se a identidade personalizada tem as permissões corretas para acessar a pasta de implantação do aplicativo.

processPath incorreto, variável de PATH ausente, pacote de hospedagem não instalado, sistema/IIS não reiniciado, Pacotes Redistribuíveis do VC++ não instalados ou violação de acesso de dotnet.exe

- **Navegador:** Erro HTTP 500.0 – Falha de carregamento de manipulador em processo ANCM
- **Log do Aplicativo:** Aplicativo 'MACHINE/WEBROOT/APPHOST/{ASSEMBLY}' com raiz física 'C: {PATH}' falhou ao iniciar o processo com a linha de comando '"{}"', ErrorCode = '0x80070002 : 0. Não foi possível iniciar o aplicativo '{PATH}'. O executável não foi encontrado em '{PATH}'. Falha ao iniciar o aplicativo '/LM/W3SVC/2/ROOT', ErrorCode '0x8007023e'.
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log não é criado.
- **Log de depuração do módulo do ASP.NET Core:** Log de eventos: Não foi possível iniciar o aplicativo '{PATH}'. O executável não foi encontrado em '{PATH}'. HRESULT com falha retornou: 0x8007023e
- **Navegador:** Erro HTTP 502.5 – falha do processo
- **Log do Aplicativo:** Aplicativo 'MACHINE/WEBROOT/APPHOST/{ASSEMBLY}' com raiz física 'C: {PATH}' falhou ao iniciar o processo com a linha de comando '"{}"', ErrorCode = '0x80070002 : 0.
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log é criado, mas vazio.

Solução de problemas:

- Confirme se o aplicativo é executado localmente no Kestrel. Uma falha do processo pode ser o resultado de um problema no aplicativo. Para obter mais informações, confira [Solução de problemas \(IIS\)](#) ou [Solução de problemas \(Serviço de Aplicativo do Azure\)](#).
- Verifique o atributo *processPath* no elemento `<aspNetCore>` em *web.config* para confirmar se ele é `dotnet` para uma FDD (implantação dependente de estrutura) ou `.\{ASSEMBLY}.exe` para uma **SCD**

(implantação autossuficiente).

- Para uma FDD, o `dotnet.exe` pode não estar acessível por meio das configurações de PATH. Confirme se `*C:\Program Files\dotnet*` existe nas configurações de PATH do Sistema.
- Para uma FDD, o `dotnet.exe` pode não estar acessível para a identidade do usuário do pool de aplicativos. Confirme se a identidade do usuário do pool de aplicativos tem acesso ao diretório `C:\Arquivos de Programas\dotnet`. Confirme se não há nenhuma regra de negação configurada para a identidade do usuário do pool de aplicativos no `C:\Arquivos de Programas\dotnet` e nos diretórios do aplicativo.
- Talvez você tenha implantado uma FDD e instalado o .NET Core sem reiniciar o IIS. Reinicie o servidor ou o IIS executando `net stop was /y` seguido por `net start w3svc` em um prompt de comando.
- Você pode ter implantado uma FDD sem instalar o tempo de execução do .NET Core no sistema de hospedagem. Se o tempo de execução do .NET Core ainda não foi instalado, execute o **Instalador do Pacote de Hospedagem do .NET Core** no sistema.

[Instalador de pacote de hospedagem do .NET Core atual \(download direto\)](#)

Para obter mais informações, confira [Instalar o pacote de hospedagem do .NET Core](#).

Se um tempo de execução específico for necessário, baixe o tempo de execução dos [Arquivos de Download do .NET](#) e instale-o no sistema. Conclua a instalação reiniciando o sistema ou o IIS executando `net stop was /y` seguido por `net start w3svc` em um prompt de comando.

- Talvez você tenha implantado uma FDD e os *Pacotes redistribuíveis do Microsoft Visual C++ 2015 (x64)* não estejam instalados no sistema. Obtenha um instalador do [Centro de Download da Microsoft](#).

## Argumentos incorretos do elemento <aspNetCore>

- **Navegador:** Erro HTTP 500.0 – Falha de carregamento de manipulador em processo ANCM
- **Log do Aplicativo:** A invocação do hostfxr para encontrar o manipulador de solicitação inprocess falha sem encontrar nenhuma dependência nativa. Isso provavelmente significa que o aplicativo está configurado incorretamente, verifique as versões do `Microsoft.AspNetCore.App` e `Microsoft.AspNetCore.App` que são afetadas pelo aplicativo e estão instaladas no computador. Não foi possível localizar o manipulador de solicitação inprocess. Saída capturada da invocação do hostfxr: Você quis dizer executar comandos do SDK do dotnet? Instale o SDK do dotnet de:  
<https://go.microsoft.com/fwlink/?LinkID=798306&clcid=0x409> Falha ao iniciar o aplicativo '/LM/W3SVC/3/ROOT', ErrorCode '0x8000ffff'.
- **Log de stdout do Módulo do ASP.NET Core:** Você quis dizer executar comandos do SDK do dotnet? Instale o SDK do dotnet de: <https://go.microsoft.com/fwlink/?LinkID=798306&clcid=0x409>
- **Log de depuração do módulo do ASP.NET Core:** A invocação do hostfxr para encontrar o manipulador de solicitação inprocess falha sem encontrar nenhuma dependência nativa. Isso provavelmente significa que o aplicativo está configurado incorretamente, verifique as versões do `Microsoft.AspNetCore.App` e `Microsoft.AspNetCore.App` que são afetadas pelo aplicativo e estão instaladas no computador. HRESULT com falha retornou: 0x8000ffff Não foi possível localizar o manipulador de solicitação inprocess. Saída capturada da invocação do hostfxr: Você quis dizer executar comandos do SDK do dotnet? Instale o SDK do dotnet de: <https://go.microsoft.com/fwlink/?LinkID=798306&clcid=0x409> HRESULT com falha retornou: 0x8000ffff
- **Navegador:** Erro HTTP 502.5 – falha do processo
- **Log do Aplicativo:** Aplicativo 'MACHINE/WEBROOT/APPHOST/{ASSEMBLY}' com a raiz física 'C:{PATH}' falha ao iniciar o processo com a linha de comando '"dotnet" .{ASSEMBLY}.dll', ErrorCode = '0x80004005: 80008081.'

- **Log de stdout do Módulo do ASP.NET Core:** O aplicativo a ser executado não existe: 'PATH{ASSEMBLY}.dll'

Solução de problemas:

- Confirme se o aplicativo é executado localmente no Kestrel. Uma falha do processo pode ser o resultado de um problema no aplicativo. Para obter mais informações, confira [Solução de problemas \(IIS\)](#) ou [Solução de problemas \(Serviço de Aplicativo do Azure\)](#).
- Examine o atributo *arguments* no elemento `<aspNetCore>` no *web.config* para confirmar se ele: (a) é `. \{ASSEMBLY\}.dll` de uma FDD (implantação dependente de estrutura); ou (b) não está presente, é uma cadeia de caracteres vazia (`arguments=""`) ou uma lista de argumentos do aplicativo (`arguments="{ARGUMENT_1}, {ARGUMENT_2}, ... {ARGUMENT_X}"`) para uma SCD (implantação autossuficiente).

## Estrutura compartilhada do .NET Core ausente

- **Navegador:** Erro HTTP 500.0 – Falha de carregamento de manipulador em processo ANCM
- **Log do Aplicativo:** A invocação do hostfxr para encontrar o manipulador de solicitação inprocess falha sem encontrar nenhuma dependência nativa. Isso provavelmente significa que o aplicativo está configurado incorretamente, verifique as versões do Microsoft.AspNetCore.App e Microsoft.AspNetCore.App que são afetadas pelo aplicativo e estão instaladas no computador. Não foi possível localizar o manipulador de solicitação inprocess. Saída capturada da invocação do hostfxr: Não foi possível encontrar nenhuma versão de estrutura compatível. A estrutura especificada 'Microsoft.AspNetCore.App', versão '{VERSION}', não foi encontrada.

Falha ao iniciar o aplicativo '/LM/W3SVC/5/ROOT', ErrorCode '0x8000ffff'.

- **Log de stdout do Módulo do ASP.NET Core:** Não foi possível encontrar nenhuma versão de estrutura compatível. A estrutura especificada 'Microsoft.AspNetCore.App', versão '{VERSION}', não foi encontrada.
- **Log de depuração do módulo do ASP.NET Core:** HRESULT com falha retornou: 0x8000ffff

Solução de problemas:

Para uma FDD (implantação dependente de estrutura), confirme se você tem o tempo de execução correto instalado no sistema.

## Pool de aplicativos interrompido

- **Navegador:** 503 Serviço Não Disponível
- **Log do Aplicativo:** Nenhuma entrada
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log não é criado.
- **Log de depuração do módulo do ASP.NET Core:** O arquivo de log não é criado.

Solução de problemas:

Confirme que o Pool de Aplicativos não está no estado *Parado*.

## O subaplicativo inclui uma seção <manipuladores>

- **Navegador:** Erro HTTP 500.19 – Erro Interno do Servidor
- **Log do Aplicativo:** Nenhuma entrada

- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log do aplicativo raiz é criado e mostra uma operação normal. O arquivo de log do subaplicativo não é criado.
- **Log de depuração do módulo do ASP.NET Core:** O arquivo de log do aplicativo raiz é criado e mostra uma operação normal. O arquivo de log do subaplicativo não é criado.

Solução de problemas:

Confirme se o arquivo `web.config` do subaplicativo não inclui uma seção `<handlers>` ou que o subaplicativo não herda os manipuladores do aplicativo pai.

A seção `<system.webServer>` do aplicativo pai de `web.config` é colocada dentro de um elemento `<location>`. A propriedade `InheritInChildApplications` é definida como `false` para indicar que as configurações especificadas no elemento `<location>` não são herdadas por aplicativos que residem em um subdiretório do aplicativo pai. Para obter mais informações, consulte [Módulo do ASP.NET Core](#).

Confirme se o arquivo `web.config` do subaplicativo não inclui uma seção `<handlers>`.

## caminho do log de stdout incorreto

- **Navegador:** O aplicativo responde normalmente.
- **Log do Aplicativo:** Não foi possível iniciar o redirecionamento de stdout em C:\Arquivos de Programas\IIS\Asp.Net Core Module\V2\aspnetcorev2.dll. Mensagem de exceção: HRESULT 0x80070005 retornado em {PATH}\aspnetcoremodulev2\commonlib\fileoutputmanager.cpp:84. Não foi possível parar o redirecionamento de stdout em C:\Arquivos de Programas\IIS\Asp.Net Core Module\V2\aspnetcorev2.dll. Mensagem de exceção: HRESULT 0x80070002 retornado em {PATH}. Não foi possível iniciar o redirecionamento de stdout em {PATH}\aspnetcorev2\_inprocess.dll.
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log não é criado.
- **Log de depuração do módulo do ASP.NET Core:** Não foi possível iniciar o redirecionamento de stdout em C:\Arquivos de Programas\IIS\Asp.Net Core Module\V2\aspnetcorev2.dll. Mensagem de exceção: HRESULT 0x80070005 retornado em {PATH}\aspnetcoremodulev2\commonlib\fileoutputmanager.cpp:84. Não foi possível parar o redirecionamento de stdout em C:\Arquivos de Programas\IIS\Asp.Net Core Module\V2\aspnetcorev2.dll. Mensagem de exceção: HRESULT 0x80070002 retornado em {PATH}. Não foi possível iniciar o redirecionamento de stdout em {PATH}\aspnetcorev2\_inprocess.dll.
- **Log do Aplicativo:** Aviso: Não foi possível criar `stdoutLogFile \?`  
`{PATH}\path_doesnt_exist\stdout_{PROCESS ID}{TIMESTAMP}.log`, `ErrorCode = -2147024893`.
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log não é criado.

Solução de problemas:

- O caminho `stdoutLogFile` especificado no elemento `<aspNetCore>` de `web.config` não existe. Para obter mais informações, confira [Módulo ASP.NET Core: criação de log e redirecionamento](#).
- O usuário do pool de aplicativos não tem acesso de gravação para o caminho do log de stdout.

## Problema geral de configuração do aplicativo

- **Navegador:** Erro HTTP 500.0 – Falha de carregamento de manipulador em processo ANCM –**OU–** Erro HTTP 500.30 – Falha de início no processo do ANCM
- **Log do Aplicativo:** Variável

- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log é criado, mas vazio, ou criado com entradas normais até o ponto da falha do aplicativo.
- **Log de depuração do módulo do ASP.NET Core:** Variável
- **Navegador:** Erro HTTP 502.5 – falha do processo
- **Log do Aplicativo:** Aplicativo 'MACHINE/WEBROOT/APPHOST/{ASSEMBLY}' com raiz física 'C:{PATH}' criada no processo com a linha de comando '"C:{PATH}{ASSEMBLY}.{exe|dll}"', mas falhou, não respondeu ou não escutou na porta '{PORT}' fornecida, ErrorCode = '{ERROR CODE}'
- **Log de stdout do Módulo do ASP.NET Core:** O arquivo de log é criado, mas vazio.

Solução de problemas:

O processo não pôde ser iniciado, provavelmente, devido a um problema de programação ou configuração do aplicativo.

Para mais informações, consulte os seguintes tópicos:

- [Solucionar problemas do ASP.NET Core no IIS](#)
- [Solucionar problemas no ASP.NET Core no Serviço de Aplicativo do Azure](#)
- [Solucionar problemas de projetos do ASP.NET Core](#)

# Implementação do servidor Web Kestrel no ASP.NET Core

17/01/2019 • 42 minutes to read • [Edit Online](#)

Por [Tom Dykstra](#), [Chris Ross](#) e [Stephen Halter](#)

Para obter a versão 1.1 deste tópico, baixe [Implementação do servidor Web Kestrel no ASP.NET Core \(versão 1.1, PDF\)](#).

O Kestrel é um [servidor Web multiplataforma para o ASP.NET Core](#). O Kestrel é o servidor Web que está incluído por padrão em modelos de projeto do ASP.NET Core.

O Kestrel dá suporte aos seguintes cenários:

- HTTPS
- Atualização do Opaque usado para habilitar o [WebSockets](#)
- Soquetes do UNIX para alto desempenho protegidos pelo Nginx
- HTTP/2 (exceto em macOS<sup>†</sup>)

<sup>†</sup>O HTTP/2 será compatível com macOS em uma versão futura.

- HTTPS
- Atualização do Opaque usado para habilitar o [WebSockets](#)
- Soquetes do UNIX para alto desempenho protegidos pelo Nginx

Há suporte para o Kestrel em todas as plataformas e versões compatíveis com o .NET Core.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Compatibilidade com HTTP/2

O [HTTP/2](#) estará disponível para aplicativos ASP.NET Core se os seguintes requisitos básicos forem atendidos:

- Sistema operacional<sup>†</sup>
  - Windows Server 2016/Windows 10 ou posterior<sup>#</sup>
  - Linux com OpenSSL 1.0.2 ou posterior (por exemplo, Ubuntu 16.04 ou posterior)
- Estrutura de destino: .NET Core 2.2 ou posterior
- Conexão [ALPN \(Negociação de protocolo de camada de aplicativo\)](#)
- Conexão TLS 1.2 ou posterior

<sup>†</sup>O HTTP/2 será compatível com macOS em uma versão futura. <sup>#</sup>O Kestrel tem suporte limitado para HTTP/2 no Windows Server 2012 R2 e Windows 8.1. O suporte é limitado porque a lista de conjuntos de codificação TLS disponível nesses sistemas operacionais é limitada. Um certificado gerado usando um ECDSA (Algoritmo de Assinatura Digital Curva Elíptica) pode ser necessário para proteger conexões TLS.

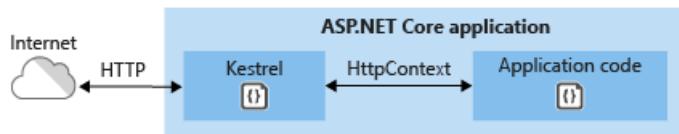
Se uma conexão HTTP/2 for estabelecida, `HttpRequest.Protocol` relatará `HTTP/2`.

HTTP/2 está desabilitado por padrão. Para obter mais informações sobre a configuração, consulte as seções [Opções do Kestrel](#) e [ListenOptions.Protocols](#).

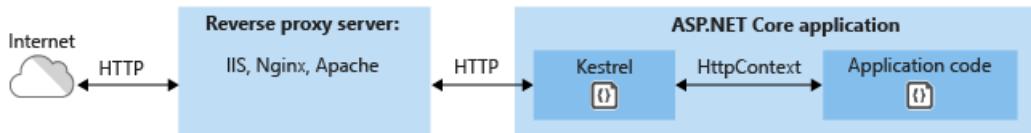
## Quando usar o Kestrel com um proxy reverso

Você pode usar o Kestrel sozinho ou com um *servidor proxy reverso* como [IIS \(Serviços de Informações da Internet\)](#), [Nginx](#) ou [Apache](#). Um servidor proxy reverso recebe solicitações HTTP da rede e encaminha-as para o Kestrel.

Kestrel usado como um servidor Web de borda (voltado para a Internet):



Kestrel usado em uma configuração de proxy reverso:



Qualquer configuração – com ou sem um servidor proxy reverso – é uma configuração de hospedagem compatível com o ASP.NET Core 2.1 ou aplicativos posteriores que recebem solicitações da Internet.

O Kestrel usado como um servidor de borda sem um servidor proxy reverso não dá suporte ao compartilhamento do mesmo IP e da mesma porta entre vários processos. Quando o Kestrel é configurado para escutar uma porta, ele manipula todo o tráfego dessa porta, independentemente dos cabeçalhos `Host` das solicitações. Um proxy reverso que pode compartilhar portas tem a capacidade de encaminhar solicitações ao Kestrel em um IP e em uma porta exclusivos.

Mesmo se um servidor proxy reverso não for necessário, o uso de um servidor proxy reverso poderá ser uma boa opção.

Um proxy reverso:

- Pode limitar a área da superfície pública exposta dos aplicativos que ele hospeda.
- Fornece uma camada adicional de configuração e proteção.
- Pode ser integrado melhor à infraestrutura existente.
- Simplifica o balanceamento de carga e a configuração de comunicação segura (HTTPS). Somente o servidor proxy reverso exige um certificado X.509 e esse servidor pode se comunicar com os servidores de aplicativos na rede interna usando HTTP simples.

### WARNING

A hospedagem em uma configuração de proxy reverso exige a [filtragem de host](#).

## Como usar o Kestrel em aplicativos ASP.NET Core

O pacote [Microsoft.AspNetCore.Server.Kestrel](#) está incluído no [metapacote Microsoft.AspNetCore.App](#) (ASP.NET Core 2.1 ou posterior).

Os modelos de projeto do ASP.NET Core usam o Kestrel por padrão. Em `Program.cs`, o código do modelo chama [CreateDefaultBuilder](#), que chama [UseKestrel](#) em segundo plano.

```
public static void Main(string[] args)
{
    CreateWebHostBuilder(args).Build().Run();
}

public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>();
```

Para fornecer configuração adicional após chamar `CreateDefaultBuilder`, use `ConfigureKestrel`:

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .ConfigureKestrel((context, options) =>
    {
        // Set properties and call methods on options
});
```

Se o aplicativo não chamar `CreateDefaultBuilder` para configurar o host, chame `UseKestrel` **antes** de chamar `ConfigureKestrel`:

```
public static void Main(string[] args)
{
    var host = new WebHostBuilder()
        .UseContentRoot(Directory.GetCurrentDirectory())
        .UseKestrel()
        .UseIISIntegration()
        .UseStartup<Startup>()
        .ConfigureKestrel((context, options) =>
    {
        // Set properties and call methods on options
    })
        .Build();

    host.Run();
}
```

Para fornecer configuração adicional após chamar `CreateDefaultBuilder`, chame `UseKestrel`:

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .UseKestrel(options =>
    {
        // Set properties and call methods on options
});
```

## Opções do Kestrel

O servidor Web do Kestrel tem opções de configuração de restrição especialmente úteis em implantações para a Internet.

Defina restrições na propriedade `Limits` da classe `KestrelServerOptions`. A propriedade `Limits` contém uma instância da classe `KestrelServerLimits`.

### Número máximo de conexões de cliente

## MaxConcurrentConnections

## MaxConcurrentUpgradedConnections

O número máximo de conexões TCP abertas simultâneas pode ser definido para o aplicativo inteiro com o código a seguir:

```
.ConfigureKestrel((context, options) =>
{
    options.Limits.MaxConcurrentConnections = 100;
    options.Limits.MaxConcurrentUpgradedConnections = 100;
    options.Limits.MaxRequestBodySize = 10 * 1024;
    options.Limits.MinRequestBodyDataRate =
        new MinDataRate(bytesPerSecond: 100, gracePeriod: TimeSpan.FromSeconds(10));
    options.Limits.MinResponseDataRate =
        new MinDataRate(bytesPerSecond: 100, gracePeriod: TimeSpan.FromSeconds(10));
    options.Listen(IPAddress.Loopback, 5000);
    options.Listen(IPAddress.Loopback, 5001, listenOptions =>
    {
        listenOptions.UseHttps("testCert.pfx", "testPassword");
    });
});
```

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .UseKestrel(options =>
    {
        options.Limits.MaxConcurrentConnections = 100;
    });
};
```

Há um limite separado para conexões que foram atualizadas do HTTP ou HTTPS para outro protocolo (por exemplo, em uma solicitação do WebSockets). Depois que uma conexão é atualizada, ela não é contada em relação ao limite de `MaxConcurrentConnections`.

```
.ConfigureKestrel((context, options) =>
{
    options.Limits.MaxConcurrentConnections = 100;
    options.Limits.MaxConcurrentUpgradedConnections = 100;
    options.Limits.MaxRequestBodySize = 10 * 1024;
    options.Limits.MinRequestBodyDataRate =
        new MinDataRate(bytesPerSecond: 100, gracePeriod: TimeSpan.FromSeconds(10));
    options.Limits.MinResponseDataRate =
        new MinDataRate(bytesPerSecond: 100, gracePeriod: TimeSpan.FromSeconds(10));
    options.Listen(IPAddress.Loopback, 5000);
    options.Listen(IPAddress.Loopback, 5001, listenOptions =>
    {
        listenOptions.UseHttps("testCert.pfx", "testPassword");
    });
});
```

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .UseKestrel(options =>
    {
        options.Limits.MaxConcurrentUpgradedConnections = 100;
    });
};
```

O número máximo de conexões é ilimitado (nulo) por padrão.

## Tamanho máximo do corpo da solicitação

### MaxRequestBodySize

O tamanho máximo do corpo da solicitação padrão é de 30.000.000 bytes, que equivale aproximadamente a 28,6 MB.

A abordagem recomendada para substituir o limite em um aplicativo ASP.NET Core MVC é usar o atributo [RequestSizeLimit](#) em um método de ação:

```
[RequestSizeLimit(100000000)]
public IActionResult MyActionMethod()
```

Aqui está um exemplo que mostra como configurar a restrição para o aplicativo em cada solicitação:

```
.ConfigureKestrel((context, options) =>
{
    options.Limits.MaxConcurrentConnections = 100;
    options.Limits.MaxConcurrentUpgradedConnections = 100;
    options.Limits.MaxRequestBodySize = 10 * 1024;
    options.Limits.MinRequestBodyDataRate =
        new MinDataRate(bytesPerSecond: 100, gracePeriod: TimeSpan.FromSeconds(10));
    options.Limits.MinResponseDataRate =
        new MinDataRate(bytesPerSecond: 100, gracePeriod: TimeSpan.FromSeconds(10));
    options.Listen(IPAddress.Loopback, 5000);
    options.Listen(IPAddress.Loopback, 5001, listenOptions =>
    {
        listenOptions.UseHttps("testCert.pfx", "testPassword");
    });
});
```

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .UseKestrel(options =>
    {
        options.Limits.MaxRequestBodySize = 10 * 1024;
    });
}
```

Substitua a configuração em uma solicitação específica no middleware:

```

app.Run(async (context) =>
{
    context.Features.Get<IHttpMaxRequestBodySizeFeature>()
        .MaxRequestBodySize = 10 * 1024;

    var minRequestRateFeature =
        context.Features.Get<IHttpMinRequestBodyDataRateFeature>();
    var minResponseRateFeature =
        context.Features.Get<IHttpMinResponseDataRateFeature>();

    if (minRequestRateFeature != null)
    {
        minRequestRateFeature.MinDataRate = new MinDataRate(
            bytesPerSecond: 100, gracePeriod: TimeSpan.FromSeconds(10));
    }

    if (minResponseRateFeature != null)
    {
        minResponseRateFeature.MinDataRate = new MinDataRate(
            bytesPerSecond: 100, gracePeriod: TimeSpan.FromSeconds(10));
    }
}

```

Uma exceção será gerada se você tentar configurar o limite em uma solicitação depois que o aplicativo começar a ler a solicitação. Há uma propriedade `IsReadOnly` que indica se a propriedade `MaxRequestBodySize` está no estado somente leitura, o que significa que é tarde demais para configurar o limite.

### Taxa de dados mínima do corpo da solicitação

`MinRequestBodyDataRate`

`MinResponseDataRate`

O Kestrel verifica a cada segundo se os dados estão sendo recebidos na taxa especificada em bytes/segundo. Se a taxa cair abaixo do mínimo, a conexão atingirá o tempo limite. O período de cortesia é o tempo que o Kestrel fornece ao cliente para aumentar sua taxa de envio até o mínimo; a taxa não é verificada durante esse período. O período de cortesia ajuda a evitar a remoção de conexões que inicialmente enviavam dados em uma taxa baixa devido ao início lento do TCP.

A taxa mínima de padrão é de 240 bytes/segundo com um período de cortesia de 5 segundos.

Uma taxa mínima também se aplica à resposta. O código para definir o limite de solicitação e o limite de resposta é o mesmo, exceto por ter `RequestBody` ou `Response` nos nomes da propriedade e da interface.

Este é um exemplo que mostra como configurar as taxas mínima de dados em *Program.cs*:

```

.ConfigureKestrel((context, options) =>
{
    options.Limits.MaxConcurrentConnections = 100;
    options.Limits.MaxConcurrentUpgradedConnections = 100;
    options.Limits.MaxRequestBodySize = 10 * 1024;
    options.Limits.MinRequestBodyDataRate =
        new MinDataRate(bytesPerSecond: 100, gracePeriod: TimeSpan.FromSeconds(10));
    options.Limits.MinResponseDataRate =
        new MinDataRate(bytesPerSecond: 100, gracePeriod: TimeSpan.FromSeconds(10));
    options.Listen(IPAddress.Loopback, 5000);
    options.Listen(IPAddress.Loopback, 5001, listenOptions =>
    {
        listenOptions.UseHttps("testCert.pfx", "testPassword");
    });
});

```

```

public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .UseKestrel(options =>
    {
        options.Limits.MinRequestBodyDataRate =
            new MinDataRate(bytesPerSecond: 100, gracePeriod:
TimeSpan.FromSeconds(10));
        options.Limits.MinResponseDataRate =
            new MinDataRate(bytesPerSecond: 100, gracePeriod:
TimeSpan.FromSeconds(10));
    });
}

```

Você pode substituir os limites de taxa mínima por solicitação no middleware:

```

app.Run(async (context) =>
{
    context.Features.Get<IHttpMaxRequestBodySizeFeature>()
        .MaxRequestBodySize = 10 * 1024;

    var minRequestRateFeature =
        context.Features.Get<IHttpMinRequestBodyDataRateFeature>();
    var minResponseRateFeature =
        context.Features.Get<IHttpMinResponseDataRateFeature>();

    if (minRequestRateFeature != null)
    {
        minRequestRateFeature.MinDataRate =
            new MinDataRate(
                bytesPerSecond: 100, gracePeriod: TimeSpan.FromSeconds(10));
    }

    if (minResponseRateFeature != null)
    {
        minResponseRateFeature.MinDataRate =
            new MinDataRate(
                bytesPerSecond: 100, gracePeriod: TimeSpan.FromSeconds(10));
    }
}

```

Nenhum desses recursos de taxa referenciados no exemplo anterior está presente no `HttpContext.Features` para solicitações HTTP/2, porque a modificação dos limites de taxa em cada solicitação não é compatível com HTTP/2 devido ao suporte de multiplexação de solicitação do protocolo. Os limites de taxa de todo o servidor configurados por meio de `KestrelServerOptions.Limits` ainda se aplicam a conexões HTTP/1.x e HTTP/2.

## Fluxos máximos por conexão

O `Http2.MaxStreamsPerConnection` limita o número de fluxos de solicitações simultâneas por

conexão HTTP/2. Os fluxos em excesso são recusados.

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .ConfigureKestrel((context, options) =>
    {
    options.Limits.Http2.MaxStreamsPerConnection = 100;
});
```

O valor padrão é 100.

### Tamanho da tabela de cabeçalho

O decodificador HPACK descompacta os cabeçalhos HTTP para conexões HTTP/2. O `Http2.HeaderTableSize` limita o tamanho da tabela de compactação de cabeçalho usada pelo decodificador HPACK. O valor é fornecido em octetos e deve ser maior do que zero (0).

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .ConfigureKestrel((context, options) =>
    {
    options.Limits.Http2.HeaderTableSize = 4096;
});
```

O valor padrão é 4096.

### Tamanho máximo do quadro

O `Http2.MaxFrameSize` indica o tamanho máximo do payload do quadro da conexão HTTP/2 a ser recebido. O valor é fornecido em octetos e deve estar entre  $2^{14}$  (16.384) e  $2^{24-1}$  (16.777.215).

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .ConfigureKestrel((context, options) =>
    {
    options.Limits.Http2.MaxFrameSize = 16384;
});
```

O valor padrão é  $2^{14}$  (16.384).

### Tamanho máximo do cabeçalho de solicitação

O `Http2.MaxRequestHeaderFieldSize` indica o tamanho máximo permitido em octetos de valores de cabeçalho de solicitação. Esse limite se aplica ao nome e ao valor em suas representações compactadas e descompactadas. O valor deve ser maior que zero (0).

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .ConfigureKestrel((context, options) =>
    {
    options.Limits.Http2.MaxRequestHeaderFieldSize = 8192;
});
```

O valor padrão é 8.192.

## Tamanho inicial da janela de conexão

`Http2.InitialConnectionWindowSize` indica os dados máximos do corpo da solicitação em bytes, que o servidor armazena em buffer ao mesmo tempo, agregados em todas as solicitações (fluxos) por conexão. As solicitações também são limitadas por `Http2.InitialStreamWindowSize`. O valor deve ser maior ou igual a 65.535 e menor que  $2^{31}$  (2.147.483.648).

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .ConfigureKestrel((context, options) =>
    {
    options.Limits.Http2.InitialConnectionWindowSize = 131072;
});
```

O valor padrão é 128 KB (131.072).

## Tamanho inicial da janela de fluxo

`Http2.InitialStreamWindowSize` indica o máximo de dados do corpo da solicitação, em bytes, que o servidor armazena em buffer ao mesmo tempo, por solicitação (fluxo). As solicitações também são limitadas por `Http2.InitialStreamWindowSize`. O valor deve ser maior ou igual a 65.535 e menor que  $2^{31}$  (2.147.483.648).

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .ConfigureKestrel((context, options) =>
    {
    options.Limits.Http2.InitialStreamWindowSize = 98304;
});
```

O valor padrão é 96 KB (98.304).

Para obter informações sobre outras opções e limites do Kestrel, confira:

- [KestrelServerOptions](#)
- [KestrelServerLimits](#)
- [ListenOptions](#)

## Configuração do ponto de extremidade

Por padrão, o ASP.NET Core associa a:

- `http://localhost:5000`
- `https://localhost:5001` (quando um certificado de desenvolvimento local está presente)

Um certificado de desenvolvimento é criado:

- Quando o [SDK do .NET Core](#) está instalado.
- A [ferramenta dev-certs](#) é usada para criar um certificado.

Alguns navegadores exigem que você conceda permissão explícita para o navegador para confiar no certificado de desenvolvimento local.

Os modelos de projeto do ASP.NET Core 2.1 e posteriores configuram os aplicativos para serem executados em HTTPS, por padrão, e incluem o [Redirecionamento de HTTPS e o](#)

suporte a HSTS.

Chame os métodos `Listen` ou `ListenUnixSocket` em `KestrelServerOptions` para configurar portas e prefixos de URL para o Kestrel.

`UseUrls`, o argumento de linha de comando `--urls`, a chave de configuração de host `urls` e a variável de ambiente `ASPNETCORE_URLS` também funcionam mas têm limitações que serão indicadas mais adiante nesta seção (um certificado padrão precisa estar disponível para a configuração do ponto de extremidade HTTPS).

Configuração do ASP.NET Core 2.1 `KestrelServerOptions`:

#### **ConfigureEndpointDefaults(Action<ListenOptions>)**

Especifica uma `Action` de configuração a ser executada para cada ponto de extremidade especificado. Chamar `ConfigureEndpointDefaults` várias vezes substitui as `Action`s pela última `Action` especificada.

#### **ConfigureHttpsDefaults(Action<HttpsConnectionAdapterOptions>)**

Especifica uma `Action` de configuração a ser executada para cada ponto de extremidade HTTPS. Chamar `ConfigureHttpsDefaults` várias vezes substitui as `Action`s pela última `Action` especificada.

#### **Configure(IConfiguration)**

Cria um carregador de configuração para configurar o Kestrel que usa uma `IConfiguration` como entrada. A configuração precisa estar no escopo da seção de configuração do Kestrel.

#### **ListenOptions.UseHttps**

Configure o Kestrel para usar HTTPS.

Extensões `ListenOptions.UseHttps`:

- `UseHttps` – Configure o Kestrel para usar HTTPS com o certificado padrão. Gera uma exceção quando não há nenhum certificado padrão configurado.
- `UseHttps(string fileName)`
- `UseHttps(string fileName, string password)`
- `UseHttps(string fileName, string password, Action<HttpsConnectionAdapterOptions> configureOptions)`
- `UseHttps(StoreName storeName, string subject)`
- `UseHttps(StoreName storeName, string subject, bool allowInvalid)`
- `UseHttps(StoreName storeName, string subject, bool allowInvalid, StoreLocation location)`
- `UseHttps(StoreName storeName, string subject, bool allowInvalid, StoreLocation location, Action<HttpsConnectionAdapterOptions> configureOptions)`
- `UseHttps(X509Certificate2 serverCertificate)`
- `UseHttps(X509Certificate2 serverCertificate, Action<HttpsConnectionAdapterOptions> configureOptions)`
- `UseHttps(Action<HttpsConnectionAdapterOptions> configureOptions)`

Parâmetros de `ListenOptions.UseHttps`:

- `filename` é o caminho e o nome do arquivo de um arquivo de certificado, relativo ao diretório que contém os arquivos de conteúdo do aplicativo.
- `password` é a senha necessária para acessar os dados do certificado X.509 .
- `configureOptions` é uma `Action` para configurar as `HttpsConnectionAdapterOptions` .  
Retorna o `ListenOptions` .
- `storeName` é o repositório de certificados do qual o certificado deve ser carregado.

- `subject` é o nome da entidade do certificado.
- `allowInvalid` indica se certificados inválidos devem ser considerados, como os certificados autoassinados.
- `location` é o local do repositório do qual o certificado deve ser carregado.
- `serverCertificate` é o certificado X.509.

Em produção, HTTPS precisa ser configurado explicitamente. No mínimo, um certificado padrão precisa ser fornecido.

Configurações com suporte descritas a seguir:

- Nenhuma configuração
- Substituir o certificado padrão da configuração
- Alterar os padrões no código

#### *Nenhuma configuração*

O Kestrel escuta em `http://localhost:5000` e em `https://localhost:5001` (se houver um certificado padrão disponível).

Especificar URLs usando:

- A variável de ambiente `ASPNETCORE_URLS`.
- O argumento de linha de comando `--urls`.
- A chave de configuração do host `urls`.
- O método de extensão `UseUrls`.

Para obter mais informações, confira [URLs de servidor](#) e [Substituir configuração](#).

O valor fornecido usando essas abordagens pode ser um ou mais pontos de extremidade HTTP e HTTPS (HTTPS se houver um certificado padrão). Configure o valor como uma lista separada por ponto e vírgula (por exemplo,

```
"Urls": "http://localhost:8000; http://localhost:8001")
```

#### *Substituir o certificado padrão da configuração*

[WebHost.CreateDefaultBuilder](#) chama

```
serverOptions.Configure(context.Configuration.GetSection("Kestrel"))
```

 por padrão ao carregar a configuração do Kestrel. Há um esquema de definições de configurações de aplicativo HTTPS padrão disponível para o Kestrel. Configure vários pontos de extremidade, incluindo URLs e os certificados a serem usados, por meio de um arquivo no disco ou de um repositório de certificados.

No exemplo de `appsettings.json` a seguir:

- Defina **AllowInvalid** como `true` para permitir o uso de certificados inválidos (por exemplo, os certificados autoassinados).
- Todo ponto de extremidade HTTPS que não especificar um certificado (**HttpsDefaultCert** no exemplo a seguir) será revertido para o certificado definido em **Certificados > Padrão** ou para o certificado de desenvolvimento.

```
{
  "Kestrel": {
    "EndPoints": {
      "Http": {
        "Url": "http://localhost:5000"
      },
      "HttpsInlineCertFile": {
        "Url": "https://localhost:5001",
        "Certificate": {
          "Path": "<path to .pfx file>",
          "Password": "<certificate password>"
        }
      },
      "HttpsInlineCertStore": {
        "Url": "https://localhost:5002",
        "Certificate": {
          "Subject": "<subject; required>",
          "Store": "<certificate store; defaults to My>",
          "Location": "<location; defaults to CurrentUser>",
          "AllowInvalid": "<true or false; defaults to false>"
        }
      },
      "HttpsDefaultCert": {
        "Url": "https://localhost:5003"
      },
      "Https": {
        "Url": "https://*:5004",
        "Certificate": {
          "Path": "<path to .pfx file>",
          "Password": "<certificate password>"
        }
      }
    },
    "Certificates": {
      "Default": {
        "Path": "<path to .pfx file>",
        "Password": "<certificate password>"
      }
    }
  }
}
```

Uma alternativa ao uso de **Caminho** e **Senha** para qualquer nó de certificado é especificar o certificado usando campos de repositório de certificados. Por exemplo, o certificado **Certificados > Padrão** pode ser especificado como:

```
"Default": {
  "Subject": "<subject; required>",
  "Store": "<cert store; defaults to My>",
  "Location": "<location; defaults to CurrentUser>",
  "AllowInvalid": "<true or false; defaults to false>"
}
```

Observações do esquema:

- Os nomes de pontos de extremidade diferenciam maiúsculas de minúsculas. Por exemplo, `HTTPS` e `Https` são válidos.
- O parâmetro `Url` é necessário para cada ponto de extremidade. O formato desse

parâmetro é o mesmo que o do parâmetro de configuração de `Urls` de nível superior, exceto que ele é limitado a um único valor.

- Esses pontos de extremidade substituem aqueles definidos na configuração de `Urls` de nível superior em vez de serem adicionados a eles. Os pontos de extremidade definidos no código por meio de `Listen` são acumulados com os pontos de extremidade definidos na seção de configuração.
- A seção `Certificate` é opcional. Se a seção `Certificate` não for especificada, os padrões definidos nos cenários anteriores serão usados. Se não houver nenhum padrão disponível, o servidor gerará uma exceção e não poderá ser iniciado.
- A seção `Certificate` é compatível com os certificados de **Caminho–Senha** e **Entidade–Repositório**.
- Qualquer número de pontos de extremidade pode ser definido dessa forma, contanto que eles não causem conflitos de porta.
- `options.Configure(context.Configuration.GetSection("Kestrel"))` retorna um `KestrelConfigurationLoader` com um método `.Endpoint(string name, options => {})` que pode ser usado para complementar as definições de um ponto de extremidade configurado:

```
options.Configure(context.Configuration.GetSection("Kestrel"))
    .Endpoint("HTTPS", opt =>
{
    opt.HttpsOptions.SslProtocols = SslProtocols.Tls12;
});
```

Você também pode acessar o `KestrelServerOptions.ConfigurationLoader` diretamente para continuar a iteração no carregador existente, como aquele fornecido pelo [WebHost.CreateDefaultBuilder](#).

- A seção de configuração de cada ponto de extremidade está disponível nas opções no método `Endpoint` para que as configurações personalizadas possam ser lidas.
- Várias configurações podem ser carregadas chamando `options.Configure(context.Configuration.GetSection("Kestrel"))` novamente com outra seção. Somente a última configuração será usada, a menos que `Load` seja chamado explicitamente nas instâncias anteriores. O metapacote não chama `Load`, portanto, sua seção de configuração padrão pode ser substituída.
- O `KestrelConfigurationLoader` espelha a família de APIs `Listen` de `KestrelServerOptions` como sobrecargas de `Endpoint`, portanto, os pontos de extremidade de código e de configuração podem ser configurados no mesmo local. Essas sobrecargas não usam nomes e consomem somente as definições padrão da configuração.

#### *Alterar os padrões no código*

`ConfigureEndpointDefaults` e `ConfigureHttpsDefaults` podem ser usados para alterar as configurações padrão de `ListenOptions` e `HttpsConnectionAdapterOptions`, incluindo a substituição do certificado padrão especificado no cenário anterior. `ConfigureEndpointDefaults` e `ConfigureHttpsDefaults` devem ser chamados antes que qualquer ponto de extremidade seja configurado.

```
options.ConfigureEndpointDefaults(opt =>
{
    opt.NoDelay = true;
});

options.ConfigureHttpsDefaults(httpsOptions =>
{
    httpsOptions.SslProtocols = SslProtocols.Tls12;
});
```

### *Suporte do Kestrel para SNI*

A [SNI \(Indicação de Nome de Servidor\)](#) pode ser usada para hospedar vários domínios no mesmo endereço IP e na mesma porta. Para que a SNI funcione, o cliente envia o nome do host da sessão segura para o servidor durante o handshake TLS para que o servidor possa fornecer o certificado correto. O cliente usa o certificado fornecido para a comunicação criptografada com o servidor durante a sessão segura que segue o handshake TLS.

O Kestrel permite a SNI por meio do retorno de chamada do `ServerCertificateSelector`. O retorno de chamada é invocado uma vez por conexão para permitir que o aplicativo inspecione o nome do host e selecione o certificado apropriado.

O suporte para SNI requer:

- Execução na estrutura de destino `netcoreapp2.1`. No `netcoreapp2.0` e no `net461`, o retorno de chamada é invocado, mas o `name` é sempre `null`. O `name` também será `null` se o cliente não fornecer o parâmetro de nome do host no handshake TLS.
- Todos os sites executados na mesma instância do Kestrel. Kestrel não é compatível com o compartilhamento de endereço IP e porta entre várias instâncias sem um proxy reverso.

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .ConfigureKestrel((context, options) =>
    {
        options.ListenAnyIP(5005, listenOptions =>
    {
        listenOptions.UseHttps(httpsOptions =>
    {
        var localhostCert = CertificateLoader.LoadFromStoreCert(
            "localhost", "My", StoreLocation.CurrentUser,
            allowInvalid: true);
        var exampleCert = CertificateLoader.LoadFromStoreCert(
            "example.com", "My", StoreLocation.CurrentUser,
            allowInvalid: true);
        var subExampleCert = CertificateLoader.LoadFromStoreCert(
            "sub.example.com", "My", StoreLocation.CurrentUser,
            allowInvalid: true);
        var certs = new Dictionary<string, X509Certificate2>(
            StringComparer.OrdinalIgnoreCase);
        certs["localhost"] = localhostCert;
        certs["example.com"] = exampleCert;
        certs["sub.example.com"] = subExampleCert;

        httpsOptions.ServerCertificateSelector = (connectionContext, name) =>
    {
        if (name != null && certs.TryGetValue(name, out var cert))
        {
            return cert;
        }

        return exampleCert;
    };
});
});
});
```

```

public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .UseKestrel((context, options) =>
    {
        options.ListenAnyIP(5005, listenOptions =>
    {
        listenOptions.UseHttps(httpsOptions =>
    {
        var localhostCert = CertificateLoader.LoadFromStoreCert(
            "localhost", "My", StoreLocation.CurrentUser,
            allowInvalid: true);
        var exampleCert = CertificateLoader.LoadFromStoreCert(
            "example.com", "My", StoreLocation.CurrentUser,
            allowInvalid: true);
        var subExampleCert = CertificateLoader.LoadFromStoreCert(
            "sub.example.com", "My", StoreLocation.CurrentUser,
            allowInvalid: true);
        var certs = new Dictionary<string, X509Certificate2>(
            StringComparer.OrdinalIgnoreCase);
        certs["localhost"] = localhostCert;
        certs["example.com"] = exampleCert;
        certs["sub.example.com"] = subExampleCert;

        httpsOptions.ServerCertificateSelector = (connectionContext, name) =>
    {
        if (name != null && certs.TryGetValue(name, out var cert))
        {
            return cert;
        }

        return exampleCert;
    };
});
});
});
    .Build();

```

### Associar a um soquete TCP

O método [Listen](#) é associado a um soquete TCP, e um lambda de opções permite a configuração do certificado X.509:

```

public static void Main(string[] args)
{
    CreateWebHostBuilder(args).Build().Run();
}

public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .ConfigureKestrel((context, options) =>
    {
        options.Listen(IPAddress.Loopback, 5000);
        options.Listen(IPAddress.Loopback, 5001, listenOptions =>
    {
        listenOptions.UseHttps("testCert.pfx", "testPassword");
    });
});

```

```

public static void Main(string[] args)
{
    CreateWebHostBuilder(args).Build().Run();
}

public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .UseKestrel(options =>
    {
        options.Listen(IPAddress.Loopback, 5000);
        options.Listen(IPAddress.Loopback, 5001, listenOptions =>
        {
            listenOptions.UseHttps("testCert.pfx", "testPassword");
        });
    });
}

```

```

public static void Main(string[] args)
{
    CreateWebHostBuilder(args).Build().Run();
}

public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .UseKestrel(options =>
    {
        options.Listen(IPAddress.Loopback, 5000);
        options.Listen(IPAddress.Loopback, 5001, listenOptions =>
        {
            listenOptions.UseHttps("testCert.pfx", "testPassword");
        });
    });
}

```

O exemplo configura o HTTPS de um ponto de extremidade com [ListenOptions](#). Use a mesma API para definir outras configurações do Kestrel para pontos de extremidade específicos.

No Windows, é possível criar certificados autoassinados usando o [cmdlet New-SelfSignedCertificate do PowerShell](#). Para ver um exemplo sem suporte, confira [UpdateIISExpressSSLForChrome.ps1](#).

Nas plataformas macOS, Linux e Windows, é possível criar certificados usando o [OpenSSL](#).

### **Associar a um soquete do UNIX**

Escute em um soquete Unix com [ListenUnixSocket](#) para melhorar o desempenho com o Nginx, como é mostrado neste exemplo:

```

.ConfigureKestrel((context, options) =>
{
    options.ListenUnixSocket("/tmp/kestrel-test.sock");
    options.ListenUnixSocket("/tmp/kestrel-test.sock", listenOptions =>
    {
        listenOptions.UseHttps("testCert.pfx", "testpassword");
    });
})

```

```

public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .UseKestrel(options =>
    {
        options.ListenUnixSocket("/tmp/kestrel-test.sock");
        options.ListenUnixSocket("/tmp/kestrel-test.sock", listenOptions =>
    {
        listenOptions.UseHttps("testCert.pfx", "testpassword");
    });
});

```

## Porta 0

Quando o número da porta `0` for especificado, o Kestrel se associará dinamicamente a uma porta disponível. O exemplo a seguir mostra como determinar a qual porta o Kestrel realmente se associou no tempo de execução:

```

public void Configure(IApplicationBuilder app)
{
    var serverAddressesFeature =
        app.ServerFeatures.Get<IServerAddressesFeature>();

    app.UseStaticFiles();

    app.Run(async (context) =>
    {
        context.Response.ContentType = "text/html";
        await context.Response
            .WriteAsync("<!DOCTYPE html><html lang=\"en\"><head>" +
                      "<title></title></head><body><p>Hosted by Kestrel</p>");

        if (serverAddressesFeature != null)
        {
            await context.Response
                .WriteAsync("<p>Listening on the following addresses: " +
                           string.Join(", ", serverAddressesFeature.Addresses) +
                           "</p>");
        }

        await context.Response.WriteAsync("<p>Request URL: " +
            $"{context.Request.GetDisplayUrl()}</p>");
    });
}

```

Quando o aplicativo é executado, a saída da janela do console indica a porta dinâmica na qual o aplicativo pode ser acessado:

```
Listening on the following addresses: http://127.0.0.1:48508
```

## Limitações

Configure pontos de extremidade com as seguintes abordagens:

- [UseUrls](#)
- O argumento de linha de comando `--urls`
- A chave de configuração do host `urls`
- A variável de ambiente `ASPNETCORE_URLS`

Esses métodos são úteis para fazer com que o código funcione com servidores que não sejam

o Kestrel. No entanto, esteja ciente das seguintes limitações:

- O protocolo HTTPS não pode ser usado com essas abordagens, a menos que um certificado padrão seja fornecido na configuração do ponto de extremidade HTTPS (por exemplo, usando a configuração `KestrelServerOptions` ou um arquivo de configuração, como já foi mostrado neste tópico).
- Quando ambas as abordagens, `Listen` e `UseUrls`, são usadas ao mesmo tempo, os pontos de extremidade de `Listen` substituem os pontos de extremidade de `UseUrls`.

### Configuração de ponto de extremidade do IIS

Ao usar o IIS, as associações de URL para IIS substituem as associações definidas por `Listen` ou `UseUrls`. Para obter mais informações, confira o tópico [Módulo do ASP.NET Core](#).

#### ListenOptions.Protocols

A propriedade `Protocols` estabelece os protocolos HTTP (`HttpProtocols`) habilitados em um ponto de extremidade de conexão ou para o servidor. Atribua um valor à propriedade `Protocols` com base na enumeração `HttpProtocols`.

VALOR DE ENUMERAÇÃO <code>HTTPPROTOCOLS</code>	PROTOCOLO DE CONEXÃO PERMITIDO
<code>Http1</code>	HTTP/1.1 apenas. Pode ser usado com ou sem TLS.
<code>Http2</code>	HTTP/2 apenas. Usado principalmente com TLS. Poderá ser usado sem TLS apenas se o cliente for compatível com um <a href="#">Modo de conhecimento prévio</a> .
<code>Http1AndHttp2</code>	HTTP/1.1 e HTTP/2. Requer uma conexão TLS e <a href="#">ALPN (Negociação de protocolo de camada de aplicativo)</a> para negociar o HTTP/2; caso contrário, o padrão da conexão será HTTP/1.1.

O protocolo padrão é HTTP/1.1.

Restrições TLS para HTTP/2:

- Versão TLS 1.2 ou posterior
- Renegociação desabilitada
- Compactação desabilitada
- Tamanhos mínimos de troca de chaves efêmera:
  - ECDHE (Diffie-Hellman de curva elíptica) [[RFC4492](#)] – mínimo de 224 bits
  - DHE (Diffie-Hellman de campo finito) [`TLS12`] – mínimo de 2048 bits
- Pacote de criptografia não autorizado

Há suporte para `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256` [`TLS-ECDHE`] com a curva elíptica P-256 [`FIPS186`] por padrão.

O exemplo a seguir permite conexões HTTP/1.1 e HTTP/2 na porta 8000. As conexões são protegidas pela TLS com um certificado fornecido:

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .ConfigureKestrel((context, options) =>
    {
        options.Listen(IPAddress.Any, 8000, listenOptions =>
    {
        listenOptions.Protocols = HttpProtocols.Http1AndHttp2;
        listenOptions.UseHttps("testCert.pfx", "testPassword");
    });
});
```

Crie opcionalmente uma implementação do `IConnectionAdapter` para filtrar os handshakes TLS por conexão para codificações específicas:

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .ConfigureKestrel((context, options) =>
    {
        options.Listen(IPAddress.Any, 8000, listenOptions =>
    {
        listenOptions.Protocols = HttpProtocols.Http1AndHttp2;
        listenOptions.UseHttps("testCert.pfx", "testPassword");
        listenOptions.ConnectionAdapters.Add(new TlsFilterAdapter());
    });
});
```

```

private class TlsFilterAdapter : IConnectionAdapter
{
    public bool IsHttps => false;

    public Task<IAdaptedConnection> OnConnectionAsync(ConnectionAdapterContext context)
    {
        var tlsFeature = context.Features.Get<ITlsHandshakeFeature>();

        // Throw NotSupportedException for any cipher algorithm that you don't
        // wish to support. Alternatively, define and compare
        // ITlsHandshakeFeature.CipherAlgorithm to a list of acceptable cipher
        // suites.
        //
        // A ITlsHandshakeFeature.CipherAlgorithm of CipherAlgorithmType.Null
        // indicates that no cipher algorithm supported by Kestrel matches the
        // requested algorithm(s).
        if (tlsFeature.CipherAlgorithm == CipherAlgorithmType.Null)
        {
            throw new NotSupportedException("Prohibited cipher: " +
                tlsFeature.CipherAlgorithm);
        }

        return Task.FromResult<IAdaptedConnection>(new
AdaptedConnection(context.ConnectionStream));
    }

    private class AdaptedConnection : IAdaptedConnection
    {
        public AdaptedConnection(Stream adaptedStream)
        {
            ConnectionStream = adaptedStream;
        }

        public Stream ConnectionStream { get; }

        public void Dispose()
        {
        }
    }
}

```

*Definir o protocolo com base na configuração*

[WebHost.CreateDefaultBuilder](#) chama

`serverOptions.Configure(context.Configuration.GetSection("Kestrel"))` por padrão ao carregar a configuração do Kestrel.

No exemplo *appsettings.json* a seguir, um protocolo de conexão padrão (HTTP/1.1 e HTTP/2) é estabelecido para todos os pontos de extremidade do Kestrel:

```
{
  "Kestrel": {
    "EndPointDefaults": {
      "Protocols": "Http1AndHttp2"
    }
  }
}
```

O arquivo de configuração de exemplo a seguir estabelece um protocolo de conexão para um ponto de extremidade específico:

```
{
  "Kestrel": {
    "EndPoints": {
      "HttpsDefaultCert": {
        "Url": "https://localhost:5001",
        "Protocols": "Http1AndHttp2"
      }
    }
  }
}
```

Os protocolos especificados no código substituem os valores definidos pela configuração.

## Configuração de transporte

Com a liberação do ASP.NET Core 2.1, o transporte padrão do Kestrel deixa de ser baseado no Libuv, baseando-se agora em soquetes gerenciados. Essa é uma alteração da falha para aplicativos ASP.NET Core 2.0 que atualizam para o 2.1 que chamam [WebHostBuilderLibuvExtensions.UseLibuv](#) e dependem de um dos seguintes pacotes:

- [Microsoft.AspNetCore.Server.Kestrel](#) (referência direta de pacote)
- [Microsoft.AspNetCore.App](#)

Para projetos do ASP.NET Core 2.1 ou posteriores que usam o [metapacote Microsoft.AspNetCore.App](#) e exigem o uso de Libuv:

- Adicione uma dependência para o pacote [Microsoft.AspNetCore.Server.Kestrel.Transport.Libuv](#) ao arquivo de projeto do aplicativo:

```
<PackageReference Include="Microsoft.AspNetCore.Server.Kestrel.Transport.Libuv"
  Version="<LATEST\_VERSION>" />
```

- Chame [WebHostBuilderLibuvExtensions.UseLibuv](#):

```
public class Program
{
  public static void Main(string[] args)
  {
    CreateWebHostBuilder(args).Build().Run();
  }

  public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
      .UseLibuv()
      .UseStartup<Startup>();
}
```

## Prefixos de URL

Ao usar `UseUrls`, o argumento de linha de comando `--urls`, a chave de configuração de host `urls` ou a variável de ambiente `ASPNETCORE_URLS`, os prefixos de URL podem estar em um dos formatos a seguir.

Somente os prefixos de URL HTTP são válidos. O Kestrel não é compatível com HTTPS ao configurar associações de URL que usam `UseUrls`.

- Endereço IPv4 com o número da porta

```
http://65.55.39.10:80/
```

`0.0.0.0` é um caso especial que associa a todos os endereços IPv4.

- Endereço IPv6 com número da porta

```
http://[0:0:0:0:ffff:4137:270a]:80/
```

`[::]` é o equivalente do IPv6 ao IPv4 `0.0.0.0`.

- Nome do host com o número da porta

```
http://contoso.com:80/
```

```
http://*:80/
```

Os nomes de host `*` e `+` não são especiais. Tudo o que não é reconhecido como um endereço IP ou um `localhost` válido é associado a todos os IPs IPv6 e IPv4. Para associar nomes de host diferentes a diferentes aplicativos ASP.NET Core na mesma porta, use o [HTTP.sys](#) ou um servidor proxy reverso, como o IIS, o Nginx ou o Apache.

#### WARNING

A hospedagem em uma configuração de proxy reverso exige a [filtragem de host](#).

- Nome do `localhost` do host com o número da porta ou o IP de loopback com o número da porta

```
http://localhost:5000/
```

```
http://127.0.0.1:5000/
```

```
http://[::1]:5000/
```

Quando o `localhost` for especificado, o Kestrel tentará se associar às interfaces de loopback IPv4 e IPv6. Se a porta solicitada está sendo usada por outro serviço em uma das interfaces de loopback, o Kestrel falha ao ser iniciado. Se uma das interfaces de loopback não estiver disponível por qualquer outro motivo (geralmente porque não há suporte para o IPv6), o Kestrel registra um aviso em log.

## Filtragem de host

Embora o Kestrel permita a configuração com base em prefixos como

`http://example.com:5000`, ele geralmente ignora o nome do host. O host `localhost` é um caso especial usado para a associação a endereços de loopback. Todo host que não for um endereço IP explícito será associado a todos os endereços IP públicos. Cabeçalhos `Host` não são validados.

Como uma solução alternativa, use o Middleware de Filtragem de Host. Middleware de Filtragem de Host é fornecido pelo pacote [Microsoft.AspNetCore.HostFiltering](#), que está incluído no [metapacote Microsoft.AspNetCore.App](#) (ASP.NET Core 2.1 ou posterior). O middleware é adicionado por [CreateDefaultBuilder](#), que chama [AddHostFiltering](#):

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>();
}
```

Middleware de Filtragem de Host está desabilitado por padrão. Para habilitar o middleware, defina uma chave `AllowedHosts` em `appSettings.JSON/appsettings`.

<`EnvironmentName>.json`. O valor dessa chave é uma lista separada por ponto e vírgula de nomes de host sem números de porta:

`appsettings.json`:

```
{
    "AllowedHosts": "example.com;localhost"
}
```

#### NOTE

Middleware de Cabeçalhos Encaminhados também tem uma opção `ForwardedHeadersOptions.AllowedHosts`. Middleware de Cabeçalhos Encaminhados e Middleware de filtragem de Host têm funcionalidades semelhantes para cenários diferentes. A definição de `AllowedHosts` com Middleware de Cabeçalhos Encaminhados é apropriada quando o cabeçalho `Host` não é preservado ao encaminhar solicitações com um servidor proxy reverso ou um平衡ador de carga. A definição de `AllowedHosts` com Middleware de Filtragem de Host é apropriada quando o Kestrel é usado como um servidor de borda voltado ao público ou quando o cabeçalho `Host` é encaminhado diretamente.

Para obter mais informações sobre o Middleware de Cabeçalhos Encaminhados, confira [Configure o ASP.NET Core para trabalhar com servidores proxy e balanceadores de carga](#).

## Recursos adicionais

- [Solucionar problemas de projetos do ASP.NET Core](#)
- [Impor HTTPS no ASP.NET Core](#)
- [Configure o ASP.NET Core para trabalhar com servidores proxy e balanceadores de carga](#)
- [Código-fonte do kestrel](#)
- [RFC 7230: sintaxe e roteamento da mensagem \(Seção 5.4: host\)](#)

# Implementação do servidor Web HTTP.sys no ASP.NET Core

16/01/2019 • 19 minutes to read • [Edit Online](#)

Por [Tom Dykstra](#), [Chris Ross](#) e [Luke Latham](#)

O [HTTP.sys](#) é um [servidor Web para ASP.NET Core](#) executado apenas no Windows. O HTTP.sys é uma alternativa ao servidor [Kestrel](#) e oferece alguns recursos não disponibilizados pelo Kestrel.

## IMPORTANT

O HTTP.sys não é compatível com o [Módulo do ASP.NET Core](#) e não pode ser usado com IIS ou IIS Express.

O HTTP.sys dá suporte aos seguintes recursos:

- [Autenticação do Windows](#)
- Compartilhamento de porta
- HTTPS com SNI
- HTTP/2 sobre TLS (Windows 10 ou posterior)
- Transmissão direta de arquivo
- Cache de resposta
- WebSockets (Windows 8 ou posterior)

Versões do Windows compatíveis:

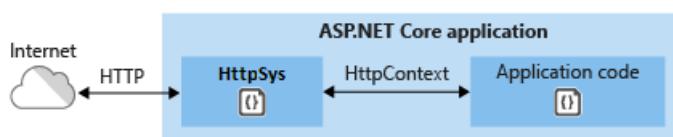
- Windows 7 ou posterior
- Windows Server 2008 R2 ou posterior

[Exibir ou baixar código de exemplo \(como baixar\)](#)

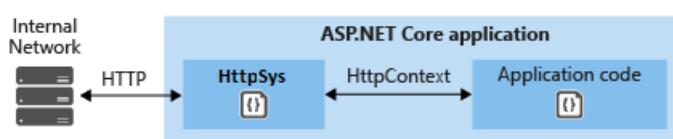
## Quando usar o HTTP.sys

O HTTP.sys é útil nas implantações em que:

- É necessário expor o servidor diretamente à Internet sem usar o IIS.



- As implantações internas exigem um recurso que não está disponível no Kestrel, como a [Autenticação do Windows](#).



O HTTP.sys é uma tecnologia madura que protege contra vários tipos de ataques e proporciona as propriedades de robustez, segurança e escalabilidade de um servidor Web completo. O próprio IIS é executado como um ouvinte HTTP sobre o HTTP.sys.

## Compatibilidade com HTTP/2

O [HTTP/2](#) estará habilitado para aplicativos ASP.NET Core se os seguintes requisitos básicos forem atendidos:

- Windows Server 2016/Windows 10 ou posterior
- Conexão [ALPN \(Negociação de protocolo de camada de aplicativo\)](#)
- Conexão TLS 1.2 ou posterior

Se uma conexão HTTP/2 for estabelecida, `HttpRequest.Protocol` relatará `HTTP/2`.

Se uma conexão HTTP/2 for estabelecida, `HttpRequest.Protocol` relatará `HTTP/1.1`.

O HTTP/2 está habilitado por padrão. Se uma conexão HTTP/2 não tiver sido estabelecida, a conexão retornará para HTTP/1.1. Em uma versão futura do Windows, os sinalizadores de configuração de HTTP/2 estarão disponíveis e contarão com a capacidade de desabilitar o HTTP/2 com HTTP.sys.

## Autenticação de modo kernel com Kerberos

O HTTP.sys delega à autenticação de modo kernel com o protocolo de autenticação Kerberos. Não há suporte para autenticação de modo de usuário com o Kerberos e o HTTP.sys. A conta do computador precisa ser usada para descriptografar o token/tíquete do Kerberos que é obtido do Active Directory e encaminhado pelo cliente ao servidor para autenticar o usuário. Registre o SPN (nome da entidade de serviço) do host, não do usuário do aplicativo.

## Como usar o HTTP.sys

### Configurar o aplicativo ASP.NET Core para usar o HTTP.sys

1. Não é necessário usar uma referência do pacote no arquivo de projeto ao usar o [metapacote Microsoft.AspNetCore.App \(nuget.org\)](#) (ASP.NET Core 2.1 ou posterior). Se não estiver usando o metapacote `Microsoft.AspNetCore.App`, adicione uma referência do pacote a [Microsoft.AspNetCore.Server.HttpSys](#).
2. Chame o método de extensão `UseHttpSys` quando criar o Host Web, especificando todas as [opções do HTTP.sys](#) necessárias:

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .UseHttpSys(options =>
    {
        // The following options are set to default values.
        options.Authentication.Schemes = AuthenticationSchemes.None;
        options.Authentication.AllowAnonymous = true;
        options.MaxConnections = null;
        options.MaxRequestBodySize = 30000000;
        options.UrlPrefixes.Add("http://localhost:5000");
    });
}
```

A configuração adicional do HTTP.sys é tratada por meio das [configurações do registro](#).

### Opções do HTTP.sys

PROPRIEDADE	DESCRIÇÃO	PADRÃO
-------------	-----------	--------

PROPRIEDADE	DESCRIÇÃO	PADRÃO
AllowSynchronousIO	Controlar quando a Entrada/Saída síncrona deve ser permitida para <code>HttpContext.Request.Body</code> e <code>HttpContext.Response.Body</code> .	<code>true</code>
Authentication.AllowAnonymous	Permitir solicitações anônimas.	<code>true</code>
Authentication.Schemes	Especificar os esquemas de autenticação permitidos. É possível modificar a qualquer momento antes de descartar o ouvinte. Os valores são fornecidos pela enumeração <code>AuthenticationSchemes</code> : <code>Basic</code> , <code>Kerberos</code> , <code>Negotiate</code> , <code>None</code> e <code>NTLM</code> .	<code>None</code>
EnableResponseCaching	Tentativa de cache do modo <code>kernel</code> para obtenção de respostas com cabeçalhos qualificados. A resposta pode não incluir <code>Set-Cookie</code> , <code>Vary</code> ou cabeçalhos <code>Pragma</code> . Ela deve incluir um cabeçalho <code>Cache-Control</code> que seja <code>public</code> e um valor <code>shared-max-age</code> ou <code>max-age</code> , ou um cabeçalho <code>Expires</code> .	<code>true</code>
MaxAccepts	O número máximo de aceitações simultâneas.	$5 \times \text{Ambiente.ProcessorCount}$
MaxConnections	O número máximo de conexões simultâneas a serem aceitas. Usar <code>-1</code> como infinito. Usar <code>null</code> a fim de usar a configuração que abranja toda máquina do registro.	<code>null</code> (ilimitado)
MaxRequestBodySize	Confira a seção <a href="#">MaxRequestBodySize</a> .	30.000.000 de bytes (28,6 MB)
RequestQueueLimit	O número máximo de solicitações que podem ser colocadas na fila.	1000
ThrowWriteExceptions	Indica se as gravações do corpo da resposta que falham quando o cliente se desconecta devem gerar exceções ou serem concluídas normalmente.	<code>false</code> (concluir normalmente)

PROPRIEDADE	DESCRIÇÃO	PADRÃO
<a href="#">Timeouts</a>	<p>Expor a configuração <a href="#">TimeoutManager</a> do HTTP.sys, que também pode ser definida no registro. Siga os links de API para saber mais sobre cada configuração, inclusive os valores padrão:</p> <ul style="list-style-type: none"> <li>• <a href="#">TimeoutManager.DrainEntityBody</a> – O tempo permitido para que a API do servidor HTTP esvazie o corpo da entidade em uma conexão Keep-Alive.</li> <li>• <a href="#">TimeoutManager.EntityBody</a> – O tempo permitido para a chegada do corpo da entidade de solicitação.</li> <li>• <a href="#">TimeoutManager.HeaderWait</a> – O tempo permitido para que a API do servidor HTTP analise o cabeçalho da solicitação.</li> <li>• <a href="#">TimeoutManager.IdleConnection</a> – O tempo permitido para uma conexão ociosa.</li> <li>• <a href="#">TimeoutManager.MinSendBytesPerSecond</a> – A taxa de envio mínima para a resposta.</li> <li>• <a href="#">TimeoutManager.RequestQueue</a> – O tempo permitido para que a solicitação permaneça na fila de solicitações até o aplicativo coletá-la.</li> </ul>	
<a href="#">UrlPrefixes</a>	<p>Especifique a <a href="#">UrlPrefixCollection</a> para registrar com o HTTP.sys. A mais útil é <a href="#">UrlPrefixCollection.Add</a>, que é usada para adicionar um prefixo à coleção. É possível modificá-las a qualquer momento antes de descartar o ouvinte.</p>	

### MaxRequestBodySize

O tamanho máximo permitido em bytes para todos os corpos de solicitação. Quando é definido como `null`, o tamanho máximo do corpo da solicitação é ilimitado. Esse limite não afeta as conexões atualizadas que são sempre ilimitadas.

O método recomendado para substituir o limite em um aplicativo ASP.NET Core MVC para um único `IActionResult` é usar o atributo [RequestSizeLimitAttribute](#) em um método de ação:

```
[RequestSizeLimit(100000000)]
public IActionResult MyActionMethod()
```

Uma exceção é gerada quando o aplicativo tenta configurar o limite de uma solicitação, depois que o aplicativo inicia a leitura da solicitação. É possível usar uma propriedade `IsReadOnly` para indicar se a propriedade `MaxRequestBodySize` está no estado somente leitura, o que significa que é tarde demais para configurar o limite.

Se o aplicativo tiver que substituir `MaxRequestBodySize` por solicitação, use `IHttpMaxRequestBodySizeFeature`:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
    ILogger<Startup> logger, IServer server)
{
    app.Use(async (context, next) =>
    {
        context.Features.Get<IHttpMaxRequestBodySizeFeature>()
            .MaxRequestBodySize = 10 * 1024;

        var serverAddressesFeature =
            app.ServerFeatures.Get<IServerAddressesFeature>();
        var addresses = string.Join(", ", serverAddressesFeature?.Addresses);

        logger.LogInformation($"Addresses: {addresses}");

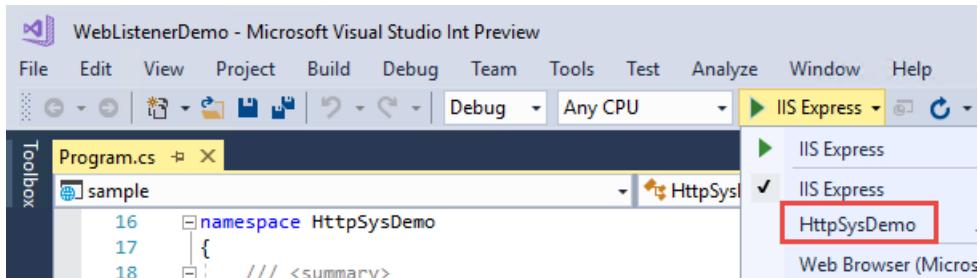
        await next.Invoke();
    });

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }

    // Enable HTTPS Redirection Middleware when hosting the app securely.
    //app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseCookiePolicy();
    app.UseMvc();
}
```

3. Quando usar o Visual Studio, verifique se que o aplicativo está configurado para executar o IIS ou IIS Express.

No Visual Studio, o perfil de inicialização padrão destina-se ao IIS Express. Para executar o projeto como um aplicativo de console, altere manualmente o perfil selecionado, conforme mostrado na captura de tela a seguir:



## Configurar o Windows Server

1. Determine as portas que serão abertas para o aplicativo e use o Firewall do Windows ou os cmdlets do PowerShell para abrir as portas de firewall e permitir que o tráfego chegue até o HTTP.sys. Ao implantar em uma VM do Azure, abra as portas no Grupo de Segurança de Rede.

Nos seguintes comandos e configuração de aplicativo, a porta 443 é usada.

2. Obtenha e instale os certificados X.509, se precisar.

No Windows, crie certificados autoassinados, usando o [cmdlet do PowerShell New-SelfSignedCertificate](#). Para ver um exemplo sem suporte, confira [UpdateIISExpressSSLForChrome.ps1](#).

Instale certificados autoassinados ou assinados pela AC no repositório **Computador Local > Pessoal** do servidor.

3. Se o aplicativo for uma [implantação dependente de estrutura](#), instale o .NET Core, o .NET Framework ou ambos (caso o aplicativo .NET Core seja direcionado ao .NET Framework).

- **.NET Core** – Se o aplicativo exigir o .NET Core, obtenha e execute o instalador do [Tempo de Execução do .NET Core](#) em [Downloads do .NET Core](#). Não instale o SDK completo no servidor.
- **.NET framework** – Se o aplicativo exigir o .NET Framework, confira o [Guia de instalação do .NET Framework](#). Instale o .NET Framework necessário. O instalador do .NET Framework mais recente está disponível na página [Downloads do .NET Core](#).

Se o aplicativo for uma [implantação autocontida](#), ele incluirá o tempo de execução em sua implantação. Nenhuma instalação do framework é necessária no servidor.

4. Configure URLs e portas no aplicativo.

Por padrão, o ASP.NET Core é associado a `http://localhost:5000`. Para configurar portas e prefixos de URL, as opções incluem:

- `UseUrls`
- O argumento de linha de comando `urls`
- A variável de ambiente `ASPNETCORE_URLS`
- `UrlPrefixes`

O exemplo de código a seguir mostra como usar `UrlPrefixes` com o endereço IP local do servidor `10.0.0.4` na porta 443:

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .UseHttpSys(options =>
    {
        options.Authentication.Schemes = AuthenticationSchemes.None;
        options.Authentication.AllowAnonymous = true;
        options.MaxConnections = null;
        options.MaxRequestBodySize = 30000000;
        // Server local IP address: 10.0.0.4
        options.UrlPrefixes.Add("https://10.0.0.4:443");
    });
}
```

Uma vantagem de usar `UrlPrefixes` é que uma mensagem de erro é gerada imediatamente no caso de prefixos formatados de forma incorreta.

As configurações de `UrlPrefixes` substituem as configurações `UseUrls` / `urls` / `ASPNETCORE_URLS`. Portanto, uma vantagem de usar `UseUrls`, `urls` e a variável de ambiente `ASPNETCORE_URLS` é que fica mais fácil alternar entre o Kestrel e o HTTP.sys. Para obter mais informações, consulte [Host da Web do ASP.NET Core](#).

O HTTP.sys usa os [formatos de cadeia de caracteres UrlPrefix da API do Servidor HTTP](#).

#### WARNING

Associações de curinga de nível superior (`http://*:80/` e `http://+:80`) **não** devem ser usadas. Associações de curinga de nível superior criam vulnerabilidades de segurança no aplicativo. Isso se aplica a curingas fortes e fracos. Use nomes de host explícitos ou endereços IP em vez de curingas. Associações de curinga de subdomínio (por exemplo, `*.mysub.com`) não serão um risco à segurança se você controlar todo o domínio pai (ao contrário de `*.com`, o qual é vulnerável). Para saber mais, confira [RFC 7230: Seção 5.4: Host](#).

## 5. Pré-registre os prefixos de URL no servidor.

O `netsh.exe` é a ferramenta interna destinada a configurar o HTTP.sys. Com o `netsh.exe`, é possível reservar prefixos de URL e atribuir certificados X.509. A ferramenta exige privilégios de administrador.

Use a ferramenta `netsh.exe` para registrar as URLs do aplicativo:

```
netsh http add urlacl url=<URL> user=<USER>
```

- `<URL>` – A URL (Uniform Resource Locator) totalmente qualificada. Não use uma associação de curinga. Use um nome de host válido ou o endereço IP local. *A URL deve incluir uma barra à direita.*
- `<USER>` – Especifica o nome de usuário ou do grupo de usuários.

No exemplo a seguir, o endereço IP local do servidor é `10.0.0.4`:

```
netsh http add urlacl url=https://10.0.0.4:443/ user=Users
```

Quando uma URL é registrada, a ferramenta responde com `URL reservation successfully added`.

Para excluir uma URL registrada, use o comando `delete urlacl`:

```
netsh http delete urlacl url=<URL>
```

## 6. Registre certificados X.509 no servidor.

Use a ferramenta `netsh.exe` para registrar certificados do aplicativo:

```
netsh http add sslcert ipport=<IP>:<PORT> certhash=<THUMBPRINT> appid="{<GUID>}"
```

- `<IP>` – Especifica o endereço IP local para a associação. Não use uma associação de curinga. Use um endereço IP válido.
- `<PORT>` – Especifica a porta da associação.
- `<THUMBPRINT>` – A impressão digital do certificado X.509.
- `<GUID>` – Um GUID gerado pelo desenvolvedor para representar o aplicativo para fins informativos.

Para fins de referência, armazene o GUID no aplicativo como uma marca de pacote:

- No Visual Studio:
  - Abra as propriedades do projeto do aplicativo, clicando com o botão direito do mouse no aplicativo no **Gerenciador de Soluções** e selecionando **Propriedades**.
  - Selecione a guia **Pacote**.

- Insira o GUID que você criou no campo **Marcas**.
- Quando não estiver usando o Visual Studio:
  - Abra o arquivo de projeto do aplicativo.
  - Adicione uma propriedade `<PackageTags>` a um `<PropertyGroup>` novo ou existente com o GUID que você criou:

```
<PropertyGroup>
  <PackageTags>9412ee86-c21b-4eb8-bd89-f650fbf44931</PackageTags>
</PropertyGroup>
```

No exemplo a seguir:

- O endereço IP local do servidor é `10.0.0.4`.
- Um gerador GUID aleatório online fornece o valor `appid`.

```
netsh http add sslcert
  ipport=10.0.0.4:443
  certhash=b66ee04419d4ee37464ab8785ff02449980eae10
  appid="{9412ee86-c21b-4eb8-bd89-f650fbf44931}"
```

Quando um certificado é registrado, a ferramenta responde com  
`SSL Certificate successfully added`.

Para excluir um registro de certificado, use o comando `delete sslcert`:

```
netsh http delete sslcert ipport=<IP>:<PORT>
```

Documentação de referência do *netsh.exe*:

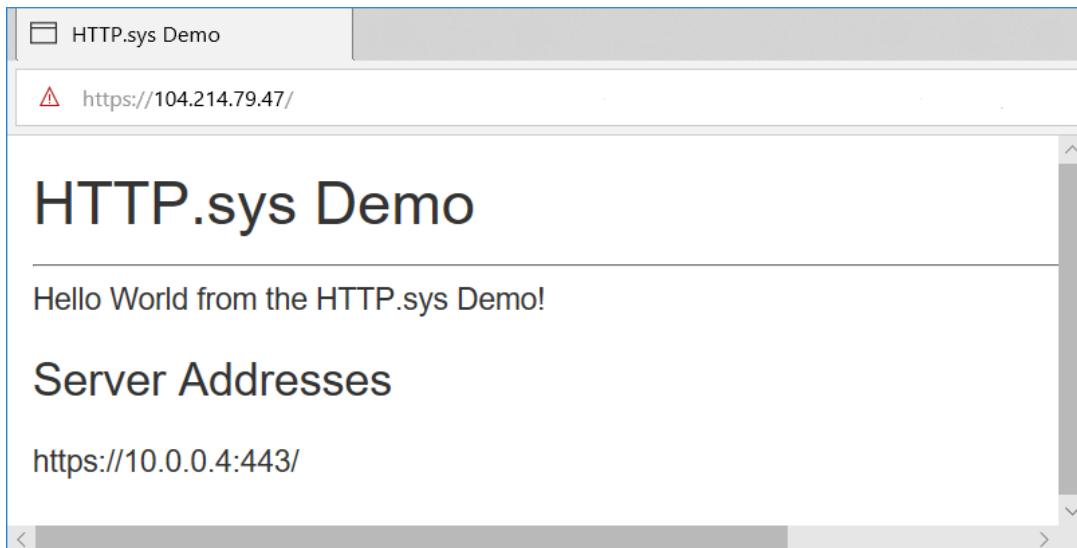
- [Comandos do Netsh para o protocolo HTTP](#)
- [Cadeias de caracteres de UrlPrefix](#)

## 7. Execute o aplicativo.

Não é necessário ter privilégios de administrador para executar o aplicativo ao associar ao localhost usando HTTP (não HTTPS) com um número de porta maior do que 1024. Para outras configurações (por exemplo, usar um endereço IP local ou associação à porta 443), execute o aplicativo com privilégios de administrador.

O aplicativo responde no endereço IP público do servidor. Neste exemplo, o servidor é acessado pela Internet como seu endereço IP público de `104.214.79.47`.

Um certificado de desenvolvimento é usado neste exemplo. A página é carregada com segurança após ignorar o aviso de certificado não confiável do navegador.



## Servidor proxy e cenários de平衡ador de carga

Para aplicativos hospedados pelo HTTP.sys que interagem com solicitações da Internet ou de uma rede corporativa, podem ser necessárias configurações adicionais ao hospedar atrás de平衡adores de carga e de servidores proxy. Para obter mais informações, veja [Configurar o ASP.NET Core para trabalhar com servidores proxy e平衡adores de carga](#).

## Recursos adicionais

- [Habilitar a autenticação do Windows com HTTP.sys](#)
- [API do servidor HTTP](#)
- [Repositório aspnet/HttpSysServer do GitHub \(código-fonte\)](#)
- [Host da Web e Host Genérico no ASP.NET Core](#)
- [Solucionar problemas de projetos do ASP.NET Core](#)

# Hospedar o ASP.NET Core em um serviço Windows

30/01/2019 • 19 minutes to read • [Edit Online](#)

Por [Luke Latham](#) e [Tom Dykstra](#)

Um aplicativo ASP.NET Core pode ser hospedado no Windows somo um [Serviço Windows](#) sem usar o IIS. Quando hospedado como um Serviço Windows, o aplicativo é iniciado automaticamente após a reinicialização.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Tipo de implantação

Você pode criar uma implantação de Serviço Windows dependente de estrutura ou autocontida. Para saber mais e obter conselhos sobre cenários de implantação, consulte [Implantação de aplicativos .NET Core](#).

### Implantação dependente de estrutura

A FDD (Implantação Dependente de Estrutura) se baseia na presença de uma versão compartilhada em todo o sistema do .NET Core no sistema de destino. Quando o cenário FDD é usado com um aplicativo de Serviço Windows do ASP.NET Core, o SDK produz um executável (\*.exe), chamado de *executável dependente de estrutura*.

### Implantação autocontida

A SCD (Implantação Autocontida) não se baseia na presença de componentes compartilhados no sistema de destino. O tempo de execução e as dependências do aplicativo são implantados com o aplicativo para o sistema de hospedagem.

## Converter um projeto em um serviço Windows

Faça as seguintes alterações em um projeto ASP.NET Core existente para executar o aplicativo como um serviço:

### Atualizações de arquivo de projeto

Com base na sua escolha de [tipo de implantação](#), atualize o arquivo de projeto:

#### FDD (Implantação dependente de estrutura)

Adicione um [RID \(Identificador do Tempo de Execução\)](#) do Windows ao `<PropertyGroup>` que contém a estrutura de destino. No exemplo a seguir, o RID é especificado como `win7-x64`. Adicione a propriedade `<SelfContained>` definida como `false`. Essas propriedades instruem o SDK a gerar um arquivo executável (.exe) para Windows.

O arquivo `web.config`, que normalmente é gerado durante a publicação de um aplicativo ASP.NET Core, é desnecessário para um aplicativo de serviços do Windows. Para desabilitar a criação de um arquivo `web.config`, adicione a propriedade `<IsTransformWebConfigDisabled>` definida como `true`.

```
<PropertyGroup>
  <TargetFramework>netcoreapp2.2</TargetFramework>
  <RuntimeIdentifier>win7-x64</RuntimeIdentifier>
  <SelfContained>false</SelfContained>
  <IsTransformWebConfigDisabled>true</IsTransformWebConfigDisabled>
</PropertyGroup>
```

Adicione a propriedade `<useAppHost>` definida como `true`. Essa propriedade fornece o serviço com um caminho de ativação (um arquivo executável .exe) para FDD.

```
<PropertyGroup>
  <TargetFramework>netcoreapp2.1</TargetFramework>
  <RuntimeIdentifier>win7-x64</RuntimeIdentifier>
  <UseAppHost>true</UseAppHost>
  <SelfContained>false</SelfContained>
  <IsTransformWebConfigDisabled>true</IsTransformWebConfigDisabled>
</PropertyGroup>
```

#### SCD (Implantação Autossuficiente)

Confirme a presença de um [RID \(Identificador de Tempo de Execução\)](#) do Windows ou adicione um RID ao `<PropertyGroup>` que contém a estrutura de destino. Desabilite a criação de um arquivo `web.config` adicionando a propriedade `<IsTransformWebConfigDisabled>` definida como `true`.

```
<PropertyGroup>
  <TargetFramework>netcoreapp2.2</TargetFramework>
  <RuntimeIdentifier>win7-x64</RuntimeIdentifier>
  <IsTransformWebConfigDisabled>true</IsTransformWebConfigDisabled>
</PropertyGroup>
```

Para publicar para vários RIDs:

- Forneça os RIDs em uma lista delimitada por ponto e vírgula.
- Use o nome da propriedade `<RuntimeIdentifiers>` (plural).

Para obter mais informações, consulte [Catálogo de RID do .NET Core](#).

Adicionar uma referência de pacote para [Microsoft.AspNetCore.Hosting.WindowsServices](#).

Para habilitar o registro em log do Log de Eventos do Windows, adicione uma referência de pacote para [Microsoft.Extensions.Logging.EventLog](#).

Para saber mais, consulte a seção [Manipular eventos de início e de parada](#).

#### Atualizações de Program.Main

Faça as seguintes alterações em `Program.Main`:

- Para testar e depurar quando a execução estiver sendo feita fora de um serviço, adicione código para determinar se o aplicativo está sendo executado como um serviço ou aplicativo de console. Verifique se o depurador está anexado ou se há um argumento de linha de comando do `--console` presente.

Se uma dessas condições for verdadeira (o aplicativo não for executado como um serviço), chame [Run](#) no Host da Web.

Se as condições forem falsas (o aplicativo for executado como um serviço):

- Chame o [SetCurrentDirectory](#) e use um caminho para o local publicado do aplicativo. Não chame o [GetCurrentDirectory](#) para obter o caminho porque um aplicativo de Serviço Windows retorna a pasta `C:\WINDOWS\system32` quando `GetCurrentDirectory` é chamado. Para saber mais, consulte a seção [Diretório atual e a raiz do conteúdo](#).
- Chame o [RunAsService](#) para executar o aplicativo como um serviço.

Como o [Provedor de Configuração da Linha de Comando](#) requer pares nome-valor para argumentos de linha de comando, a opção `--console` é removida dos argumentos antes de o [CreateDefaultBuilder](#) receberlos.

- Para gravar no Log de Eventos do Windows, adicione o Provedor de Log de Eventos a [ConfigureLogging](#). Defina o nível de log com a chave `Logging:LogLevel:Default` no arquivo `appsettings.Production.JSON`. Para fins de teste e demonstração, o arquivo de configurações de Produção do exemplo define o

nível de log para `Information`. Na produção, o valor normalmente é definido como `Error`. Para obter mais informações, consulte [Registro em log no ASP.NET Core](#).

```
public class Program
{
    public static void Main(string[] args)
    {
        var isService = !(Debugger.IsAttached || args.Contains("--console"));

        if (isService)
        {
            var pathToExe = Process.GetCurrentProcess().MainModule.FileName;
            var pathToContentRoot = Path.GetDirectoryName(pathToExe);
            Directory.SetCurrentDirectory(pathToContentRoot);
        }

        var builder = CreateWebHostBuilder(
            args.Where(arg => arg != "--console").ToArray());

        var host = builder.Build();

        if (isService)
        {
            // To run the app without the CustomWebHostService change the
            // next line to host.RunAsService();
            host.RunAsCustomService();
        }
        else
        {
            host.Run();
        }
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .ConfigureLogging((hostingContext, logging) =>
            {
                logging.AddEventLog();
            })
            .ConfigureAppConfiguration((context, config) =>
            {
                // Configure the app here.
            })
            .UseStartup<Startup>();
}
```

## Publique o aplicativo

Publique o aplicativo usando [dotnet publish](#), um [perfil de publicação do Visual Studio](#) ou o Visual Studio Code. Ao usar o Visual Studio, selecione o **FolderProfile** e configure o **Local de destino** antes de selecionar o botão **Publicar**.

Para publicar o aplicativo de exemplo usando as ferramentas da CLI (Interface de Linha de Comando), execute o comando [dotnet publish](#) em um prompt de comando da pasta do projeto, passando a configuração de uma versão para a opção `-c|--configuration`. Use a opção `-o|--output` com um caminho para publicar em uma pasta fora do aplicativo.

### Publicar uma FDD (Implantação Dependente de Estrutura)

No exemplo a seguir, o aplicativo é publicado na pasta `c:\svc`:

```
dotnet publish --configuration Release --output c:\svc
```

### Publicar uma SCD (Implantação Autossuficiente)

O RID deve ser especificado na propriedade `<RuntimeIdentifier>` (ou `<RuntimeIdentifiers>`) do arquivo de projeto. Insira o tempo de execução na opção `-r|--runtime` do comando `dotnet publish`.

No exemplo a seguir, o aplicativo é publicado para o tempo de execução `win7-x64` para a pasta `c:\svc`:

```
dotnet publish --configuration Release --runtime win7-x64 --output c:\svc
```

## Criar uma conta de usuário

Crie uma conta de usuário para o serviço usando o comando `net user`:

```
net user {USER ACCOUNT} {PASSWORD} /add
```

Para o aplicativo de exemplo, crie uma conta de usuário com o nome `ServiceUser` e uma senha. No comando a seguir, substitua `{PASSWORD}` por uma [senha forte](#).

```
net user ServiceUser {PASSWORD} /add
```

Se você precisar adicionar o usuário a um grupo, use o comando `net localgroup`, onde `{GROUP}` é o nome do grupo:

```
net localgroup {GROUP} {USER ACCOUNT} /add
```

Para obter mais informações, confira [Contas de Usuário do Serviço](#).

## Configurar permissões

Conceda acesso de gravação/leitura/execução para a pasta do aplicativo usando o comando `icacls`:

```
icacls "{PATH}" /grant {USER ACCOUNT}:(OI)(CI){PERMISSION FLAGS} /t
```

- `{PATH}` – Caminho para a pasta do aplicativo.
- `{USER ACCOUNT}` – A conta de usuário (SID).
- `(OI)` – O sinalizador de Herança de Objeto propaga as permissão para os arquivos subordinados.
- `(CI)` – O sinalizador de Herança de Contêiner propaga as permissão para as pastas subordinadas.
- `{PERMISSION FLAGS}` – Define as permissões de acesso do aplicativo.
  - Gravar (`w`)
  - Ler (`r`)
  - Executar (`x`)
  - Completo (`f`)
  - Modificar (`m`)
- `/t` – Aplique recursivamente aos arquivos e pastas subordinadas existentes.

Para o aplicativo de exemplo publicado na pasta `c:\svc` e a conta `ServiceUser` com permissões de gravação/leitura/execução, use o seguinte comando:

```
icacls "c:\svc" /grant ServiceUser:(OI)(CI)WRX /t
```

Para obter mais informações, confira [icacls](#).

# Gerenciar o serviço

## Criar o serviço

Use a ferramenta de linha de comando `sc.exe` para criar o serviço. O valor `binPath` é o caminho para o executável do aplicativo, que inclui o nome do arquivo executável. **É necessário o espaço entre o sinal de igual e o caractere de aspas de cada parâmetro e valor.**

```
sc create {SERVICE NAME} binPath= "{PATH}" obj= "{DOMAIN}\{USER ACCOUNT}" password= "{PASSWORD}"
```

- `{SERVICE NAME}` – O nome a ser atribuído ao serviço no [Gerenciador de Controle de Serviço](#).
- `{PATH}` – O caminho para o executável do serviço.
- `{DOMAIN}` – O domínio de um computador ingressado no domínio. Se o computador não estiver ingressado no domínio, o nome do computador local.
- `{USER ACCOUNT}` – A conta do usuário na qual o serviço é executado.
- `{PASSWORD}` – A senhas da conta de usuário.

### WARNING

**Não omita o parâmetro `obj`.** O valor padrão para `obj` é a [conta LocalSystem](#). Executar um serviço na conta `LocalSystem` apresenta um risco de segurança significativo. Sempre execute um serviço com uma conta de usuário com privilégios restritos.

Observe o seguinte aplicativo de exemplo:

- O nome do serviço é **MyService**.
- O serviço publicado reside na pasta `c:\svc`. O executável do aplicativo é chamado de `SampleApp.exe`. Coloque o valor `binPath` entre aspas duplas ("").
- O serviço é executado na conta `ServiceUser`. Substitua `{DOMAIN}` com o domínio ou nome do computador local da conta de usuário. Coloque o valor `obj` entre aspas duplas (""). Exemplo: se o sistema de hospedagem é um computador local denominado `MairaPC`, defina `obj` como `"MairaPC\ServiceUser"`.
- Substitua `{PASSWORD}` com a senha da conta do usuário. Coloque o valor `password` entre aspas duplas ("").

```
sc create MyService binPath= "c:\svc\sampleapp.exe" obj= "{DOMAIN}\ServiceUser" password= "{PASSWORD}"
```

### IMPORTANT

Certifique-se de que os espaços entre os sinais de igual e os valores dos parâmetros estão presentes.

## Iniciar o serviço

Inicie o serviço com o comando `sc start {SERVICE NAME}`.

Para iniciar o serviço de aplicativo de exemplo, use o seguinte comando:

```
sc start MyService
```

O comando leva alguns segundos para iniciar o serviço.

## Determinar o status do serviço

Para verificar o status do serviço, use o comando `sc query {SERVICE NAME}`. O status é relatado como um dos

seguintes valores:

- START\_PENDING
- RUNNING
- STOP\_PENDING
- STOPPED

Use o seguinte comando para verificar o status do serviço de aplicativo de exemplo:

```
sc query MyService
```

### Procurar um serviço de aplicativo Web

Quando o serviço estiver no estado `RUNNING` e se o serviço for um aplicativo Web, procure o aplicativo em seu caminho (por padrão, `http://localhost:5000`, que redireciona para `https://localhost:5001` ao usar [Middleware de Redirecionamento HTTPS](#)).

Para o serviço de aplicativo de exemplo, procure o aplicativo em `http://localhost:5000`.

### Parar o serviço

Interrompa o serviço com o comando `sc stop {SERVICE NAME}`.

O comando a seguir interrompe o serviço de aplicativo de exemplo:

```
sc stop MyService
```

### Excluir o serviço

Após um pequeno atraso para interromper um serviço, desinstale o serviço com o comando

```
sc delete {SERVICE NAME}
```

Verifique o status do serviço de aplicativo de exemplo:

```
sc query MyService
```

Quando o serviço de aplicativo de exemplo estiver no estado `STOPPED`, use o seguinte comando para desinstalar o serviço de aplicativo de exemplo:

```
sc delete MyService
```

## Manipular eventos de início e de parada

Para manipular os eventos `OnStarting`, `OnStarted` e `OnStopping`, faça as seguintes alterações adicionais:

1. Crie uma classe que derive de `WebHostService` com os métodos `OnStarting`, `OnStarted` e `OnStopping`:

```

internal class CustomWebHostService : WebHostService
{
    private ILogger _logger;

    public CustomWebHostService(IWebHost host) : base(host)
    {
        _logger = host.Services
            .GetRequiredService<ILogger<CustomWebHostService>>();
    }

    protected override void OnStarting(string[] args)
    {
        _logger.LogInformation("OnStarting method called.");
        base.OnStarting(args);
    }

    protected override void OnStarted()
    {
        _logger.LogInformation("OnStarted method called.");
        base.OnStarted();
    }

    protected override void OnStopping()
    {
        _logger.LogInformation("OnStopping method called.");
        base.OnStopping();
    }
}

```

2. Crie um método de extensão para `IWebHost` que passe o `CustomWebHostService` para `Run`:

```

public static class WebHostServiceExtensions
{
    public static void RunAsCustomService(this IWebHost host)
    {
        var webHostService = new CustomWebHostService(host);
        ServiceBase.Run(webHostService);
    }
}

```

3. Em `Program.Main`, chame o método de extensão `RunAsCustomService`, em vez de `RunAsService`:

```

host.RunAsCustomService();

```

Para ver o local do `RunAsService` no `Program.Main`, consulte o exemplo de código mostrado na seção [Converter um projeto em um Serviço Windows](#).

## Servidor proxy e cenários de平衡ador de carga

Serviços que interagem com solicitações da Internet ou de uma rede corporativa e estão atrás de um proxy ou de um balanceador de carga podem exigir configuração adicional. Para obter mais informações, consulte [Configure o ASP.NET Core para trabalhar com servidores proxy e balanceadores de carga](#).

## Configurar o HTTPS

Para configurar o serviço com um ponto de extremidade seguro:

- Crie um certificado X.509 para o sistema de hospedagem usando os mecanismos de implantação e aquisição do certificado da sua plataforma.

2. Especifique uma [Configuração do ponto de extremidade HTTPS do servidor Kestrel](#) para usar o certificado.

Não há suporte para uso do certificado de desenvolvimento HTTPS do ASP.NET Core para proteger um ponto de extremidade de serviço.

## Diretório atual e a raiz do conteúdo

O diretório de trabalho atual retornado ao chamar [GetCurrentDirectory](#) de um serviço Windows é a pasta C:\WINDOWS\system32. A pasta system32 não é um local adequado para armazenar os arquivos de um serviço (por exemplo, os arquivos de configurações). Use uma das seguintes abordagens para manter e acessar ativos e os arquivos de configuração de um serviço.

### Defina o caminho da raiz do conteúdo para a pasta do aplicativo

O [ContentRootPath](#) é o mesmo caminho fornecido para o argumento `binPath` quando o serviço é criado. Em vez de chamar [GetCurrentDirectory](#) para criar caminhos para arquivos de configuração, chame [SetCurrentDirectory](#) com o caminho para a raiz do conteúdo do aplicativo.

No `Program.Main`, determine o caminho para a pasta do executável do serviço e use o caminho para estabelecer a raiz do conteúdo do aplicativo:

```
var pathToExe = Process.GetCurrentProcess().MainModule.FileName;
var pathToContentRoot = Path.GetDirectoryName(pathToExe);
Directory.SetCurrentDirectory(pathToContentRoot);

CreateWebHostBuilder(args)
    .Build()
    .RunAsService();
```

### Armazene os arquivos do serviço em um local adequado no disco

Especifique um caminho absoluto com [SetBasePath](#) quando usar um [IConfigurationBuilder](#) para a pasta que contém os arquivos.

## Recursos adicionais

- [Configuração do ponto de extremidade Kestrel](#) (inclui a configuração HTTPS e suporte à SNI)
- [Host da Web do ASP.NET Core](#)
- [Solucionar problemas de projetos do ASP.NET Core](#)

# Host ASP.NET Core no Linux com Nginx

16/01/2019 • 24 minutes to read • [Edit Online](#)

Por Sourabh Shirhatti

Este guia explica como configurar um ambiente ASP.NET Core pronto para produção em um servidor Ubuntu 16.04. Essas instruções provavelmente funcionarão com versões mais recentes do Ubuntu, mas as instruções não foram testadas com versões mais recentes.

Para saber mais sobre outras distribuições do Linux compatíveis com o ASP.NET Core, veja [Pré-requisitos para o .NET Core no Linux](#).

## NOTE

Para Ubuntu 14.04, o *supervisord* é recomendado como uma solução para monitorar o processo do Kestrel. O *systemd* não está disponível no Ubuntu 14.04. Para obter instruções Ubuntu 14.04, veja a [versão anterior deste tópico](#).

Este guia:

- Coloca um aplicativo ASP.NET Core existente em um servidor proxy reverso.
- Configura o servidor proxy reverso para encaminhar solicitações ao servidor Web do Kestrel.
- Assegura que o aplicativo Web seja executado na inicialização como um daemon.
- Configura uma ferramenta de gerenciamento de processo para ajudar a reiniciar o aplicativo Web.

## Pré-requisitos

1. Acesso a um servidor Ubuntu 16.04 com uma conta de usuário padrão com privilégio sudo.
2. Instale o tempo de execução do .NET Core no servidor.
  - a. Acesse a [página Todos os Downloads do .NET Core](#).
  - b. Selecione o tempo de execução não de versão prévia mais recente da lista em **Tempo de Execução**.
  - c. Selecione e siga as instruções para Ubuntu que correspondem à versão Ubuntu do servidor.
3. Um aplicativo ASP.NET Core existente.

## Publicar e copiar o aplicativo

Configurar o aplicativo para um [implantação dependente de estrutura](#).

Execute `dotnet publish` do ambiente de desenvolvimento para empacotar um aplicativo em um diretório (por exemplo, `bin/Release/<target_framework_moniker>/publish`) que pode ser executado no servidor:

```
dotnet publish --configuration Release
```

O aplicativo também poderá ser publicado como uma [implantação autossuficiente](#) se você preferir não manter o tempo de execução do .NET Core no servidor.

Copie o aplicativo ASP.NET Core para o servidor usando uma ferramenta que se integre ao fluxo de trabalho da organização (por exemplo, SCP, SFTP). É comum para localizar os aplicativos Web no diretório `var` (por exemplo, `var/www/helloapp`).

#### NOTE

Em um cenário de implantação de produção, um fluxo de trabalho de integração contínua faz o trabalho de publicar o aplicativo e copiar os ativos para o servidor.

Teste o aplicativo:

1. Na linha de comando, execute o aplicativo: `dotnet <app_assembly>.dll`.
2. Em um navegador, vá para `http://<serveraddress>:<port>` para verificar se o aplicativo funciona no Linux localmente.

## Configurar um servidor proxy reverso

Um proxy reverso é uma configuração comum para atender a aplicativos Web dinâmicos. Um proxy reverso encerra a solicitação HTTP e a encaminha para o aplicativo ASP.NET Core.

### Usar um servidor proxy reverso

O Kestrel é excelente para servir conteúdo dinâmico do ASP.NET Core. No entanto, as funcionalidades de servidor Web não têm tantos recursos quanto servidores como IIS, Apache ou Nginx. Um servidor proxy reverso pode descarregar trabalho como servir conteúdo estático, armazenar solicitações em cache, compactar solicitações e terminar HTTPS do servidor HTTP. Um servidor proxy reverso pode residir em um computador dedicado ou pode ser implantado junto com um servidor HTTP.

Para os fins deste guia, uma única instância de Nginx é usada. Ela é executada no mesmo servidor, junto com o servidor HTTP. Com base nos requisitos, uma configuração diferente pode ser escolhida.

Uma vez que as solicitações são encaminhadas pelo proxy reverso, use o [Middleware de Cabeçalhos Encaminhados](#) do pacote [Microsoft.AspNetCore.HttpOverrides](#). O middleware atualiza o `Request.Scheme` usando o cabeçalho `X-Forwarded-Proto`, de forma que URLs de redirecionamento e outras políticas de segurança funcionam corretamente.

Qualquer componente que dependa do esquema, como autenticação, geração de link, redirecionamentos e localização geográfica, deverá ser colocado depois de invocar o Middleware de Cabeçalhos Encaminhados. Como regra geral, o Middleware de Cabeçalhos Encaminhados deve ser executado antes de outro middleware, exceto middleware de tratamento de erro e de diagnóstico. Essa ordenação garantirá que o middleware conte com informações de cabeçalhos encaminhadas que podem consumir os valores de cabeçalho para processamento.

Invoque o método `UseForwardedHeaders` em `Startup.Configure` antes de chamar `UseAuthentication` ou um middleware de esquema de autenticação semelhante. Configure o middleware para encaminhar os cabeçalhos `X-Forwarded-For` e `X-Forwarded-Proto`:

```
app.UseForwardedHeaders(new ForwardedHeadersOptions
{
    ForwardedHeaders = ForwardedHeaders.XForwardedFor | ForwardedHeaders.XForwardedProto
});

app.UseAuthentication();
```

Invoque o método `UseForwardedHeaders` em `Startup.Configure` antes de chamar `UseIdentity` e `UseFacebookAuthentication` ou um middleware de esquema de autenticação semelhante. Configure o middleware para encaminhar os cabeçalhos `X-Forwarded-For` e `X-Forwarded-Proto`:

```
app.UseForwardedHeaders(new ForwardedHeadersOptions
{
    ForwardedHeaders = ForwardedHeaders.XForwardedFor | ForwardedHeaders.XForwardedProto
});

app.UseIdentity();
app.UseFacebookAuthentication(new FacebookOptions()
{
    AppId = Configuration["Authentication:Facebook:AppId"],
    AppSecret = Configuration["Authentication:Facebook:AppSecret"]
});
```

Se nenhum [ForwardedHeadersOptions](#) for especificado para o middleware, os cabeçalhos padrão para encaminhar serão `None`.

Somente os proxies em execução no localhost (127.0.0.1, [:1]) são confiáveis por padrão. Se outros proxies ou redes confiáveis em que a organização trata solicitações entre a Internet e o servidor Web, adicione-os à lista de [KnownProxies](#) ou [KnownNetworks](#) com [ForwardedHeadersOptions](#). O exemplo a seguir adiciona um servidor proxy confiável no endereço IP 10.0.0.100 ao Middleware de cabeçalhos encaminhados `KnownProxies` em `Startup.ConfigureServices`:

```
services.Configure<ForwardedHeadersOptions>(options =>
{
    options.KnownProxies.Add(IPAddress.Parse("10.0.0.100"));
});
```

Para obter mais informações, consulte [Configure o ASP.NET Core para trabalhar com servidores proxy e平衡adores de carga](#).

## Instalar o Nginx

Use `apt-get` para instalar o Nginx. O instalador cria um script de inicialização `systemd` que executa o Nginx como daemon na inicialização do sistema. Siga as instruções de instalação do Ubuntu no [Nginx: pacotes Debian/Ubuntu oficiais](#).

### NOTE

Se módulos Nginx opcionais forem exigidos, poderá haver necessidade de criar o Nginx da origem.

Já que Nginx foi instalado pela primeira vez, inicie-o explicitamente executando:

```
sudo service nginx start
```

Verifique se um navegador exibe a página de aterrissagem padrão do Nginx. A página de aterrissagem é acessível em `http://<server_IP_address>/index.nginx-debian.html`.

## Configurar o Nginx

Para configurar o Nginx como um proxy inverso para encaminhar solicitações para o nosso aplicativo ASP.NET Core, modifique `/etc/nginx/sites-available/default`. Abra-o em um editor de texto arquivo e substitua o conteúdo pelo mostrado a seguir:

```

server {
    listen      80;
    server_name example.com *.example.com;
    location / {
        proxy_pass          http://localhost:5000;
        proxy_http_version 1.1;
        proxy_set_header   Upgrade $http_upgrade;
        proxy_set_header   Connection keep-alive;
        proxy_set_header   Host $host;
        proxy_cache_bypass $http_upgrade;
        proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header   X-Forwarded-Proto $scheme;
    }
}

```

Quando nenhum `server_name` corresponde, o Nginx usa o servidor padrão. Se nenhum servidor padrão é definido, o primeiro servidor no arquivo de configuração é o servidor padrão. Como prática recomendada, adicione um servidor padrão específico que retorna um código de status 444 no arquivo de configuração. Um exemplo de configuração de servidor padrão é:

```

server {
    listen 80 default_server;
    # listen [::]:80 default_server deferred;
    return 444;
}

```

Com o servidor padrão e o arquivo de configuração anterior, o Nginx aceita tráfego público na porta 80 com um cabeçalho de host `example.com` ou `*.example.com`. Solicitações que não correspondam a esses hosts não serão encaminhadas para o Kestrel. O Nginx encaminha as solicitações correspondentes para o Kestrel em `http://localhost:5000`. Veja [Como o nginx processa uma solicitação](#) para obter mais informações. Para alterar o IP/porta do Kestrel, veja [Kestrel: configuração do ponto de extremidade](#).

#### **WARNING**

Falta ao especificar uma diretiva `server_name` expõe seu aplicativo para vulnerabilidades de segurança. Associações de curva de subdomínio (por exemplo, `*.example.com`) não oferecerão esse risco de segurança se você controlar o domínio pai completo (em vez de `*.com`, o qual é vulnerável). Veja [rfc7230 section-5.4](#) para obter mais informações.

Quando a configuração Nginx é estabelecida, execute `sudo nginx -t` para verificar a sintaxe dos arquivos de configuração. Se o teste do arquivo de configuração for bem-sucedido, você poderá forçar o Nginx a acompanhar as alterações executando `sudo nginx -s reload`.

Para executar o aplicativo diretamente no servidor:

1. Navegue até o diretório do aplicativo.
2. Execute o aplicativo: `dotnet <app_assembly.dll>`, em que `app_assembly.dll` é o nome do arquivo do assembly do aplicativo.

Se o aplicativo for executado no servidor, mas não responder pela Internet, verifique o firewall do servidor e confirme que a porta 80 está aberta. Se você estiver usando uma VM do Azure Ubuntu, adicione uma regra NSG (Grupo de Segurança de Rede) que permite tráfego de entrada na porta 80. Não é necessário habilitar uma regra de saída da porta 80, uma vez que o tráfego de saída é concedido automaticamente quando a regra de entrada é habilitada.

Quando terminar de testar o aplicativo, encerre o aplicativo com `Ctrl+C` no prompt de comando.

# Monitorar o aplicativo

O servidor agora está configurado para encaminhar solicitações feitas a `http://<serveraddress>:80` ao aplicativo ASP.NET Core em execução no Kestrel em `http://127.0.0.1:5000`. No entanto, o Nginx não está configurado para gerenciar o processo do Kestrel. É possível usar o `systemd` para criar um arquivo de serviço para iniciar e monitorar o aplicativo Web subjacente. `systemd` é um sistema de inicialização que fornece muitos recursos poderosos para iniciar, parar e gerenciar processos.

## Criar o arquivo de serviço

Crie o arquivo de definição de serviço:

```
sudo nano /etc/systemd/system/kestrel-helloapp.service
```

A seguir, um exemplo de arquivo de serviço para o aplicativo:

```
[Unit]
Description=Example .NET Web API App running on Ubuntu

[Service]
WorkingDirectory=/var/www/helloapp
ExecStart=/usr/bin/dotnet /var/www/helloapp/helloapp.dll
Restart=always
# Restart service after 10 seconds if the dotnet service crashes:
RestartSec=10
KillSignal=SIGINT
SyslogIdentifier=dotnet-example
User=www-data
Environment=ASPNETCORE_ENVIRONMENT=Production
Environment=DOTNET_PRINT_TELEMETRY_MESSAGE=false

[Install]
WantedBy=multi-user.target
```

Se o usuário `www-data` não é usado pela configuração, o usuário definido aqui deve ser criado primeiro e a propriedade adequada dos arquivos deve ser concedida a ele.

Use `TimeoutStopSec` para configurar a duração do tempo de espera para o aplicativo desligar depois de receber o sinal de interrupção inicial. Se o aplicativo não desligar nesse período, o SIGKILL será emitido para encerrá-lo. Forneça o valor como segundos sem unidade (por exemplo, `150`), um valor de duração (por exemplo, `2min 30s`) ou `infinity` para desabilitar o tempo limite. `TimeoutStopSec` é revertido para o valor padrão de `DefaultTimeoutStopSec` no arquivo de configuração do gerenciador (`systemd-system.conf`, `system.conf.d`, `systemd-user.conf` e `user.conf.d`). O tempo limite padrão para a maioria das distribuições é de 90 segundos.

```
# The default value is 90 seconds for most distributions.
TimeoutStopSec=90
```

O Linux tem um sistema de arquivos que diferencia maiúsculas de minúsculas. Definir `ASPNETCORE_ENVIRONMENT` para "Production" resulta em uma pesquisa pelo arquivo de configuração `appsettings.Production.json`, e não `appsettings.production.json`.

Alguns valores (por exemplo, cadeias de conexão de SQL) devem ser escapadas para que os provedores de configuração leiam as variáveis de ambiente. Use o seguinte comando para gerar um valor corretamente com caracteres de escape para uso no arquivo de configuração:

```
systemd-escape "<value-to-escape>"
```

Salve o arquivo e habilite o serviço.

```
sudo systemctl enable kestrel-helloapp.service
```

Inicie o serviço e verifique se ele está em execução.

```
sudo systemctl start kestrel-helloapp.service
sudo systemctl status kestrel-helloapp.service

● kestrel-helloapp.service - Example .NET Web API App running on Ubuntu
    Loaded: loaded (/etc/systemd/system/kestrel-helloapp.service; enabled)
    Active: active (running) since Thu 2016-10-18 04:09:35 NZDT; 35s ago
      Main PID: 9021 (dotnet)
        CGroup: /system.slice/kestrel-helloapp.service
                  └─9021 /usr/local/bin/dotnet /var/www/helloapp/helloapp.dll
```

Com o proxy reverso configurado e o Kestrel gerenciado por meio de systemd, o aplicativo Web está totalmente configurado e pode ser acessado em um navegador no computador local em `http://localhost`. Ele também é acessível por meio de um computador remoto, bloqueando qualquer firewall que o possa estar bloqueando. Inspecionando os cabeçalhos de resposta, o cabeçalho `Server` mostra o aplicativo ASP.NET Core sendo servido pelo Kestrel.

```
HTTP/1.1 200 OK
Date: Tue, 11 Oct 2016 16:22:23 GMT
Server: Kestrel
Keep-Alive: timeout=5, max=98
Connection: Keep-Alive
Transfer-Encoding: chunked
```

## Exibir logs

Já que o aplicativo Web usando Kestrel é gerenciado usando `systemd`, todos os eventos e os processos são registrados em um diário centralizado. No entanto, esse diário contém todas as entradas para todos os serviços e processos gerenciados pelo `systemd`. Para exibir os itens específicos de `kestrel-helloapp.service`, use o seguinte comando:

```
sudo journalctl -fu kestrel-helloapp.service
```

Para obter mais filtragem, opções de hora como `--since today`, `--until 1 hour ago` ou uma combinação delas, pode reduzir a quantidade de entradas retornadas.

```
sudo journalctl -fu kestrel-helloapp.service --since "2016-10-18" --until "2016-10-18 04:00"
```

## Proteção de dados

A [pilha de proteção de dados do ASP.NET Core](#) é usada por vários [middlewares](#) do ASP.NET Core, incluindo o middleware de autenticação (por exemplo, o middleware de cookie) e as proteções de CSRF (solicitação intersite forjada). Mesmo se as APIs de Proteção de Dados não forem chamadas pelo código do usuário, a proteção de dados deverá ser configurada para criar um [repositório de chaves](#) criptográfico persistente. Se a proteção de dados não estiver configurada, as chaves serão mantidas na memória e descartadas quando o aplicativo for reiniciado.

Se o token de autenticação for armazenado na memória quando o aplicativo for reiniciado:

- Todos os tokens de autenticação baseados em cookies serão invalidados.
- Os usuários precisam entrar novamente na próxima solicitação deles.
- Todos os dados protegidos com o token de autenticação não poderão mais ser descriptografados. Isso pode incluir os [tokens CSRF](#) e [cookies TempData do MVC do ASP.NET Core](#).

Para configurar a proteção de dados de modo que ela mantenha e criptografe o token de autenticação, consulte:

- [Provedores de armazenamento de chaves no ASP.NET Core](#)
- [Chave de criptografia em repouso no ASP.NET Core](#)

## Campos de cabeçalho da solicitação muito grandes

Se o aplicativo exigir campos de cabeçalho de solicitação mais longos que o permitido pelas configurações padrão do servidor proxy (normalmente de 4K ou 8K dependendo da plataforma), as seguintes diretivas precisarão de ajustes. Os valores que serão aplicados são dependentes de cenário. Para obter mais informações, confira a documentação do servidor.

- [proxy\\_buffer\\_size](#)
- [proxy\\_buffers](#)
- [proxy\\_busy\\_buffers\\_size](#)
- [large\\_client\\_header\\_buffers](#)

### **WARNING**

Não aumente os valores padrão de buffers de proxy a menos que necessário. Aumentar esses valores aumenta o risco de estouro de buffer (estouro) e ataques de DoS (negação de serviço) por usuários mal-intencionados.

## Proteger o aplicativo

### Habilitar AppArmor

O LSM (Módulos de Segurança do Linux) é uma estrutura que é parte do kernel do Linux desde o Linux 2.6. O LSM dá suporte a diferentes implementações de módulos de segurança. O [AppArmor](#) é um LSM que implementa um sistema de controle de acesso obrigatório que permite restringir o programa a um conjunto limitado de recursos. Verifique se o AppArmor está habilitado e configurado corretamente.

### Configurar o firewall

Feche todas as portas externas que não estão em uso. Firewall descomplicado (ufw) fornece um front-end para [iptables](#) fornecendo uma interface de linha de comando para configurar o firewall.

### **WARNING**

Um firewall impedirá o acesso a todo o sistema se não ele estiver configurado corretamente. Se houver falha ao especificar a porta SSH correta, você será bloqueado do sistema se estiver usando o SSH para se conectar a ele. A porta padrão é a 22. Para obter mais informações, consulte a [introdução ao ufw e o manual](#).

Instale o [ufw](#) e configure-o para permitir o tráfego em todas as portas necessárias.

```
sudo apt-get install ufw

sudo ufw allow 22/tcp
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp

sudo ufw enable
```

## Proteger o Nginx

### Alterar o nome da resposta do Nginx

Edite `src/http/ngx_http_header_filter_module.c`:

```
static char ngx_http_server_string[] = "Server: Web Server" CRLF;
static char ngx_http_server_full_string[] = "Server: Web Server" CRLF;
```

### Configurar opções

Configure o servidor com os módulos adicionais necessários. Considere usar um firewall de aplicativo Web como [ModSecurity](#) para fortalecer o aplicativo.

### Configuração de HTTPS

- Configure o servidor para escutar tráfego HTTPS na porta `443` especificando um certificado válido emitido por uma AC (autoridade de certificação) confiável.
- Aprimore a segurança, empregando algumas das práticas descritas no arquivo `/etc/nginx/nginx.conf` a seguir. Exemplos incluem a escolha de uma criptografia mais forte e o redirecionamento de todo o tráfego por meio de HTTP para HTTPS.
- A adição de um cabeçalho `HTTP Strict-Transport-Security` (HSTS) garante que todas as próximas solicitações feitas pelo cliente sejam por HTTPS.
- Não adicione o cabeçalho HSTS ou escolha um `max-age` apropriado, se quiser desabilitar o HTTPS futuramente.

Adicione o arquivo de configuração `/etc/nginx/proxy.conf`:

```
proxy_redirect      off;
proxy_set_header    Host      $host;
proxy_set_header    X-Real-IP   $remote_addr;
proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header    X-Forwarded-Proto $scheme;
client_max_body_size 10m;
client_body_buffer_size 128k;
proxy_connect_timeout 90;
proxy_send_timeout 90;
proxy_read_timeout 90;
proxy_buffers 32 4k;
```

Edite o arquivo de configuração `/etc/nginx/nginx.conf`. O exemplo contém ambas as seções `http` e `server` em um arquivo de configuração.

```

http {
    include /etc/nginx/proxy.conf;
    limit_req_zone $binary_remote_addr zone=one:10m rate=5r/s;
    server_tokens off;

    sendfile on;
    keepalive_timeout 29; # Adjust to the lowest possible value that makes sense for your use case.
    client_body_timeout 10; client_header_timeout 10; send_timeout 10;

    upstream hellomvc{
        server localhost:5000;
    }

    server {
        listen *:80;
        add_header Strict-Transport-Security max-age=15768000;
        return 301 https://$host$request_uri;
    }

    server {
        listen *:443 ssl;
        server_name example.com;
        ssl_certificate /etc/ssl/certs/testCert.crt;
        ssl_certificate_key /etc/ssl/certs/testCert.key;
        ssl_protocols TLSv1.1 TLSv1.2;
        ssl_prefer_server_ciphers on;
        ssl_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH";
        ssl_ecdh_curve secp384r1;
        ssl_session_cache shared:SSL:10m;
        ssl_session_tickets off;
        ssl_stapling on; #ensure your cert is capable
        ssl_stapling_verify on; #ensure your cert is capable

        add_header Strict-Transport-Security "max-age=63072000; includeSubdomains; preload";
        add_header X-Frame-Options DENY;
        add_header X-Content-Type-Options nosniff;

        #Redirects all traffic
        location / {
            proxy_pass http://hellomvc;
            limit_req zone=one burst=10 nodelay;
        }
    }
}

```

### Proteger o Nginx de clickjacking

[Clickjacking](#), também conhecido como um *ataque por inferência na interface do usuário*, é um ataque mal-intencionado em que o visitante do site é levado a clicar em um link ou botão em uma página diferente daquela que está visitando atualmente. Use `X-FRAME-OPTIONS` para proteger o site.

Para atenuar ataques de clickjacking:

1. Edite o arquivo `nginx.conf`:

```
sudo nano /etc/nginx/nginx.conf
```

Adicione a linha `add_header X-Frame-Options "SAMEORIGIN";`.

2. Salve o arquivo.
3. Reinicie o Nginx.

### Detecção de tipo MIME

Esse cabeçalho evita que a maioria dos navegadores faça detecção MIME de uma resposta distante do tipo de conteúdo declarado, visto que o cabeçalho instrui o navegador para não substituir o tipo de conteúdo de resposta. Com a opção `nosniff`, se o servidor informa que o conteúdo é "text/html", o navegador renderiza-a como "text/html".

Edite o arquivo `nginx.conf`.

```
sudo nano /etc/nginx/nginx.conf
```

Adicione a linha `add_header X-Content-Type-Options "nosniff";` e salve o arquivo, depois reinicie o Nginx.

## Recursos adicionais

- [Pré-requisitos para o .NET Core no Linux](#)
- [Nginx: versões binárias: pacotes Debian/Ubuntu oficiais](#)
- [Solucionar problemas de projetos do ASP.NET Core](#)
- [Configure o ASP.NET Core para trabalhar com servidores proxy e平衡adores de carga](#)
- [NGINX: usando o cabeçalho Encaminhado](#)

# Hospedar o ASP.NET Core no Linux com o Apache

16/01/2019 • 22 minutes to read • [Edit Online](#)

Por Shayne Boyer

Usando este guia, saiba como configurar o [Apache](#) como um servidor proxy reverso no [CentOS 7](#) para redirecionar o tráfego HTTP para um aplicativo Web ASP.NET Core em execução no servidor [Kestrel](#). A extensão [mod\\_proxy](#) e os módulos relacionados criam o proxy reverso do servidor.

## Pré-requisitos

- Servidor que executa o CentOS 7 com uma conta de usuário padrão com privilégio sudo.
- Instale o tempo de execução do .NET Core no servidor.
  1. Acesse a [página Todos os Downloads do .NET Core](#).
  2. Selecione o tempo de execução não de versão prévia mais recente da lista em **Tempo de Execução**.
  3. Selecione e siga as instruções para CentOS/Oracle.
- Um aplicativo ASP.NET Core existente.

## Publicar e copiar o aplicativo

Configurar o aplicativo para um [implantação dependente de estrutura](#).

Execute [dotnet publish](#) do ambiente de desenvolvimento para empacotar um aplicativo em um diretório (por exemplo, `bin/Release/<target_framework_moniker>/publish`) que pode ser executado no servidor:

```
dotnet publish --configuration Release
```

O aplicativo também poderá ser publicado como uma [implantação autossuficiente](#) se você preferir não manter o tempo de execução do .NET Core no servidor.

Copie o aplicativo ASP.NET Core para o servidor usando uma ferramenta que se integre ao fluxo de trabalho da organização (por exemplo, SCP, SFTP). É comum para localizar os aplicativos Web no diretório `var` (por exemplo, `var/www/helloapp`).

### NOTE

Em um cenário de implantação de produção, um fluxo de trabalho de integração contínua faz o trabalho de publicar o aplicativo e copiar os ativos para o servidor.

## Configurar um servidor proxy

Um proxy reverso é uma configuração comum para atender a aplicativos Web dinâmicos. O proxy reverso encerra a solicitação HTTP e a encaminha para o aplicativo ASP.NET.

Um servidor proxy é aquele que encaminha as solicitações de cliente para outro servidor, em vez de atendê-las por conta própria. Um proxy reverso encaminha para um destino fixo, normalmente, em nome de clientes arbitrários. Neste guia, o Apache é configurado como o proxy reverso em execução no mesmo servidor em que o Kestrel está servindo o aplicativo ASP.NET Core.

Uma vez que as solicitações são encaminhadas pelo proxy reverso, use o [Middleware de Cabeçalhos](#)

Encaminhados do pacote [Microsoft.AspNetCore.HttpOverrides](#). O middleware atualiza o `Request.Scheme` usando o cabeçalho `X-Forwarded-Proto`, de forma que URLs de redirecionamento e outras políticas de segurança funcionam corretamente.

Qualquer componente que dependa do esquema, como autenticação, geração de link, redirecionamentos e localização geográfica, deverá ser colocado depois de invocar o Middleware de Cabeçalhos Encaminhados. Como regra geral, o Middleware de Cabeçalhos Encaminhados deve ser executado antes de outro middleware, exceto middleware de tratamento de erro e de diagnóstico. Essa ordenação garantirá que o middleware conte com informações de cabeçalhos encaminhadas que podem consumir os valores de cabeçalho para processamento.

Invoque o método `UseForwardedHeaders` em `Startup.Configure` antes de chamar `UseAuthentication` ou um middleware de esquema de autenticação semelhante. Configure o middleware para encaminhar os cabeçalhos `X-Forwarded-For` e `X-Forwarded-Proto`:

```
app.UseForwardedHeaders(new ForwardedHeadersOptions
{
    ForwardedHeaders = ForwardedHeaders.XForwardedFor | ForwardedHeaders.XForwardedProto
});

app.UseAuthentication();
```

Invoque o método `UseForwardedHeaders` em `Startup.Configure` antes de chamar `UseIdentity` e `UseFacebookAuthentication` ou um middleware de esquema de autenticação semelhante. Configure o middleware para encaminhar os cabeçalhos `X-Forwarded-For` e `X-Forwarded-Proto`:

```
app.UseForwardedHeaders(new ForwardedHeadersOptions
{
    ForwardedHeaders = ForwardedHeaders.XForwardedFor | ForwardedHeaders.XForwardedProto
});

app.UseIdentity();
app.UseFacebookAuthentication(new FacebookOptions()
{
    AppId = Configuration["Authentication:Facebook:AppId"],
    AppSecret = Configuration["Authentication:Facebook:AppSecret"]
});
```

Se nenhum `ForwardedHeadersOptions` for especificado para o middleware, os cabeçalhos padrão para encaminhar serão `None`.

Somente os proxies em execução no localhost (127.0.0.1, [:1]) são confiáveis por padrão. Se outros proxies ou redes confiáveis em que a organização trata solicitações entre a Internet e o servidor Web, adicione-os à lista de `KnownProxies` ou `KnownNetworks` com `ForwardedHeadersOptions`. O exemplo a seguir adiciona um servidor proxy confiável no endereço IP 10.0.0.100 ao Middleware de cabeçalhos encaminhados `KnownProxies` em `Startup.ConfigureServices`:

```
services.Configure<ForwardedHeadersOptions>(options =>
{
    options.KnownProxies.Add(IPAddress.Parse("10.0.0.100"));
});
```

Para obter mais informações, consulte [Configure o ASP.NET Core para trabalhar com servidores proxy e平衡adores de carga](#).

## Instalar o Apache

Atualize os pacotes CentOS para as respectivas versões estáveis mais recentes:

```
sudo yum update -y
```

Instale o servidor Web do Apache no CentOS com um único comando `yum`:

```
sudo yum -y install httpd mod_ssl
```

Saída de exemplo depois de executar o comando:

```
Downloading packages:
httpd-2.4.6-40.el7.centos.4.x86_64.rpm | 2.7 MB 00:00:01
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Installing : httpd-2.4.6-40.el7.centos.4.x86_64      1/1
Verifying  : httpd-2.4.6-40.el7.centos.4.x86_64      1/1

Installed:
httpd.x86_64 0:2.4.6-40.el7.centos.4

Complete!
```

#### NOTE

Neste exemplo, o resultado reflete `httpd.86_64`, pois a versão do CentOS 7 é de 64 bits. Para verificar o local em que o Apache está instalado, execute `whereis httpd` em um prompt de comando.

## Configurar o Apache

Os arquivos de configuração do Apache estão localizados no diretório `/etc/httpd/conf.d/`. Qualquer arquivo com a extensão `.conf` é processado em ordem alfabética, além dos arquivos de configuração do módulo em `/etc/httpd/conf.modules.d/`, que contém todos os arquivos de configuração necessários para carregar os módulos.

Crie um arquivo de configuração chamado `helloapp.conf` para o aplicativo:

```
<VirtualHost *:*
```

O bloco `VirtualHost` pode aparecer várias vezes, em um ou mais arquivos em um servidor. No arquivo de configuração anterior, o Apache aceita tráfego público na porta 80. O domínio `www.example.com` está sendo atendido e o alias `*.example.com` é resolvido para o mesmo site. Veja [Suporte a host virtual baseado em nome](#) para obter mais informações. As solicitações passadas por proxy na raiz para a porta 5000 do servidor em

127.0.0.1. Para a comunicação bidirecional, `ProxyPass` e `ProxyPassReverse` são necessários. Para alterar o IP/porta do Kestrel, veja [Kestrel: configuração do ponto de extremidade](#).

#### WARNING

Falha ao especificar uma diretiva `ServerName` no bloco `VirtualHost` expõe seu aplicativo para vulnerabilidades de segurança. Associações de curva de subdomínio (por exemplo, `*.example.com`) não oferecerão esse risco de segurança se você controlar o domínio pai completo (em vez de `*.com`, o qual é vulnerável). Veja [rfc7230 section-5.4](#) para obter mais informações.

O registro em log pode ser configurado por `VirtualHost` usando diretivas `ErrorLog` e `CustomLog`. `ErrorLog` é o local em que o servidor registrará em log os erros, enquanto `CustomLog` define o nome de arquivo e o formato do arquivo de log. Nesse caso, esse é o local em que as informações de solicitação são registradas em log. Há uma linha para cada solicitação.

Salve o arquivo e teste a configuração. Se tudo passar, a resposta deverá ser `Syntax [OK]`.

```
sudo service httpd configtest
```

Reinic peace o Apache:

```
sudo systemctl restart httpd
sudo systemctl enable httpd
```

## Monitorar o aplicativo

O Apache agora está configurado para encaminhar solicitações feitas a `http://localhost:80` para o aplicativo ASP.NET Core em execução no Kestrel em `http://127.0.0.1:5000`. No entanto, o Apache não está configurado para gerenciar o processo do Kestrel. Use `systemd` e crie um arquivo de serviço para iniciar e monitorar o aplicativo Web subjacente. `systemd` é um sistema de inicialização que fornece muitos recursos poderosos para iniciar, parar e gerenciar processos.

#### Criar o arquivo de serviço

Crie o arquivo de definição de serviço:

```
sudo nano /etc/systemd/system/kestrel-helloapp.service
```

Eis um exemplo de arquivo de serviço para o aplicativo:

```

[Unit]
Description=Example .NET Web API App running on CentOS 7

[Service]
WorkingDirectory=/var/www/helloapp
ExecStart=/usr/local/bin/dotnet /var/www/helloapp/helloapp.dll
Restart=always
# Restart service after 10 seconds if the dotnet service crashes:
RestartSec=10
KillSignal=SIGINT
SyslogIdentifier=dotnet-example
User=apache
Environment=ASPNETCORE_ENVIRONMENT=Production

[Install]
WantedBy=multi-user.target

```

Se o usuário `apache` não for usado pela configuração, o usuário precisará ser criado primeiro e a propriedade adequada dos arquivos precisará ser concedida a ele.

Use `TimeoutStopSec` para configurar a duração do tempo de espera para o aplicativo desligar depois de receber o sinal de interrupção inicial. Se o aplicativo não desligar nesse período, o SIGKILL será emitido para encerrá-lo. Forneça o valor como segundos sem unidade (por exemplo, `150`), um valor de duração (por exemplo, `2min 30s`) ou `infinity` para desabilitar o tempo limite. `TimeoutStopSec` é revertido para o valor padrão de `DefaultTimeoutStopSec` no arquivo de configuração do gerenciador (`systemd-system.conf`, `system.conf.d`, `systemd-user.conf` e `user.conf.d`). O tempo limite padrão para a maioria das distribuições é de 90 segundos.

```

# The default value is 90 seconds for most distributions.
TimeoutStopSec=90

```

Alguns valores (por exemplo, cadeias de conexão de SQL) devem ser escapadas para que os provedores de configuração leiam as variáveis de ambiente. Use o seguinte comando para gerar um valor corretamente com caracteres de escape para uso no arquivo de configuração:

```
systemd-escape "<value-to-escape>"
```

Salve o arquivo e habilite o serviço:

```
sudo systemctl enable kestrel-helloapp.service
```

Inicie o serviço e verifique se ele está em execução:

```

sudo systemctl start kestrel-helloapp.service
sudo systemctl status kestrel-helloapp.service

● kestrel-helloapp.service - Example .NET Web API App running on CentOS 7
   Loaded: loaded (/etc/systemd/system/kestrel-helloapp.service; enabled)
   Active: active (running) since Thu 2016-10-18 04:09:35 NZDT; 35s ago
     Main PID: 9021 (dotnet)
        CGroup: /system.slice/kestrel-helloapp.service
                  └─9021 /usr/local/bin/dotnet /var/www/helloapp/helloapp.dll

```

Com o proxy reverso configurado e o Kestrel gerenciado por meio de `systemd`, o aplicativo Web está totalmente configurado e pode ser acessado em um navegador no computador local em `http://localhost`. Inspecionando os cabeçalhos de resposta, o cabeçalho **Server** indica que o aplicativo ASP.NET Core é servido pelo Kestrel:

```
HTTP/1.1 200 OK
Date: Tue, 11 Oct 2016 16:22:23 GMT
Server: Kestrel
Keep-Alive: timeout=5, max=98
Connection: Keep-Alive
Transfer-Encoding: chunked
```

## Exibir logs

Já que o aplicativo Web usando Kestrel é gerenciado usando *systemd*, os eventos e os processos são registrados em um diário centralizado. No entanto, esse diário contém todas as entradas para todos os serviços e processos gerenciados por *systemd*. Para exibir os itens específicos de `kestrel-helloapp.service`, use o seguinte comando:

```
sudo journalctl -fu kestrel-helloapp.service
```

Para filtragem de hora, especifique opções de tempo com o comando. Por exemplo, use `--since today` para filtrar o dia atual ou `--until 1 hour ago` para ver as entradas da hora anterior. Para obter mais informações, confira a [página de manual para journalctl](#).

```
sudo journalctl -fu kestrel-helloapp.service --since "2016-10-18" --until "2016-10-18 04:00"
```

## Proteção de dados

A [pilha de proteção de dados do ASP.NET Core](#) é usada por vários [middlewares](#) do ASP.NET Core, incluindo o middleware de autenticação (por exemplo, o middleware de cookie) e as proteções de CSRF (solicitação intersite forjada). Mesmo se as APIs de Proteção de Dados não forem chamadas pelo código do usuário, a proteção de dados deverá ser configurada para criar um [repositório de chaves](#) criptográfico persistente. Se a proteção de dados não estiver configurada, as chaves serão mantidas na memória e descartadas quando o aplicativo for reiniciado.

Se o token de autenticação for armazenado na memória quando o aplicativo for reiniciado:

- Todos os tokens de autenticação baseados em cookies serão invalidados.
- Os usuários precisam entrar novamente na próxima solicitação deles.
- Todos os dados protegidos com o token de autenticação não poderão mais ser descriptografados. Isso pode incluir os [tokens CSRF](#) e [cookies TempData do MVC do ASP.NET Core](#).

Para configurar a proteção de dados de modo que ela mantenha e criptografe o token de autenticação, consulte:

- [Provedores de armazenamento de chaves no ASP.NET Core](#)
- [Chave de criptografia em repouso no ASP.NET Core](#)

## Proteger o aplicativo

### Configurar o firewall

*Firewalld* é um daemon dinâmico para gerenciar o firewall com suporte para zonas de rede. Portas e filtragem de pacote ainda podem ser gerenciados pelo iptables. *Firewalld* deve ser instalado por padrão. `yum` pode ser usado para instalar o pacote ou verificar se ele está instalado.

```
sudo yum install firewalld -y
```

Use `firewalld` para abrir somente as portas necessárias para o aplicativo. Nesse caso, as portas 80 e 443 são usadas. Os comandos a seguir definem permanentemente as portas 80 e 443 para estarem abertas:

```
sudo firewall-cmd --add-port=80/tcp --permanent  
sudo firewall-cmd --add-port=443/tcp --permanent
```

Recarregue as configurações de firewall. Verifique os serviços e as portas disponíveis na zona padrão. As opções estão disponíveis por meio da inspeção de `firewall-cmd -h`.

```
sudo firewall-cmd --reload  
sudo firewall-cmd --list-all
```

```
public (default, active)  
interfaces: eth0  
sources:  
services: dhcipv6-client  
ports: 443/tcp 80/tcp  
masquerade: no  
forward-ports:  
icmp-blocks:  
rich rules:
```

## Configuração de HTTPS

Para configurar o Apache para HTTPS, o módulo `mod_ssl` é usado. Quando o módulo `httpd` foi instalado, o módulo `mod_ssl` também foi instalado. Se ele não foi instalado, use `yum` para adicioná-lo à configuração.

```
sudo yum install mod_ssl
```

Para impor o HTTPS, instale o módulo `mod_rewrite` para habilitar a regravação de URL:

```
sudo yum install mod_rewrite
```

Modifique o arquivo `helloapp.conf` para habilitar a regravação de URL e proteger a comunicação na porta 443:

```
<VirtualHost *:>
    RequestHeader set "X-Forwarded-Proto" expr=%{REQUEST_SCHEME}
</VirtualHost>

<VirtualHost *:80>
    RewriteEngine On
    RewriteCond %{HTTPS} !=on
    RewriteRule ^/?(.*) https:// %{SERVER_NAME}/$1 [R,L]
</VirtualHost>

<VirtualHost *:443>
    ProxyPreserveHost On
    ProxyPass / http://127.0.0.1:5000/
    ProxyPassReverse / http://127.0.0.1:5000/
    ErrorLog /var/log/httpd/helloapp-error.log
    CustomLog /var/log/httpd/helloapp-access.log common
    SSLEngine on
    SSLProtocol all -SSLv2
    SSLCipherSuite ALL:!ADH:!EXPORT:!SSLv2:!RC4+RSA:+HIGH:+MEDIUM:!LOW:!RC4
    SSLCertificateFile /etc/pki/tls/certs/localhost.crt
    SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
</VirtualHost>
```

#### NOTE

Este exemplo usa um certificado gerado localmente. **SSLCertificateFile** deve ser o arquivo de certificado primário para o nome de domínio. **SSLCertificateKeyFile** deve ser o arquivo de chave gerado quando o CSR é criado.

**SSLCertificateChainFile** deve ser o arquivo de certificado intermediário (se houver) fornecido pela autoridade de certificação.

Salve o arquivo e teste a configuração:

```
sudo service httpd configtest
```

Reinic peace o Apache:

```
sudo systemctl restart httpd
```

## Sugestões adicionais do Apache

### Cabeçalhos adicionais

Para proteção contra ataques mal-intencionados, há alguns cabeçalhos que devem ser modificados ou adicionados. Verifique se o módulo `mod_headers` está instalado:

```
sudo yum install mod_headers
```

### Proteger o Apache contra ataques de clickjacking

[Clickjacking](#), também conhecido como um *ataque por inferência na interface do usuário*, é um ataque mal-intencionado em que o visitante do site é levado a clicar em um link ou botão em uma página diferente daquela que está visitando atualmente. Use `X-FRAME-OPTIONS` para proteger o site.

Para atenuar ataques de clickjacking:

1. Edite o arquivo `httpd.conf`:

```
sudo nano /etc/httpd/conf/httpd.conf
```

Adicione a linha `Header append X-FRAME-OPTIONS "SAMEORIGIN"`.

2. Salve o arquivo.

3. Reinicie o Apache.

#### Detecção de tipo MIME

O cabeçalho `X-Content-Type-Options` impedirá o Internet Explorer de *farejar por MIME* (determinar o `Content-Type` de um arquivo com base no conteúdo do arquivo). Se o servidor define o cabeçalho `Content-Type` para `text/html` com a opção `nosniff` definida, o Internet Explorer renderiza o conteúdo como `text/html`, independentemente do conteúdo do arquivo.

Edite o arquivo `httpd.conf`:

```
sudo nano /etc/httpd/conf/httpd.conf
```

Adicione a linha `Header set X-Content-Type-Options "nosniff"`. Salve o arquivo. Reinicie o Apache.

#### Balanceamento de carga

Este exemplo mostra como instalar e configurar o Apache no CentOS 7 e no Kestrel no mesmo computador da instância. Para não ter um ponto único de falha, o uso de `mod_proxy_balancer` e a modificação do **VirtualHost** permitiriam o gerenciamento de várias instâncias dos aplicativos Web protegidos pelo servidor proxy do Apache.

```
sudo yum install mod_proxy_balancer
```

No arquivo de configuração mostrado abaixo, uma instância adicional do `helloapp` é configurada para ser executada na porta 5001. A seção `Proxy` é definida com uma configuração de平衡ador com dois membros para balancear carga de `byrequests`.

```

<VirtualHost *:>
    RequestHeader set "X-Forwarded-Proto" expr=%{REQUEST_SCHEME}
</VirtualHost>

<VirtualHost *:80>
    RewriteEngine On
    RewriteCond %{HTTPS} !=on
    RewriteRule ^/?(.*) https://{$SERVER_NAME}/$1 [R,L]
</VirtualHost>

<VirtualHost *:443>
    ProxyPass / balancer://mycluster/

    ProxyPassReverse / http://127.0.0.1:5000/
    ProxyPassReverse / http://127.0.0.1:5001/

    <Proxy balancer://mycluster>
        BalancerMember http://127.0.0.1:5000
        BalancerMember http://127.0.0.1:5001
        ProxySet lbmethod=byrequests
    </Proxy>

    <Location />
        SetHandler balancer
    </Location>
    ErrorLog /var/log/httpd/helloapp-error.log
    CustomLog /var/log/httpd/helloapp-access.log common
    SSLEngine on
    SSLProtocol all -SSLv2
    SSLCipherSuite ALL:!ADH:!EXPORT:!SSLv2:!RC4+RSA:+HIGH:+MEDIUM:!LOW:!RC4
    SSLCertificateFile /etc/pki/tls/certs/localhost.crt
    SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
</VirtualHost>

```

## Límites de taxa

Usando `mod_ratelimit`, que está incluído no módulo `httpd`, a largura de banda de clientes pode ser limitada:

```
sudo nano /etc/httpd/conf.d/ratelimit.conf
```

O arquivo de exemplo limita a largura de banda a 600 KB/s no local raiz:

```

<IfModule mod_ratelimit.c>
    <Location />
        SetOutputFilter RATE_LIMIT
        SetEnv rate-limit 600
    </Location>
</IfModule>

```

## Campos de cabeçalho da solicitação muito grandes

Se o aplicativo exigir campos de cabeçalho de solicitação maiores do que o permitido pela configuração padrão do servidor proxy (normalmente 8.190 bytes), ajuste o valor da diretiva `LimitRequestFieldSize`. O valor que será aplicada é dependentes de cenário. Para obter mais informações, confira a documentação do servidor.

### WARNING

Não aumente o valor padrão de `LimitRequestFieldSize` a menos que necessário. Aumentar esse valor aumenta o risco de estouro de buffer (estouro) e ataques de DoS (negação de serviço) por usuários mal-intencionados.

## Recursos adicionais

- [Pré-requisitos para o .NET Core no Linux](#)
- [Solucionar problemas de projetos do ASP.NET Core](#)
- [Configure o ASP.NET Core para trabalhar com servidores proxy e平衡adores de carga](#)

# Hospedar o ASP.NET Core em contêineres do Docker

21/06/2018 • 2 minutes to read • [Edit Online](#)

Os artigos a seguir estão disponíveis para saber mais sobre como hospedar aplicativos ASP.NET Core no Docker:

## [Introdução aos contêineres e ao Docker](#)

Veja como o uso de contêineres é uma abordagem de desenvolvimento de software na qual um aplicativo ou serviço, suas dependências e sua configuração são empacotados juntos como uma imagem de contêiner. A imagem pode ser testada e, em seguida, implantada em um host.

## [O que é o Docker?](#)

Descubra como o Docker é um projeto de software livre para automatizar a implantação de aplicativos como contêineres autossuficientes portáteis que podem ser executados na nuvem ou localmente.

## [Terminologia do Docker](#)

Aprenda termos e definições para a tecnologia do Docker.

## [Registros, imagens e contêineres do Docker](#)

Descubra como imagens de contêiner do Docker são armazenadas em um registro de imagem para implantação consistente entre ambientes.

## [Compilar imagens do Docker para Aplicativos .NET Core](#)

Saiba como criar um aplicativo ASP.NET Core e colocá-lo no Docker. Explore imagens do Docker mantidas pela Microsoft e examine os casos de uso.

## [Ferramentas do Visual Studio para Docker](#)

Descubra como o Visual Studio 2017 permite a criação, depuração e execução de aplicativos ASP.NET Core direcionados ao .NET Framework ou ao .NET Core no Docker para Windows. Contêineres do Windows e do Linux são compatíveis.

## [Publicar em uma imagem do Docker](#)

Saiba como usar a extensão Ferramentas do Visual Studio para Docker para implantar um aplicativo do ASP.NET Core para um host Docker no Azure usando o PowerShell.

## [Configurar o ASP.NET Core para trabalhar com servidores proxy e平衡adores de carga](#)

Configuração adicional pode ser necessária para aplicativos hospedados atrás de servidores proxy e balanceadores de carga. Passar solicitações por meio de um proxy geralmente oculta informações sobre a solicitação original, como o esquema e o IP de cliente. Talvez seja necessário encaminhar manualmente algumas informações sobre a solicitação para o aplicativo.

# Ferramentas do Visual Studio para Docker com ASP.NET Core

30/10/2018 • 19 minutes to read • [Edit Online](#)

O Visual Studio 2017 é compatível com a criação, depuração e execução de aplicativos do ASP.NET Core em contêineres direcionados ao .Net Core. Contêineres do Windows e do Linux são compatíveis.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

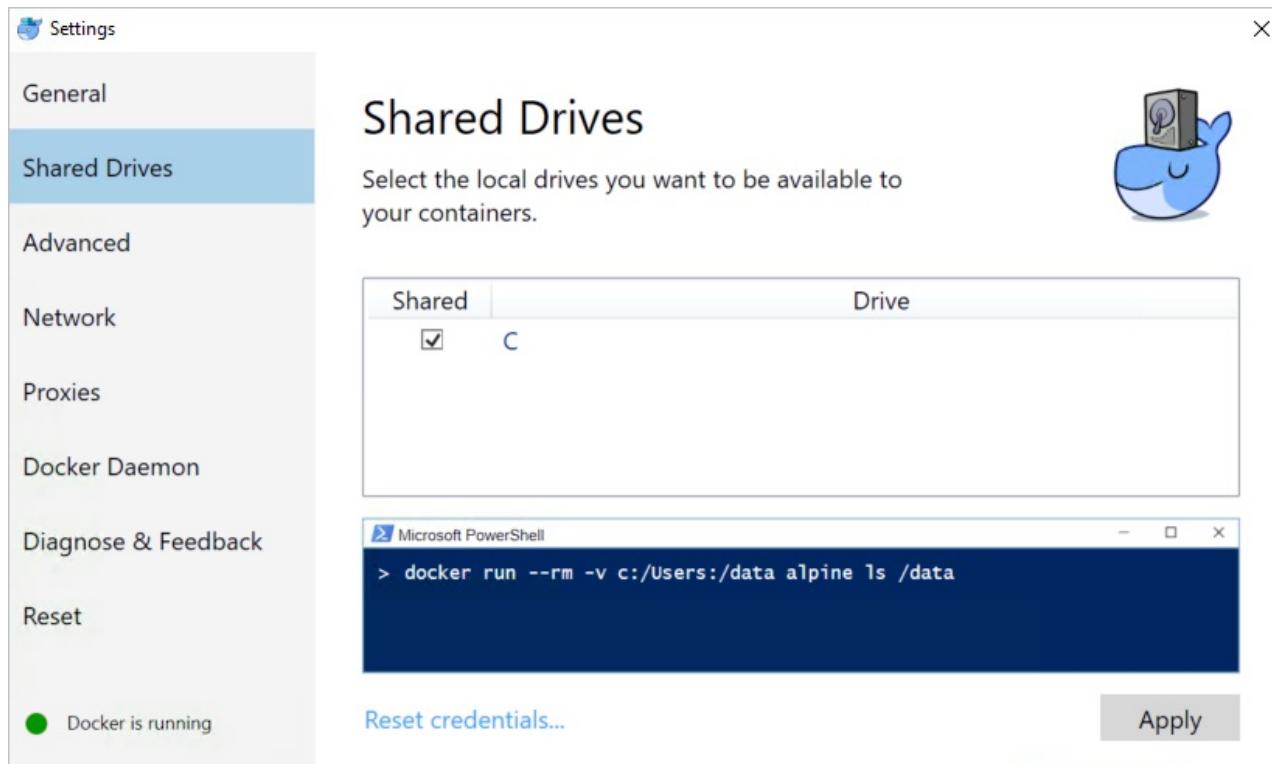
## Pré-requisitos

- [Docker para Windows](#)
- O [Visual Studio 2017](#) com a carga de trabalho **Desenvolvimento de multiplataforma do .NET Core**

## Instalação e configuração

Para a instalação do Docker, primeiro examine as informações em [Docker for Windows: What to know before you install](#) (Docker para Windows: o que saber antes de instalar). Em seguida, instale o [Docker for Windows](#).

As **Unidades Compartilhadas** do Docker para Windows devem ser configuradas para dar suporte ao mapeamento do volume e à depuração. Clique com o botão direito do mouse no ícone do Docker na Bandeja do Sistema, selecione **Configurações** e selecione **Unidades Compartilhadas**. Selecione a unidade em que o Docker armazena arquivos. Clique em **Aplicar**.



### TIP

A versão 15.6 e as versões posteriores do Visual Studio 2017 exibem um aviso quando as **Unidades Compartilhadas** não estão configuradas.

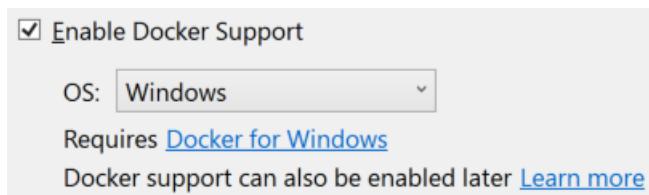
# Adicionar um projeto a um contêiner do Docker

Para colocar um projeto do ASP.NET Core em contêineres, o projeto deve ser destinado ao .Net Core. Contêineres do Linux e Windows são ambos compatíveis.

Ao adicionar o suporte do Docker a um projeto, escolha um contêiner do Windows ou Linux. O host do Docker deve estar executando o mesmo tipo de contêiner. Para alterar o tipo de contêiner na instância do Docker em execução, clique com o botão direito do mouse no ícone do Docker na Bandeja do Sistema e escolha **Alternar para contêineres do Windows...** ou **Alternar para contêineres do Linux....**

## **Novo aplicativo**

Ao criar um novo aplicativo com os modelos de projeto do **Aplicativo Web ASP.NET Core**, marque a caixa de seleção **Habilitar Suporte do Docker**:



Se a estrutura de destino for o .NET Core, a lista suspensa **SO** permitirá a seleção de um tipo de contêiner.

## **Aplicativo existente**

Para projetos do ASP.NET Core direcionados ao .NET Core, há duas opções para adicionar o suporte do Docker por meio das ferramentas. Abra o projeto no Visual Studio e escolha uma das seguintes opções:

- Selecione **Suporte do Docker** no menu **Projeto**.
- Clique com o botão direito do mouse no projeto no **Gerenciador de Soluções** e selecione **Adicionar > Suporte do Docker**.

As Ferramentas do Visual Studio para Docker não dão suporte à adição do Docker a um projeto ASP.NET Core existente direcionado ao .NET Framework.

## Visão geral do Dockerfile

Um *Dockerfile*, a receita para criar uma imagem final do Docker, é adicionado à raiz do projeto. Consulte [Referência do Dockerfile](#) para compreender seus comandos. Esse *Dockerfile* específico usa um [build de vários estágios](#), com quatro estágios de build nomeados distintos:

```

FROM microsoft/dotnet:2.1-aspnetcore-runtime AS base
WORKDIR /app
EXPOSE 59518
EXPOSE 44364

FROM microsoft/dotnet:2.1-sdk AS build
WORKDIR /src
COPY HelloDockerTools/HelloDockerTools.csproj HelloDockerTools/
RUN dotnet restore HelloDockerTools/HelloDockerTools.csproj
COPY . .
WORKDIR /src/HelloDockerTools
RUN dotnet build HelloDockerTools.csproj -c Release -o /app

FROM build AS publish
RUN dotnet publish HelloDockerTools.csproj -c Release -o /app

FROM base AS final
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "HelloDockerTools.dll"]

```

O *Dockerfile* precedente se baseia na imagem [microsoft/dotnet](#). Essa imagem base inclui o tempo de execução do ASP.NET Core e os pacotes NuGet. Os pacotes são compilados JIT (Just-In-Time) para melhorar o desempenho de inicialização.

Quando a caixa de seleção **Configurar para HTTPS** do novo projeto estiver marcada, o *Dockerfile* exibirá duas portas. Uma porta é usada para o tráfego HTTP; a outra porta é usada para o HTTPS. Se a caixa de seleção não estiver marcada, uma única porta (80) será exposta para o tráfego HTTP.

```

FROM microsoft/aspnetcore:2.0 AS base
WORKDIR /app
EXPOSE 80

FROM microsoft/aspnetcore-build:2.0 AS build
WORKDIR /src
COPY HelloDockerTools/HelloDockerTools.csproj HelloDockerTools/
RUN dotnet restore HelloDockerTools/HelloDockerTools.csproj
COPY . .
WORKDIR /src/HelloDockerTools
RUN dotnet build HelloDockerTools.csproj -c Release -o /app

FROM build AS publish
RUN dotnet publish HelloDockerTools.csproj -c Release -o /app

FROM base AS final
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "HelloDockerTools.dll"]

```

O *Dockerfile* precedente se baseia na imagem [microsoft/aspnetcore](#). Essa imagem base inclui os pacotes NuGet do ASP.NET Core, que são compilados JIT (Just-In-Time) para melhorar o desempenho de inicialização.

## Adicionar o suporte de orquestrador de contêineres a um aplicativo

As versões 15.7 ou posteriores do Visual Studio 2017 são compatíveis com o [Docker Compose](#) como a única solução de orquestração de contêineres. Os artefatos do Docker Compose são adicionados por meio da opção **Adicionar > Suporte ao Docker**.

As versões 15.8 ou posteriores do Visual Studio 2017 adicionam uma solução de orquestração apenas quando instruído. Clique com o botão direito do mouse no projeto no **Gerenciador de Soluções** e selecione **Adicionar > Suporte do orquestrador de contêineres**. Duas opções diferentes são oferecidas: [Docker Compose](#) e [Service](#)

Fabric.

## Docker Compose

As Ferramentas do Visual Studio para Docker adicionam um projeto `docker-compose` à solução com os seguintes arquivos:

- `docker-compose.dcproj` – O arquivo que representa o projeto. Inclui um elemento `<DockerTargetos>` que especifica o sistema operacional a ser utilizado.
- `.dockerignore` – Lista os padrões de arquivo e diretório a serem excluídos ao gerar um contexto de build.
- `docker-compose.yml` – O arquivo base do **Docker Compose** usado para definir a coleção de imagens compiladas e executadas com o `docker-compose build` e `docker-compose run`, respectivamente.
- `docker-compose.override.yml` – Um arquivo opcional, lido pelo Docker Compose, com substituições de configuração para os serviços. O Visual Studio executa  
`docker-compose -f "docker-compose.yml" -f "docker-compose.override.yml"` para mesclar esses arquivos.

O arquivo `docker-compose.yml` faz referência ao nome da imagem que é criada quando o projeto é executado:

```
version: '3.4'

services:
  hellodockertools:
    image: ${DOCKER_REGISTRY}hellodockertools
    build:
      context: .
    dockerfile: HelloDockerTools/Dockerfile
```

No exemplo anterior, `image: hellodockertools` gera a imagem `hellodockertools:dev` quando o aplicativo é executado no modo **Depuração**. A imagem `hellodockertools:latest` é gerada quando o aplicativo é executado no modo **Versão**.

Prefixe o nome da imagem com o nome de usuário do **hub do Docker** (por exemplo, `dockerhubusername/hellodockertools`) se a imagem for enviada por push para o registro. Como alternativa, altere o nome da imagem para incluir a URL do registro privado (por exemplo, `privateregistry.domain.com/hellodockertools`), dependendo da configuração.

Se você desejar um comportamento diferente com base na configuração de build (por exemplo, Depuração ou Versão), adicione arquivos `docker-compose` específicos para a configuração. Os arquivos precisam ser nomeados de acordo com a configuração de build (por exemplo, `docker-compose.vs.debug.yml` e `docker-compose.vs.release.yml`) e colocado no mesmo local que o arquivo `docker-compose-override.yml`.

Usando os arquivos de substituição específicos da configuração, é possível especificar definições de configuração diferentes (como variáveis de ambiente ou pontos de entrada) para as configurações de build de Depuração e Versão.

## Service Fabric

Além dos [pré-requisitos](#) básicos, a solução de orquestração do [Service Fabric](#) exige os seguintes pré-requisitos:

- [SDK do Microsoft Azure Service Fabric](#) versão 2.6 ou posterior
- Carga de trabalho de **desenvolvimento do Azure** no Visual Studio 2017

O Service Fabric não é compatível com a execução de contêineres do Linux no cluster de desenvolvimento local no Windows. Se o projeto já estiver usando um contêiner do Linux, o Visual Studio pedirá para alternar para os contêineres do Windows.

As Ferramentas do Visual Studio para Docker realizam as seguintes tarefas:

- Adiciona um projeto `<project_name>do***Aplicativo do Service Fabric*` à solução.

- Adiciona um *Dockerfile* e um arquivo *.dockerignore* ao projeto ASP.NET Core. Se um *Dockerfile* já existir no projeto do ASP.NET Core, ele já terá sido renomeado para *Dockerfile.original*. Um novo *Dockerfile*, semelhante ao seguinte, foi criado:

```
# See https://aka.ms/containerimagehelp for information on how to use Windows Server 1709 containers
with Service Fabric.
# FROM microsoft/aspnetcore:2.0-nanoserver-1709
FROM microsoft/aspnetcore:2.0-nanoserver-sac2016
ARG source
WORKDIR /app
COPY ${source:-obj/Docker/publish} .
ENTRYPOINT ["dotnet", "HelloDockerTools.dll"]
```

- Adiciona um elemento `<IsServiceFabricServiceProject>` ao arquivo *.csproj* do projeto do ASP.NET Core:

```
<IsServiceFabricServiceProject>True</IsServiceFabricServiceProject>
```

- Adiciona uma pasta *PackageRoot* ao projeto ASP.NET Core. A pasta inclui o manifesto do serviço e as configurações para o novo serviço.

Para obter mais informações, consulte [Implantar um aplicativo .NET em um contêiner do Windows no Azure Service Fabric](#).

## Depurar

Selecione **Docker** no menu suspenso de depuração na barra de ferramentas e inicie a depuração do aplicativo. A exibição **Docker** da janela **Saída** mostra as seguintes ações em andamento:

- A marcação *2.1-aspnetcore-runtime* da imagem do tempo de execução *microsoft/dotnet* foi adquirida (se ainda não estiver no cache). A imagem instala os tempos de execução do ASP.NET Core e do .Net Core e bibliotecas associadas. Ela é otimizada para executar aplicativos do ASP.NET Core em produção.
- A variável de ambiente `ASPNETCORE_ENVIRONMENT` é definida como `Development` dentro do contêiner.
- Duas portas atribuídas dinamicamente são expostas: uma para HTTP e outra para HTTPS. A porta atribuída ao localhost pode ser consultada com o comando `docker ps`.
- O aplicativo é copiado para o contêiner.
- O navegador padrão é iniciado com o depurador anexado ao contêiner, usando a porta atribuída dinamicamente.

A imagem resultante do Docker do aplicativo é marcada como *dev*. A imagem é baseada na marcação *2.1-aspnetcore-runtime* da imagem base *microsoft/dotnet*. Execute o comando `docker images` na janela do PMC (**Console do Gerenciador de Pacotes**). As imagens no computador são exibidas:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hellodockertools	dev	d72ce0f1dfe7	30 seconds ago	255MB
microsoft/dotnet	2.1-aspnetcore-runtime	fcc3887985bb	6 days ago	255MB

- A imagem em tempo de execução *microsoft/aspnetcore* é adquirida (se ainda não está no cache).
- A variável de ambiente `ASPNETCORE_ENVIRONMENT` é definida como `Development` dentro do contêiner.
- A porta 80 é exposta e mapeada para uma porta atribuída dinamicamente para o localhost. A porta é determinada pelo host do Docker e pode ser consultada com o comando `docker ps`.
- O aplicativo é copiado para o contêiner.
- O navegador padrão é iniciado com o depurador anexado ao contêiner, usando a porta atribuída dinamicamente.

A imagem resultante do Docker do aplicativo é marcada como *dev*. A imagem é baseada na imagem base *microsoft/aspnetcore*. Execute o comando `docker images` na janela do PMC (**Console do Gerenciador de Pacotes**). As imagens no computador são exibidas:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hellodockertools	dev	5fafef5d1ad5b	4 minutes ago	347MB
microsoft/aspnetcore	2.0	c69d39472da9	13 days ago	347MB

#### NOTE

A imagem *dev* não inclui o conteúdo do aplicativo, pois as configurações de **Depuração** usam a montagem de volume para fornecer uma experiência iterativa. Para enviar uma imagem por push, use a configuração **Versão**.

Execute o comando `docker ps` no PMC. Observe que o aplicativo está em execução usando o contêiner:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS	NAMES			
baf9a678c88d	hellodockertools:dev	"C:\\remote_debugge..."	21 seconds ago	Up 19 seconds
0.0.0.0:37630->80/tcp	dockercompose4642749010770307127_hellodockertools_1			

## Editar e continuar

As alterações em arquivos estáticos e exibições do Razor são atualizadas automaticamente sem a necessidade de uma etapa de compilação. Faça a alteração, salve e atualize o navegador para exibir a atualização.

As modificações nos arquivos de código exigem a compilação e a reinicialização do Kestrel dentro do contêiner. Depois de fazer a alteração, use `CTRL+F5` para executar o processo e iniciar o aplicativo dentro do contêiner. O contêiner do Docker não é recompilado nem interrompido. Execute o comando `docker ps` no PMC. Observe que o contêiner original ainda está em execução como há 10 minutos:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS	NAMES			
baf9a678c88d	hellodockertools:dev	"C:\\remote_debugge..."	10 minutes ago	Up 10 minutes
0.0.0.0:37630->80/tcp	dockercompose4642749010770307127_hellodockertools_1			

## Publicar imagens do Docker

Depois de concluir o ciclo de desenvolvimento e depuração de seu aplicativo, as Ferramentas do Visual Studio para Docker ajudarão você a criar a imagem de produção do aplicativo. Altere a lista suspensa de configuração para **Versão** e compile o aplicativo. O conjunto de ferramentas adquire a imagem de compilação/publicação do hub do Docker (se ainda não estiver no cache). Uma imagem é produzida com a marcação *latest*, que você pode enviar por push para seu registro privado ou para o hub do Docker.

Execute o comando `docker images` no PMC para ver a lista de imagens. Uma saída semelhante à apresentada a seguir será exibida:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hellodockertools	latest	e3984a64230c	About a minute ago	258MB
hellodockertools	dev	d72ce0f1dfe7	4 minutes ago	255MB
microsoft/dotnet	2.1-sdk	9e243db15f91	6 days ago	1.7GB
microsoft/dotnet	2.1-aspnetcore-runtime	fcc3887985bb	6 days ago	255MB

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hellodockertools	latest	cd28f0d4abbd	12 seconds ago	349MB
hellodockertools	dev	5fafe5d1ad5b	23 minutes ago	347MB
microsoft/aspnetcore-build	2.0	7fed40fb647	13 days ago	2.02GB
microsoft/aspnetcore	2.0	c69d39472da9	13 days ago	347MB

As imagens `microsoft/aspnetcore-build` e `microsoft/aspnetcore` listadas na saída anterior são substituídas pelas imagens `microsoft/dotnet` com o .Net Core 2.1. Para obter mais informações, consulte [o comunicado de migração de repositórios do Docker](#).

#### NOTE

O comando `docker images` retorna imagens intermediárias com nomes de repositório e marcas identificadas como `<none>` (não listadas acima). Essas imagens sem nome são produzidas pelo *Dockerfile* no [build de vários estágios](#). Elas melhoraram a eficiência da compilação da imagem final — apenas as camadas necessárias são recompiladas quando ocorrem alterações. Quando você não precisar mais das imagens intermediárias, exclua-as usando o comando `docker rmi`.

Pode haver uma expectativa de que a imagem de produção ou versão seja menor em comparação com a imagem `dev`. Devido ao mapeamento do volume, o depurador e o aplicativo estavam em execução no computador local e não dentro do contêiner. A *última* imagem empacotou o código do aplicativo necessário para executar o aplicativo em um computador host. Portanto, o delta é o tamanho do código do aplicativo.

## Recursos adicionais

- [Desenvolvimento de contêiner com o Visual Studio](#)
- [Azure Service Fabric: prepare seu ambiente de desenvolvimento](#)
- [Implantar um aplicativo .NET em um contêiner do Windows no Azure Service Fabric](#)
- [Soluçinar problemas de desenvolvimento do Visual Studio 2017 com o Docker](#)
- [Repositório do GitHub para Ferramentas do Visual Studio para Docker](#)

# Configure o ASP.NET Core para trabalhar com servidores proxy e平衡adores de carga

08/10/2018 • 26 minutes to read • [Edit Online](#)

Por [Luke Latham](#) e [Chris Ross](#)

Na configuração recomendada para o ASP.NET Core, o aplicativo é hospedado usando IIS/Módulo do ASP.NET Core, Nginx ou Apache. Servidores proxy, balanceadores de carga e outros dispositivos de rede geralmente ocultam informações sobre a solicitação antes de ela alcançar o aplicativo:

- Quando solicitações HTTPS são passadas por proxy por HTTP, o esquema original (HTTPS) é perdido e deve ser encaminhado em um cabeçalho.
- Devido a um aplicativo receber uma solicitação do proxy e não de sua origem verdadeira na Internet ou rede corporativa, o endereço IP do cliente originador também deve ser encaminhado em um cabeçalho.

Essas informações podem ser importantes no processamento de solicitações, por exemplo, em redirecionamentos, autenticação, geração de link, avaliação de política e localização geográfica do cliente.

## Cabeçalhos encaminhados

Por convenção, os proxies encaminham informações em cabeçalhos HTTP.

CABEÇALHO	DESCRIÇÃO
X-Forwarded-For	Contém informações sobre o cliente que iniciou a solicitação e os proxies subsequentes em uma cadeia de proxies. Esse parâmetro pode conter endereços IP (e, opcionalmente, os números de porta). Em uma cadeia de servidores proxy, o primeiro parâmetro indica o cliente em que a solicitação foi feita pela primeira vez. Depois, vêm os identificadores de proxy subsequentes. O último proxy na cadeia não está na lista de parâmetros. O endereço IP do último proxy (e opcionalmente um número da porta) estão disponíveis como o endereço IP remoto na camada de transporte.
X-Forwarded-Proto	O valor do esquema de origem (HTTP/HTTPS). O valor também pode ser uma lista de esquemas se a solicitação percorreu vários proxies.
X-Forwarded-Host	O valor original do campo de cabeçalho do host. Normalmente, os proxies não modificam o cabeçalho do host. Veja <a href="#">Microsoft Security Advisory CVE-2018-0787</a> para obter informações sobre uma vulnerabilidade de elevação de privilégios que afeta os sistemas em que o proxy não valida ou restringe cabeçalhos de Host a valores válidos conhecidos.

O middleware de cabeçalhos encaminhados, do pacote [Microsoft.AspNetCore.HttpOverrides](#), lê esses cabeçalhos e preenche os campos associados em [HttpContext](#).

As atualizações de middleware:

- `HttpContext.Connection.RemoteIpAddress` – Definido usando o valor do cabeçalho `X-Forwarded-For`. Configurações adicionais influenciam o modo como o middleware define `RemoteIpAddress`. Para obter detalhes, veja as [Opções de middleware de cabeçalhos encaminhados](#).
- `HttpContext.Request.Scheme` – Definido usando o valor do cabeçalho `X-Forwarded-Proto`.
- `HttpContext.Request.Host` – Definido usando o valor do cabeçalho `X-Forwarded-Host`.

As [configurações padrão](#) de middleware de cabeçalhos encaminhados podem ser definidas. As configurações padrão são:

- Há apenas *um proxy* entre o aplicativo e a origem das solicitações.
- Somente os endereços de loopback são configurados para proxies conhecidos e redes conhecidas.
- Os cabeçalhos encaminhados são nomeados `X-Forwarded-For` e `X-Forwarded-Proto`.

Nem todos os dispositivos de rede adicionam os cabeçalhos `X-Forwarded-For` e `X-Forwarded-Proto` sem configuração adicional. Consulte as diretrizes do fabricante do dispositivo se as solicitações de proxies não contiverem esses cabeçalhos quando atingirem o aplicativo. Se o dispositivo usar nomes de cabeçalho diferentes de `X-Forwarded-For` e de `X-Forwarded-Proto`, defina as opções `ForwardedForHeaderName` e `ForwardedProtoHeaderName` para corresponderem aos nomes de cabeçalho usados pelo dispositivo. Para obter mais informações, consulte [Forwarded Headers Middleware options](#) (Opções de middleware de cabeçalhos encaminhados) e [Configuration for a proxy that uses different header names](#) (Configuração de um proxy que usa diferentes nomes de cabeçalho).

## O IIS/IIS Express e o Módulo do ASP.NET Core

O middleware de cabeçalhos encaminhados é habilitado por padrão pelo middleware de integração do IIS quando o aplicativo é executado por trás do IIS e do Módulo do ASP.NET Core. O middleware de cabeçalhos encaminhados é ativado para ser executado primeiro no pipeline de middleware, com uma configuração restrita específica para o Módulo do ASP.NET Core devido a questões de confiança com cabeçalhos encaminhados (por exemplo, [falsificação de IP](#)). O middleware está configurado para encaminhar os cabeçalhos `X-Forwarded-For` e `X-Forwarded-Proto` e é restrito a um proxy de localhost único. Se configuração adicional for necessária, veja as [Opções de middleware de cabeçalhos encaminhados](#).

## Outros cenários de servidor proxy e balanceador de carga

Exceto pelo uso de middleware de integração do IIS, o middleware de cabeçalhos encaminhados não é habilitado por padrão. O middleware de cabeçalhos encaminhados deve ser habilitado para um aplicativo para processar cabeçalhos encaminhados com `UseForwardedHeaders`. Após a habilitação do middleware, se nenhum `ForwardedHeadersOptions` for especificado para o middleware, o `ForwardedHeadersOptions.ForwardedHeaders` padrão será `ForwardedHeaders.None`.

Configure o middleware com `ForwardedHeadersOptions` para encaminhar os cabeçalhos `X-Forwarded-For` e `X-Forwarded-Proto` em `Startup.ConfigureServices`. invoque o método `UseForwardedHeaders` em `Startup.Configure` antes de chamar outro middleware:

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    services.Configure<ForwardedHeadersOptions>(options =>
    {
        options.ForwardedHeaders =
            ForwardedHeaders.XForwardedFor | ForwardedHeaders.XForwardedProto;
    });
}

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseForwardedHeaders();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseStaticFiles();
    // In ASP.NET Core 1.x, replace the following line with: app.UseIdentity();
    app.UseAuthentication();
    app.UseMvc();
}

```

#### NOTE

Se nenhuma `ForwardedHeadersOptions` é especificada em `Startup.ConfigureServices` ou diretamente para o método de extensão com `UseForwardedHeaders(IApplicationBuilder, ForwardedHeadersOptions)`, os cabeçalhos padrão para encaminhar são `ForwardedHeaders.None`. A propriedade `ForwardedHeadersOptions.ForwardedHeaders` deve ser configurada com os cabeçalhos para encaminhar.

## Configuração de Nginx

Para encaminhar os cabeçalhos `X-Forwarded-For` e `X-Forwarded-Proto`, consulte [Host ASP.NET Core no Linux com Nginx](#). Para obter mais informações, veja [NGINX: usando o cabeçalho encaminhado](#).

## Configuração do Apache

`X-Forwarded-For` é adicionado automaticamente (veja [mod\\_proxy do módulo do Apache: cabeçalhos de solicitação de proxy reverso](#)). Para obter informações sobre como encaminhar o cabeçalho `X-Forwarded-Proto`, consulte [Hospedar o ASP.NET Core no Linux com o Apache](#).

## Opções de middleware de cabeçalhos encaminhados

`ForwardedHeadersOptions` controlam o comportamento do middleware de cabeçalhos encaminhados. O seguinte exemplo altera os valores padrão:

- Limite o número de entradas nos cabeçalhos encaminhados a `2`.
- Adicione um endereço de proxy conhecido de `127.0.10.1`.
- Altere o nome do cabeçalho encaminhado do padrão `X-Forwarded-For` para `X-Forwarded-For-My-Custom-Header-Name`.

```

services.Configure<ForwardedHeadersOptions>(options =>
{
    options.ForwardLimit = 2;
    options.KnownProxies.Add(IPAddress.Parse("127.0.10.1"));
    options.ForwardedForHeaderName = "X-Forwarded-For-My-Custom-Header-Name";
});

```

OPÇÃO	DESCRIÇÃO
AllowedHosts	<p>Restringe os hosts com o cabeçalho <code>X-Forwarded-Host</code> para os valores fornecidos.</p> <ul style="list-style-type: none"> <li>• Os valores são comparados usando ordinal-ignore-case.</li> <li>• Os número de porta devem ser excluídos.</li> <li>• Se a lista estiver vazia, todos os hosts serão permitidos.</li> <li>• Um curinga de nível superior <code>*</code> permite todos os hosts não vazios.</li> <li>• Curingas de subdomínio são permitidos, mas não correspondem ao domínio raiz. Por exemplo, <code>*.contoso.com</code> corresponde o subdomínio <code>foo.contoso.com</code>, mas não ao domínio raiz <code>contoso.com</code>.</li> <li>• Nomes do host Unicode são permitidos, mas são convertidos em <a href="#">Punycode</a> para correspondência.</li> <li>• <a href="#">Endereços IPv6</a> devem incluir colchetes delimitadores e estar no <a href="#">formato convencional</a> (por exemplo, <code>[ABCD:EF01:2345:6789:ABCD:EF01:2345:6789]</code>).</li> </ul> <p>Endereços IPv6 não têm caso especial para verificar se há igualdade lógica entre formatos diferentes, e nenhuma canonicalização é executada.</p> <ul style="list-style-type: none"> <li>• Falha ao restringir os hosts permitidos pode permitir que um atacante falsifique links gerados pelo serviço.</li> </ul> <p>O valor padrão é uma <a href="#">IList&lt;cadeia de caracteres&gt;</a> vazia.</p>
ForwardedForHeaderName	<p>Use o cabeçalho especificado por essa propriedade, em vez de um especificado por <a href="#">ForwardedHeadersDefaults.XForwardedForHeaderName</a>. Esta opção é usada quando o proxy/encaminhador não usa o cabeçalho <code>X-Forwarded-For</code>, mas usa algum outro cabeçalho para encaminhar as informações.</p> <p>O padrão é <code>X-Forwarded-For</code>.</p>
ForwardedHeaders	<p>Identifica quais encaminhadores devem ser processados. Veja o <a href="#">ForwardedHeaders Enum</a> para a lista de campos aplicáveis. Os valores típicos atribuídos a essa propriedade são</p> <div style="border: 1px solid #ccc; padding: 2px;"> <code>ForwardedHeaders.XForwardedFor   ForwardedHeaders.XForwardedProto</code> </div> <p>O valor padrão é <a href="#">ForwardedHeaders.None</a>.</p>

OPÇÃO	DESCRIÇÃO
<a href="#">ForwardedHostHeaderName</a>	<p>Use o cabeçalho especificado por essa propriedade, em vez de um especificado por <a href="#">ForwardedHeadersDefaults.XForwardedHostHeaderName</a>. Esta opção é usada quando o proxy/encaminhador não usa o cabeçalho <code>X-Forwarded-Host</code>, mas usa algum outro cabeçalho para encaminhar as informações.</p> <p>O padrão é <code>X-Forwarded-Host</code>.</p>
<a href="#">ForwardedProtoHeaderName</a>	<p>Use o cabeçalho especificado por essa propriedade, em vez de um especificado por <a href="#">ForwardedHeadersDefaults.XForwardedProtoHeaderName</a>. Esta opção é usada quando o proxy/encaminhador não usa o cabeçalho <code>X-Forwarded-Proto</code>, mas usa algum outro cabeçalho para encaminhar as informações.</p> <p>O padrão é <code>X-Forwarded-Proto</code>.</p>
<a href="#">ForwardLimit</a>	<p>Limita o número de entradas nos cabeçalhos que são processados. Defina para <code>null</code> para desabilitar o limite, mas isso só deve ser feito se <code>KnownProxies</code> ou <code>KnownNetworks</code> estão configurados.</p> <p>O padrão é 1.</p>
<a href="#">KnownNetworks</a>	<p>Intervalos de endereços de redes conhecidas dos quais aceitar cabeçalhos encaminhados. Forneça os intervalos de IP usando notação de CIDR (Roteamento entre Domínios sem Classificação).</p> <p>O servidor usa soquetes de modo duplo e os endereços IPv4 são fornecidos em um formato IPv6 (por exemplo, <code>10.0.0.1</code> no formato IPv4 representado no formato IPv6 como <code>::ffff:10.0.0.1</code>). Confira <a href="#">IPAddress.MapToIPv6</a>. Para determinar se este formato é obrigatório, veja <a href="#">HttpContext.Connection.RemoteIpAddress</a>. Para saber mais, veja a seção <a href="#">Configuração de endereços IPv4 representados como endereços IPv6</a>.</p> <p>O padrão é um <code>IList&lt;IPNetwork&gt;</code> contendo uma única entrada para <code> IPAddress.Loopback</code>.</p>

OPÇÃO	DESCRIÇÃO
KnownProxies	<p>Endereços de proxies conhecidos dos quais aceitar cabeçalhos encaminhados. Use <code>KnownProxies</code> especificar correspondências exatas de endereço IP.</p> <p>O servidor usa soquetes de modo duplo e os endereços IPv4 são fornecidos em um formato IPv6 (por exemplo, <code>10.0.0.1</code> no formato IPv4 representado no formato IPv6 como <code>::ffff:10.0.0.1</code>). Confira <code>IPAddress.MapToIPv6</code>. Para determinar se este formato é obrigatório, veja <code>HttpContext.Connection.RemoteIpAddress</code>. Para saber mais, veja a seção <a href="#">Configuração de endereços IPv4 representados como endereços IPv6</a>.</p> <p>O padrão é um <code>IList&lt;IPAddress&gt;</code> contendo uma única entrada para <code> IPAddress.IPv6Loopback</code>.</p>
OriginalForHeaderName	<p>Use o cabeçalho especificado por essa propriedade, em vez de um especificado por <code>ForwardedHeadersDefaults.XOriginalForHeaderName</code>.</p> <p>O padrão é <code>x-Original-For</code>.</p>
OriginalHostHeaderName	<p>Use o cabeçalho especificado por essa propriedade, em vez de um especificado por <code>ForwardedHeadersDefaults.XOriginalHostHeaderName</code>.</p> <p>O padrão é <code>x-Original-Host</code>.</p>
OriginalProtoHeaderName	<p>Use o cabeçalho especificado por essa propriedade, em vez de um especificado por <code>ForwardedHeadersDefaults.XOriginalProtoHeaderName</code>.</p> <p>O padrão é <code>x-Original-Proto</code>.</p>
RequireHeaderSymmetry	<p>Exigem o número de valores de cabeçalho a serem sincronizados entre os <code>ForwardedHeadersOptions.ForwardedHeaders</code> sendo processados.</p> <p>O padrão no ASP.NET Core 1.x é <code>true</code>. O padrão no ASP.NET Core 2.0 ou posterior é <code>false</code>.</p>
OPÇÃO	DESCRIÇÃO
ForwardedForHeaderName	<p>Use o cabeçalho especificado por essa propriedade, em vez de um especificado por <code>ForwardedHeadersDefaults.XForwardedForHeaderName</code>. Esta opção é usada quando o proxy/encaminhador não usa o cabeçalho <code>X-Forwarded-For</code>, mas usa algum outro cabeçalho para encaminhar as informações.</p> <p>O padrão é <code>x-Forwarded-For</code>.</p>

OPÇÃO	DESCRIÇÃO
<a href="#">ForwardedHeaders</a>	<p>Identifica quais encaminhadores devem ser processados. Veja o <a href="#">ForwardedHeaders Enum</a> para a lista de campos aplicáveis. Os valores típicos atribuídos a essa propriedade são</p> <pre data-bbox="879 309 1250 361"><code>ForwardedHeaders.XForwardedFor   ForwardedHeaders.XForwardedProto</code></pre> <p>O valor padrão é <code>ForwardedHeaders.None</code>.</p>
<a href="#">ForwardedHostHeaderName</a>	<p>Use o cabeçalho especificado por essa propriedade, em vez de um especificado por <a href="#">ForwardedHeadersDefaults.XForwardedHostHeaderName</a>. Esta opção é usada quando o proxy/encaminhador não usa o cabeçalho <code>X-Forwarded-Host</code>, mas usa algum outro cabeçalho para encaminhar as informações.</p> <p>O padrão é <code>x-Forwarded-Host</code>.</p>
<a href="#">ForwardedProtoHeaderName</a>	<p>Use o cabeçalho especificado por essa propriedade, em vez de um especificado por <a href="#">ForwardedHeadersDefaults.XForwardedProtoHeaderName</a>. Esta opção é usada quando o proxy/encaminhador não usa o cabeçalho <code>X-Forwarded-Proto</code>, mas usa algum outro cabeçalho para encaminhar as informações.</p> <p>O padrão é <code>x-Forwarded-Proto</code>.</p>
<a href="#">ForwardLimit</a>	<p>Limita o número de entradas nos cabeçalhos que são processados. Defina para <code>null</code> para desabilitar o limite, mas isso só deve ser feito se <a href="#">KnownProxies</a> ou <a href="#">KnownNetworks</a> estão configurados.</p> <p>O padrão é 1.</p>
<a href="#">KnownNetworks</a>	<p>Intervalos de endereços de redes conhecidas dos quais aceitar cabeçalhos encaminhados. Forneça os intervalos de IP usando notação de CIDR (Roteamento entre Domínios sem Classificação).</p> <p>O servidor usa soquetes de modo duplo e os endereços IPv4 são fornecidos em um formato IPv6 (por exemplo, <code>10.0.0.1</code> no formato IPv4 representado no formato IPv6 como <code>::ffff:10.0.0.1</code>). Confira <a href="#">IPAddress.MapToIPv6</a>. Para determinar se este formato é obrigatório, veja <a href="#">HttpContext.Connection.RemoteIpAddress</a>. Para saber mais, veja a seção <a href="#">Configuração de endereços IPv4 representados como endereços IPv6</a>.</p> <p>O padrão é um <code>IList&lt;IPNetwork&gt;</code> contendo uma única entrada para <code> IPAddress.Loopback</code>.</p>

OPÇÃO	DESCRIÇÃO
<a href="#">KnownProxies</a>	<p>Endereços de proxies conhecidos dos quais aceitar cabeçalhos encaminhados. Use <code>KnownProxies</code> especificar correspondências exatas de endereço IP.</p> <p>O servidor usa soquetes de modo duplo e os endereços IPv4 são fornecidos em um formato IPv6 (por exemplo, <code>10.0.0.1</code> no formato IPv4 representado no formato IPv6 como <code>::ffff:10.0.0.1</code>). Confira <a href="#">IPAddress.MapToIPv6</a>. Para determinar se este formato é obrigatório, veja <a href="#">HttpContext.Connection.RemoteIpAddress</a>. Para saber mais, veja a seção <a href="#">Configuração de endereços IPv4 representados como endereços IPv6</a>.</p> <p>O padrão é um <code>IList&lt;IPAddress&gt;</code> contendo uma única entrada para <code> IPAddress.IPv6Loopback</code>.</p>
<a href="#">OriginalForHeaderName</a>	<p>Use o cabeçalho especificado por essa propriedade, em vez de um especificado por <a href="#">ForwardedHeadersDefaults.XOriginalForHeaderName</a>.</p> <p>O padrão é <code>x-Original-For</code>.</p>
<a href="#">OriginalHostHeaderName</a>	<p>Use o cabeçalho especificado por essa propriedade, em vez de um especificado por <a href="#">ForwardedHeadersDefaults.XOriginalHostHeaderName</a>.</p> <p>O padrão é <code>x-Original-Host</code>.</p>
<a href="#">OriginalProtoHeaderName</a>	<p>Use o cabeçalho especificado por essa propriedade, em vez de um especificado por <a href="#">ForwardedHeadersDefaults.XOriginalProtoHeaderName</a>.</p> <p>O padrão é <code>x-Original-Proto</code>.</p>
<a href="#">RequireHeaderSymmetry</a>	<p>Exigem o número de valores de cabeçalho a serem sincronizados entre os <a href="#">ForwardedHeadersOptions.ForwardedHeaders</a> sendo processados.</p> <p>O padrão no ASP.NET Core 1.x é <code>true</code>. O padrão no ASP.NET Core 2.0 ou posterior é <code>false</code>.</p>

## Cenários e casos de uso

### Quando não é possível adicionar cabeçalhos encaminhados e todas as solicitações são seguras

Em alguns casos, pode não ser possível adicionar cabeçalhos encaminhados para as solicitações passadas por proxy ao aplicativo. Se o proxy está impondo que todas as solicitações externas públicas sejam HTTPS, o esquema pode ser definido manualmente em `startup.Configure` antes de usar qualquer tipo de middleware:

```
app.Use((context, next) =>
{
    context.Request.Scheme = "https";
    return next();
});
```

Esse código pode ser desabilitado com uma variável de ambiente ou outra definição de configuração em um ambiente de preparo ou de desenvolvimento.

### Lidar com o caminho base e proxies que alteram o caminho da solicitação

Alguns proxies passam o caminho intacto, mas com um caminho base de aplicativo que deve ser removido para que o roteamento funcione corretamente. O middleware de [UsePathBaseExtensions.UsePathBase](#) divide o caminho em [HttpRequest.Path](#) e o caminho base do aplicativo em [HttpRequest.PathBase](#).

Se `/foo` é o caminho base do aplicativo para um caminho de proxy passado como `/foo/api/1`, o middleware define `Request.PathBase` para `/foo` e `Request.Path` para `/api/1` com o seguinte comando:

```
app.UsePathBase("/foo");
```

O caminho original e o caminho base são reaplicados quando o middleware é chamado novamente na ordem inversa. Para obter mais informações sobre o processamento de ordem de middleware, consulte [Middleware do ASP.NET Core](#).

Se o proxy cortar o caminho (por exemplo, encaminhando `/foo/api/1` para `/api/1`), corrija redirecionamentos e links definindo a propriedade [PathBase](#) da solicitação:

```
app.Use((context, next) =>
{
    context.Request.PathBase = new PathString("/foo");
    return next();
});
```

Se o proxy estiver adicionando dados de caminho, descarte a parte do caminho para corrigir redirecionamentos e links usando [StartsWithSegments \(PathString, PathString\)](#) e atribuindo para a propriedade [Path](#):

```
app.Use((context, next) =>
{
    if (context.Request.Path.StartsWithSegments("/foo", out var remainder))
    {
        context.Request.Path = remainder;
    }

    return next();
});
```

### Configuração de um proxy que usa diferentes nomes de cabeçalho

Se o proxy não usar cabeçalhos nomeados `X-Forwarded-For` e `X-Forwarded-Proto` para encaminhar a porta/endereço do proxy e as informações de origem do esquema, defina as opções [ForwardedForHeaderName](#) e [ForwardedProtoHeaderName](#) para corresponder aos nomes de cabeçalho usados pelo proxy:

```
services.Configure<ForwardedHeadersOptions>(options =>
{
    options.ForwardedForHeaderName = "Header_Name_Used_By_Proxy_For_X-Forwarded-For_Header";
    options.ForwardedProtoHeaderName = "Header_Name_Used_By_Proxy_For_X-Forwarded-Proto_Header";
});
```

## Configuração de endereços IPv4 representados como endereços IPv6

O servidor usa soquetes de modo duplo e os endereços IPv4 são fornecidos em um formato IPv6 (por exemplo, `10.0.0.1` no formato IPv4 representado no formato IPv6 como `::ffff:10.0.0.1` ou `::ffff:a00:1`). Confira [IPAddress.MapToIPv6](#). Para determinar se este formato é obrigatório, veja [HttpContext.Connection.RemoteIpAddress](#).

No exemplo a seguir, um endereço de rede que fornece cabeçalhos encaminhados é adicionado à lista `KnownNetworks` no formato IPv6.

Endereço IPv4: `10.11.12.1/8`

Conversão em endereço IPv6: `::ffff:10.11.12.1`

Comprimento do prefixo convertido: 104

Você pode também fornecer o endereço no formato hexadecimal (`10.11.12.1`, representado no formato IPv6 como `::ffff:0a0b:0c01`). Quando converter um endereço IPv4 em IPv6, adicione 96 ao comprimento do prefixo CIDR (`8` no exemplo) para levar em conta o prefixo IPv6 adicional `::ffff:` ( $8 + 96 = 104$ ).

```
// To access IPNetwork and IPAddress, add the following namespaces:
// using System.Net;
// using Microsoft.AspNetCore.HttpOverrides;
services.Configure<ForwardedHeadersOptions>(options =>
{
    options.ForwardedHeaders =
        ForwardedHeaders.XForwardedFor | ForwardedHeaders.XForwardedProto;
    options.KnownNetworks.Add(new IPNetwork(
        IPAddress.Parse("::ffff:10.11.12.1"), 104));
});
```

## Solução de problemas

Quando os cabeçalhos não são encaminhados conforme o esperado, habilite [registro em log](#). Se os logs não fornecerem informações suficientes para solucionar o problema, enumere os cabeçalhos de solicitação recebidos pelo servidor. Use middleware embutido para gravar cabeçalhos de solicitação para uma resposta do aplicativo ou para log dos cabeçalhos. Coloque um dos exemplos de código a seguir imediatamente após a chamada para `UseForwardedHeaders` em `Startup.Configure`.

Para escrever os cabeçalhos na resposta do aplicativo, use o seguinte middleware embutido terminal:

```

app.Run(async (context) =>
{
    context.Response.ContentType = "text/plain";

    // Request method, scheme, and path
    await context.Response.WriteAsync(
        $"Request Method: {context.Request.Method}{Environment.NewLine}");
    await context.Response.WriteAsync(
        $"Request Scheme: {context.Request.Scheme}{Environment.NewLine}");
    await context.Response.WriteAsync(
        $"Request Path: {context.Request.Path}{Environment.NewLine}");

    // Headers
    await context.Response.WriteAsync($"Request Headers:{Environment.NewLine}");

    foreach (var header in context.Request.Headers)
    {
        await context.Response.WriteAsync($"{header.Key}: " +
            $"{header.Value}{Environment.NewLine}");
    }

    await context.Response.WriteAsync(Environment.NewLine);

    // Connection: RemoteIp
    await context.Response.WriteAsync(
        $"Request RemoteIp: {context.Connection.RemoteIpAddress}");
});

```

Também é possível escrever nos logs em vez do corpo da resposta usando o seguinte middleware embutido. Isso permite que o site funcione normalmente durante a depuração.

```

var logger = _loggerFactory.CreateLogger<Startup>();

app.Use(async (context, next) =>
{
    // Request method, scheme, and path
    logger.LogDebug("Request Method: {METHOD}", context.Request.Method);
    logger.LogDebug("Request Scheme: {SCHEME}", context.Request.Scheme);
    logger.LogDebug("Request Path: {PATH}", context.Request.Path);

    // Headers
    foreach (var header in context.Request.Headers)
    {
        logger.LogDebug("Header: {KEY}: {VALUE}", header.Key, header.Value);
    }

    // Connection: RemoteIp
    logger.LogDebug("Request RemoteIp: {REMOTE_IP_ADDRESS}",
        context.Connection.RemoteIpAddress);

    await next();
});

```

Quando processado, os valores `X-Forwarded-{For|Proto|Host}` são movidos para `X-Original-{For|Proto|Host}`. Se há vários valores em um determinado cabeçalho, observe que o middleware de cabeçalhos encaminhados processa cabeçalhos na ordem inversa, da direita para esquerda. O `ForwardLimit` padrão é 1 (um), portanto, apenas o valor mais à direita dos cabeçalhos será processado, a menos que o valor de `ForwardLimit` aumente.

O IP de remoto original da solicitação precisa corresponder a uma entrada nas listas `KnownProxies` ou `KnownNetworks` antes dos cabeçalhos encaminhados serem processados. Isso limita a falsificação de cabeçalho por não aceitar encaminhadores de proxies não confiáveis. Quando um proxy desconhecido é

detectado, o registro em log indica o endereço dele:

```
September 20th 2018, 15:49:44.168 Unknown proxy: 10.0.0.100:54321
```

No exemplo anterior, 10.0.0.100 é um servidor proxy. Se o servidor for um proxy confiável, adicione o endereço IP do servidor a `KnownProxies` (ou adicione uma rede confiável a `KnownNetworks`) em `Startup.ConfigureServices`. Para obter mais informações, consulte a seção [Opções de middleware de cabeçalhos encaminhados](#).

```
services.Configure<ForwardedHeadersOptions>(options =>
{
    options.KnownProxies.Add(IPAddress.Parse("10.0.0.100"));
});
```

#### IMPORTANT

Permitir que somente proxies e redes confiáveis encaminhem os cabeçalhos. Caso contrário, podem ocorrer ataques de [falsificação de IP](#).

## Recursos adicionais

- [Hospedar o ASP.NET Core em um web farm](#)
- [Microsoft Security Advisory CVE-2018-0787: vulnerabilidade de elevação de privilégio do ASP.NET Core](#)

# Hospedar o ASP.NET Core em um web farm

12/12/2018 • 9 minutes to read • [Edit Online](#)

Por [Luke Latham](#) e [Chris Ross](#)

Um *web farm* é um grupo de dois ou mais servidores web (ou *nós*) que hospedam várias instâncias de um aplicativo. Quando as solicitações de usuários chegam a um web farm, um *balanceador de carga* distribui as solicitações para os nós de farm da web. Os web farms melhoram:

- **Disponibilidade/confiabilidade** – quando um ou mais nós falham, o balanceador de carga pode rotear solicitações para outros nós em funcionamento para continuar a processar solicitações.
- **Capacidade e desempenho** – vários nós podem processar mais solicitações que um único servidor. O balanceador de carga equilibra a carga de trabalho ao distribuir as solicitações para os nós.
- **Escalabilidade** – quando a necessidade da capacidade for maior ou menor, o número de nós ativos pode ser aumentado ou diminuído para corresponder à carga de trabalho. Tecnologias de plataforma de web farm, como o [Serviço de Aplicativo do Azure](#), podem adicionar ou remover nós por solicitação do administrador do sistema ou automaticamente sem a intervenção humana.
- **Facilidade de manutenção** – os nós de um web farm podem contar com um conjunto de serviços compartilhados, o que resulta em um gerenciamento mais fácil do sistema. Por exemplo, os nós de um web farm podem contar com um servidor de banco de dados individual e um local de rede comum para recursos estáticos, como imagens e arquivos para download.

Este tópico descreve as configurações e dependências para aplicativos ASP.NET Core hospedados em um web farm que conta com recursos compartilhados.

## Configuração geral

### [Hospedar e implantar o ASP.NET Core](#)

Aprenda como configurar ambientes de hospedagem e implantar aplicativos ASP.NET Core. Configure um gerenciador de processo em cada nó do web farm para automatizar o início e a reinicialização do aplicativo. Cada nó requer o tempo de execução do ASP.NET Core. Para saber mais, confira os tópicos na área [Hospedar e implantar](#) da documentação.

### [Configure o ASP.NET Core para trabalhar com servidores proxy e平衡adores de carga](#)

Saiba mais sobre a configuração para aplicativos hospedados por trás de servidores proxy e平衡adores de carga, o que muitas vezes oculta informações de solicitação importantes.

### [Implantar aplicativos ASP.NET Core no Serviço de Aplicativo do Azure](#)

[Serviço de Aplicativo do Azure](#) é um [serviço de plataforma de computação em nuvem da Microsoft](#) para hospedar aplicativos Web, incluindo o ASP.NET Core. O Serviço de Aplicativo é uma plataforma totalmente gerenciada que fornece escalabilidade automática, balanceamento de carga, aplicação de patch e implantação contínua.

## Dados do aplicativo

Quando um aplicativo é dimensionado para várias instâncias, pode haver um estado do aplicativo que exija o compartilhamento entre os nós. Se o estado for transitório, considere a possibilidade de compartilhar um [IDistributedCache](#). Se o estado compartilhado exigir persistência, considere armazenar o estado compartilhado em um banco de dados.

# Configuração necessária

O serviço de cache e a proteção de dados exigem uma configuração para aplicativos implantados em um web farm.

## Proteção de Dados

O [sistema de proteção de dados do ASP.NET Core](#) é usado por aplicativos para proteger os dados. A proteção de dados baseia-se em um conjunto de chaves de criptografia armazenados em um *token de autenticação*. Quando o sistema de Proteção de dados é inicializado, ele aplica [as configurações padrão](#) que armazenam localmente o token de autenticação. Sob a configuração padrão, um token de autenticação exclusivo é armazenado em cada nó do web farm. Consequentemente, cada nó do web farm não pode descriptografar os dados criptografados por um aplicativo em qualquer outro nó. A configuração padrão normalmente não é adequada para hospedagem de aplicativos em um web farm. Uma alternativa à implementação de um token de autenticação compartilhado é sempre rotear as solicitações do usuário para o mesmo nó. Para saber mais sobre a configuração do sistema de Proteção de dados para implantações de web farm, confira [Configurar a proteção de dados do ASP.NET Core](#).

## Cache

Em um ambiente de web farm, o mecanismo de cache deve compartilhar os itens em cache pelos nós do web farm. O serviço de cache deve contar com um cache Redis comum, um banco de dados compartilhado do SQL Server ou uma implementação personalizada de cache que compartilha os itens em cache pelo web farm. Para obter mais informações, consulte [O cache no ASP.NET Core distribuído](#).

# Componentes dependentes

Os cenários a seguir não exigem configuração adicional, mas dependem de tecnologias que exigem configurações para web farms.

CENÁRIO	DEPENDE DE ...
Autenticação	<p>Proteção de dados (confira <a href="#">Configurar a proteção de dados do ASP.NET Core</a>).</p> <p>Para obter mais informações, consulte <a href="#">Usar autenticação de cookie sem o ASP.NET Core Identity</a> e <a href="#">Compartilhar cookies entre aplicativos com o ASP.NET e ASP.NET Core</a>.</p>
Identidade	<p>Configuração e autenticação do banco de dados.</p> <p>Para obter mais informações, consulte <a href="#">Introdução à identidade do ASP.NET Core</a>.</p>
Session	<p>Proteção de dados (cookies criptografados) (confira <a href="#">Configurar a proteção de dados do ASP.NET Core</a>) e cache (confira <a href="#">O cache no ASP.NET Core distribuído</a>).</p> <p>Para saber mais, confira <a href="#">Estado de sessão e aplicativo: estado de sessão</a>.</p>
TempData	<p>Proteção de dados (cookies criptografados) (confira <a href="#">Configurar a proteção de dados do ASP.NET Core</a>) ou sessão (confira <a href="#">Estado de sessão e aplicativo: estado de sessão</a>).</p> <p>Para saber mais, consulte <a href="#">Estado de sessão e aplicativo: TempData</a>.</p>

CENÁRIO	DEPENDE DE ...
Antifalsificação	<p>Proteção de dados (confira <a href="#">Configurar a proteção de dados do ASP.NET Core</a>).</p> <p>Para obter mais informações, consulte <a href="#">Ataques de evitar entre solicitação intersetor forjada (CSRF/XSRF) no ASP.NET Core</a>.</p>

## Solução de problemas

### Proteção de dados e cache

Quando a proteção de dados ou cache não está configurada para um ambiente de web farm, erros intermitentes ocorrem quando as solicitações são processadas. Isso acontece porque os nós não compartilham os mesmos recursos e as solicitações do usuário não são sempre roteadas para o mesmo nó.

Considere um usuário que entra no aplicativo usando a autenticação de cookie. O usuário entra no aplicativo em um nó do web farm. Se a sua próxima solicitação chegar ao mesmo nó no qual ele se conectou, o aplicativo consegue descriptografar o cookie de autenticação e permite o acesso ao recurso do aplicativo. Se a sua próxima solicitação chegar em um nó diferente, o aplicativo não consegue descriptografar o cookie de autenticação a partir do nó em que o usuário entrou e a autorização do recurso solicitado falhará.

Quando qualquer um dos seguintes sintomas ocorrem **de forma intermitente**, o problema normalmente é rastreado, indicando a configuração incorreta de proteção de dados ou cache para um ambiente de web farm:

- Quebras de autenticação – o cookie de autenticação está configurado incorretamente ou não pode ser descriptografado. Falha de login OpenIdConnect ou OAuth (Facebook, Microsoft, Twitter) com o erro "Falha de correlação".
- Quebras de autorização – a identidade foi perdida.
- O estado de sessão perde os dados.
- Os itens em cache desaparecem.
- O TempData falhará.
- Os POSTs falham – a verificação antifalsificação falhará.

Para saber mais sobre a configuração de Proteção de dados para implantações de web farm, confira [Configurar a proteção de dados do ASP.NET Core](#). Para saber mais sobre a configuração de cache para implantações de web farm, confira [O cache no ASP.NET Core distribuído](#).

## Obter dados de aplicativos

Se os aplicativos do web farm forem capazes de responder às solicitações, obtenha as solicitações, conexão e dados adicionais dos aplicativos que usarem o middleware embutido de terminal. Para saber mais e obter um código de exemplo, consulte [Solucionar problemas de projetos do ASP.NET Core](#).

# Perfis de publicação do Visual Studio para a implantação do aplicativo ASP.NET Core

30/01/2019 • 23 minutes to read • [Edit Online](#)

Por [Sayed Hashimi de Ibrahim](#) e [Rick Anderson](#)

Para obter a versão 1.1 deste tópico, baixe [Perfis de publicação do Visual Studio para implantação de aplicativo do ASP.NET Core \(versão 1.1, PDF\)](#).

Este documento se concentra no uso do Visual Studio 2017 (ou posterior) para criar e usar perfis de publicação. Os perfis de publicação criados com o Visual Studio podem ser executados do MSBuild e do Visual Studio. Consulte [Publicar um aplicativo Web ASP.NET Core no Serviço de Aplicativo do Azure usando o Visual Studio](#) para obter instruções sobre a publicação no Azure.

O arquivo de projeto a seguir foi criado com o comando `dotnet new mvc`:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

<PropertyGroup>
  <TargetFramework>netcoreapp2.2</TargetFramework>
</PropertyGroup>

<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore.App" />
</ItemGroup>

</Project>
```

```
<Project Sdk="Microsoft.NET.Sdk.Web">

<PropertyGroup>
  <TargetFramework>netcoreapp2.1</TargetFramework>
</PropertyGroup>

<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore.App" />
</ItemGroup>

</Project>
```

O atributo `<Project>` do elemento `Sdk` realiza as seguintes tarefas:

- Importa o arquivo de propriedades de `$(MSBuildSDKsPath)\Microsoft.NET.Sdk.Web\Sdk\Sdk.props` no início.
- Importa o arquivo de destino de `$(MSBuildSDKsPath)\Microsoft.NET.Sdk.Web\Sdk\Sdk.targets` no fim.

O local padrão para `MSBuildSDKsPath` (com o Visual Studio Enterprise 2017) é a pasta `%programfiles(x86)%\Microsoft Visual Studio\2017\Enterprise\MSBuild\Sdks`.

O SDK de `Microsoft.NET.Sdk.Web` depende de:

- `Microsoft.NET.Sdk.Web.ProjectSystem`
- `Microsoft.NET.Sdk.Publish`

O que faz com que as propriedades e os destinos a seguir a sejam importados:

- `$(MSBuildSDKsPath)\Microsoft.NET.Sdk.Web.ProjectSystem\Sdk\Sdk.Props`
- `$(MSBuildSDKsPath)\Microsoft.NET.Sdk.Web.ProjectSystem\Sdk\Sdk.targets`
- `$(MSBuildSDKsPath)\Microsoft.NET.Sdk.Publish\Sdk\Sdk.Props`
- `$(MSBuildSDKsPath)\Microsoft.NET.Sdk.Publish\Sdk\Sdk.targets`

Destinos de publicação importam o conjunto certo de destinos com base no método de publicação usado.

Quando o MSBuild ou o Visual Studio carrega um projeto, as seguintes ações de nível alto ocorrem:

- Compilar projeto
- Computar arquivos a publicar
- Publicar arquivos para o destino

## Itens de projeto de computação

Quando o projeto é carregado, os itens de projeto (arquivos) são computados. O atributo `item type` determina como o arquivo é processado. Por padrão, os arquivos `.cs` são incluídos na lista de itens `Compile`. Os arquivos na lista de itens `Compile` são compilados.

A lista de itens `Content` contém arquivos que são publicados juntamente com as saídas de build. Por padrão, os arquivos correspondendo ao padrão `wwwroot/**` são incluídos no item `Content`. O padrão glob `wwwroot/\*/*` corresponde a todos os arquivos na pasta `wwwroot` e subpastas. Para adicionar explicitamente um arquivo à lista de publicação, adicione o arquivo diretamente no arquivo `.csproj` conforme mostrado em [Arquivos de Inclusão](#).

Ao selecionar o botão **Publicar** no Visual Studio ou ao publicar da linha de comando:

- Os itens/propriedades são calculados (os arquivos necessários para compilar).
- **Somente Visual Studio:** os pacotes do NuGet são restaurados. (A restauração precisa ser explícita pelo usuário na CLI.)
- O projeto é compilado.
- Os itens de publicação são computados (os arquivos necessários para a publicação).
- O projeto é publicado (os arquivos computados são copiados para o destino de publicação).

Quando um projeto do ASP.NET Core faz referência a `Microsoft.NET.Sdk.Web` no arquivo de projeto, um arquivo `app_offline.htm` é colocado na raiz do diretório do aplicativo Web. Quando o arquivo estiver presente, o módulo do ASP.NET Core apenas desligará o aplicativo e servirá o arquivo `app_offline.htm` durante a implantação. Para obter mais informações, consulte [Referência de configuração do módulo do ASP.NET Core](#).

## Publicação de linha de comando básica

A publicação de linha de comando funciona em todas as plataformas compatíveis com o .NET Core e não requer o Visual Studio. Nas amostras abaixo, o comando `dotnet publish` é executado no diretório do projeto (que contém o arquivo `.csproj`). Se você não estiver na pasta do projeto, passe explicitamente no caminho do arquivo de projeto. Por exemplo:

```
dotnet publish C:\Webs\Web1
```

Execute os comandos a seguir para criar e publicar um aplicativo Web:

```
dotnet new mvc  
dotnet publish
```

O comando `dotnet publish` gera uma saída semelhante à seguinte:

```
C:\Webs\Web1>dotnet publish  
Microsoft (R) Build Engine version 15.3.409.57025 for .NET Core  
Copyright (C) Microsoft Corporation. All rights reserved.  
  
Web1 -> C:\Webs\Web1\bin\Debug\netcoreapp2.0\Web1.dll  
Web1 -> C:\Webs\Web1\bin\Debug\netcoreapp2.0\publish\
```

A pasta de publicação padrão é `bin\$(Configuration)\netcoreapp<version>\publish`. O padrão para `$(Configuration)` é `Debug`. Na amostra anterior, o `<TargetFramework>` é `netcoreapp2.0`.

`dotnet publish -h` exibe informações de ajuda para publicação.

O comando a seguir especifica um build de `Release` e o diretório de publicação:

```
dotnet publish -c Release -o C:\MyWebs\test
```

O comando `dotnet publish` chama MSBuild, que invoca o destino `Publish`. Quaisquer parâmetros passados para `dotnet publish` são passados para o MSBuild. O parâmetro `-c` é mapeado para a propriedade do MSBuild `Configuration`. O parâmetro `-o` é mapeado para `OutputPath`.

Propriedades do MSBuild podem ser passadas usando qualquer um dos seguintes formatos:

- `p:<NAME>=<VALUE>`
- `/p:<NAME>=<VALUE>`

O comando a seguir publica um build de `Release` para um compartilhamento de rede:

```
dotnet publish -c Release /p:PublishDir=//r8/release/AdminWeb
```

O compartilhamento de rede é especificado com barras "/" (`//r8/`) e funciona em todas as plataformas com suporte do .NET Core.

Confirme que o aplicativo publicado para implantação não está em execução. Os arquivos da pasta `publish` são bloqueados quando o aplicativo está em execução. A implantação não pode ocorrer porque arquivos bloqueados não podem ser copiados.

## Perfis de publicação

Esta seção usa o Visual Studio 2017 (ou posterior) para criar um perfil de publicação. Uma vez criado, é possível publicar do Visual Studio ou da linha de comando.

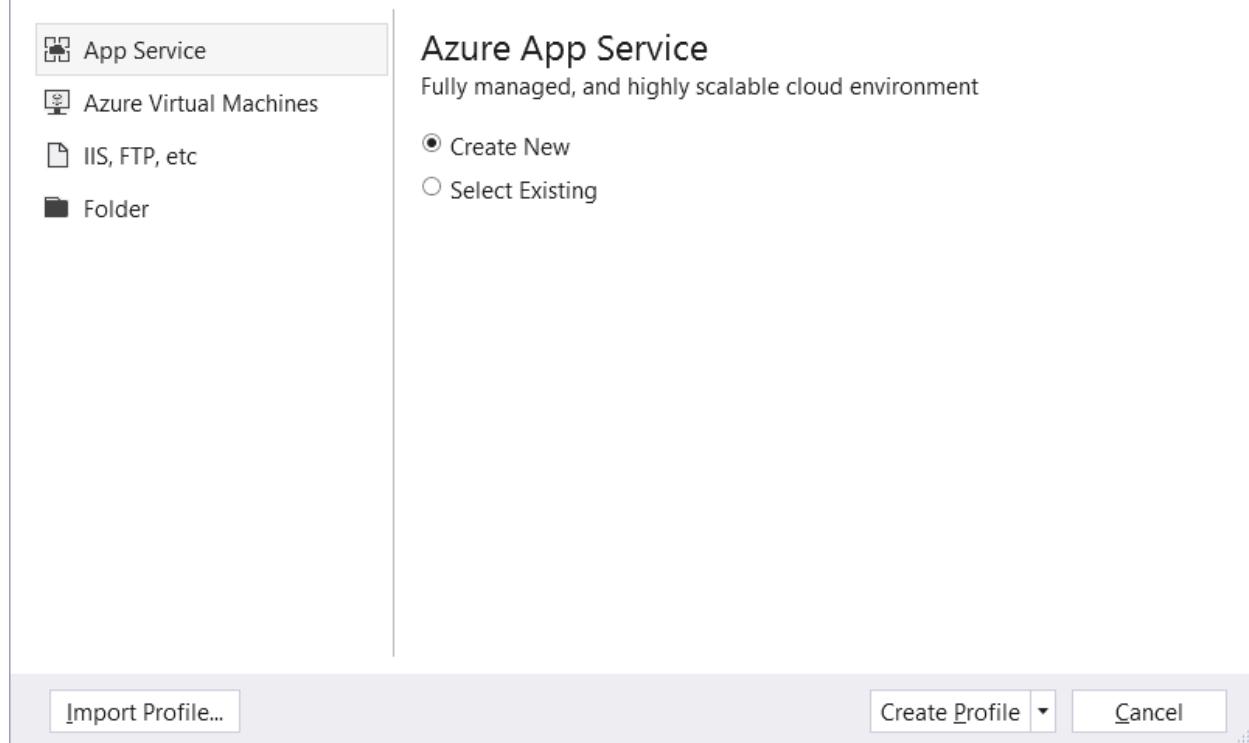
Perfis de publicação podem simplificar o processo de publicação e podem existir em qualquer número. Crie um perfil de publicação no Visual Studio escolhendo um dos seguintes caminhos:

- Clique com o botão direito do mouse no projeto no Gerenciador de Soluções e selecione **Publicar**.
- Selecione **Publicar {NOME DO PROJETO}** no menu **Build**.

A guia **Publicar** da página de capacidades do aplicativo é exibida. Se o projeto não tem um perfil de publicação, a página a seguir é exibida:

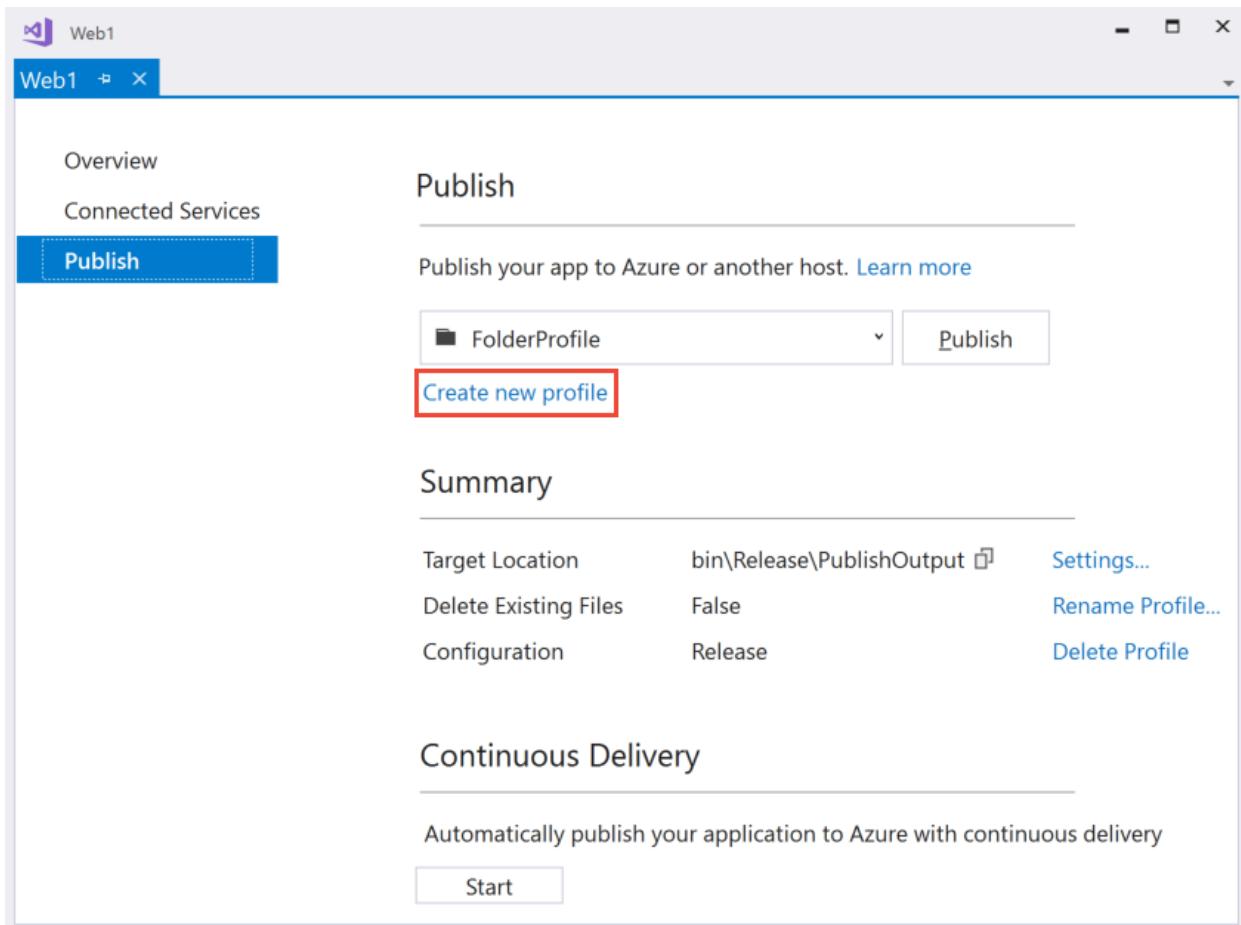
x

## Pick a publish target



Quando a opção **Pasta** é selecionada, especifique um caminho de pasta para armazenar os ativos publicados. A pasta padrão é `bin\Release\PublishOutput`. Clique no botão **Criar Perfil** para concluir.

Depois de um perfil de publicação ser criado, a guia **Publicar** é alterada. O perfil recém-criado é exibido em uma lista suspensa. Clique em **Criar novo perfil** para fazer isso.



O Assistente de publicação dá suporte aos destinos de publicação a seguir:

- Serviço de Aplicativo do Azure
- Máquinas Virtuais do Azure
- IIS, FTP, etc. (para qualquer servidor Web)
- Pasta
- Importar Perfil

Para obter mais informações, confira [Quais opções de publicação são adequadas para mim.](#)

Quando você cria um perfil de publicação com o Visual Studio, um arquivo do MSBuild *Properties/PublishProfiles/{NOME DO PERFIL}.pubxml* é criado. O *.pubxml* é um arquivo do MSBuild e contém definições de configuração de publicação. Esse arquivo pode ser alterado para personalizar o processo de build e de publicação. Esse arquivo é lido pelo processo de publicação. `<LastUsedBuildConfiguration>` é especial porque é uma propriedade global e não deverá estar em nenhum arquivo importado no build. Para obter mais informações, confira [MSBuild: como definir a propriedade de configuração](#).

Ao publicar em um destino do Azure, o arquivo *.pubxml* contém o identificador de assinatura do Azure. Com esse tipo de destino, não é recomendável adicionar esse arquivo ao controle do código-fonte. Ao publicar em um destino não Azure, é seguro fazer check-in do arquivo *.pubxml*.

Informações confidenciais (como a senha de publicação) são criptografadas em um nível por usuário/computador. Elas são armazenadas no arquivo *Properties/PublishProfiles/{NOME DO PERFIL}.pubxmlUser*. Já que esse arquivo pode armazenar informações confidenciais, o check-in dele não deve ser realizado no controle do código-fonte.

Para obter uma visão geral de como publicar um aplicativo Web do ASP.NET Core, veja [Hospedar e implantar](#). As tarefas e os destinos de MSBuild necessários para publicar um aplicativo ASP.NET Core são de software livre no <https://github.com/aspnet/websdk>.

O `dotnet publish` pode usar os perfis de publicação de pasta, MS Deploy e Kudu:

Pasta (funciona em modo multiplataforma):

```
dotnet publish WebApplication.csproj /p:PublishProfile=<FolderProfileName>
```

MS Deploy (atualmente só funciona no Windows, uma vez que o MS Deploy não é multiplataforma):

```
dotnet publish WebApplication.csproj /p:PublishProfile=<MsDeployProfileName> /p:Password=<DeploymentPassword>
```

Pacote MS Deploy (atualmente só funciona no Windows, uma vez que o MS Deploy não é multiplataforma):

```
dotnet publish WebApplication.csproj /p:PublishProfile=<MsDeployPackageProfileName>
```

Nos exemplos anteriores, **não** passe o `deployOnBuild` para `dotnet publish`.

Para obter mais informações, confira [Microsoft.NET.Sdk.Publish](#).

O `dotnet publish` dá suporte às APIs do Kudu, para publicar no Azure de qualquer plataforma. A publicação do Visual Studio dá suporte às APIs do Kudu, mas ela é compatível com o WebSDK para publicação multiplataforma para o Azure.

Adicione um perfil de publicação à pasta *Properties/PublishProfiles* com o seguinte conteúdo:

```
<Project>
  <PropertyGroup>
    <PublishProtocol>Kudu</PublishProtocol>
    <PublishSiteName>nodewebapp</PublishSiteName>
    <UserName>username</UserName>
    <Password>password</Password>
  </PropertyGroup>
</Project>
```

Execute o seguinte comando para compactar os conteúdos de publicação e publicá-los no Azure usando as APIs do Kudu:

```
dotnet publish /p:PublishProfile=Azure /p:Configuration=Release
```

Defina as seguintes propriedades de MSBuild ao usar um perfil de publicação:

- `DeployOnBuild=true`
- `PublishProfile={PUBLISH PROFILE}`

Ao publicar com um perfil chamado *FolderProfile*, é possível executar qualquer um dos comandos abaixo:

- `dotnet build /p:DeployOnBuild=true /p:PublishProfile=FolderProfile`
- `msbuild /p:DeployOnBuild=true /p:PublishProfile=FolderProfile`

Ao invocar `dotnet build`, ele chama `msbuild` para executar o processo de build e de publicação. Chamar `dotnet build` ou `msbuild` é equivalente ao passar um perfil de pasta. Ao chamar o MSBuild diretamente no Windows, a versão do .NET Framework do MSBuild é usada. O MS Deploy é atualmente limitado a computadores Windows para a publicação. Chamar `dotnet build` em um perfil não de pasta invoca o MSBuild que, por sua vez, usa MS Deploy em perfis não de pasta. Chamar `dotnet build` em um perfil não de pasta invoca o MSBuild (usando MS Deploy) e resulta em uma falha (mesmo quando em execução em uma

plataforma do Windows). Para publicar com um perfil não de pasta, chame o MSBuild diretamente.

A pasta de perfil de publicação a seguir foi criada com o Visual Studio e publica em um compartilhamento de rede:

```
<?xml version="1.0" encoding="utf-8"?>
<!--
This file is used by the publish/package process of your Web project.
You can customize the behavior of this process by editing this
MSBuild file.
-->
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <WebPublishMethod>FileSystem</WebPublishMethod>
    <PublishProvider>FileSystem</PublishProvider>
    <LastUsedBuildConfiguration>Release</LastUsedBuildConfiguration>
    <LastUsedPlatform>Any CPU</LastUsedPlatform>
    <SiteUrlToLaunchAfterPublish />
    <LaunchSiteAfterPublish>True</LaunchSiteAfterPublish>
    <ExcludeApp_Data>False</ExcludeApp_Data>
    <PublishFramework>netcoreapp1.1</PublishFramework>
    <ProjectGuid>c30c453c-312e-40c4-aec9-394a145dee0b</ProjectGuid>
    <publishUrl>\\r8\Release\AdminWeb</publishUrl>
    <DeleteExistingFiles>False</DeleteExistingFiles>
  </PropertyGroup>
</Project>
```

Observe que `<LastUsedBuildConfiguration>` é definido como `Release`. Ao publicar no Visual Studio, o valor da propriedade de configuração `<LastUsedBuildConfiguration>` é definido usando o valor quando o processo de publicação é iniciado. A propriedade de configuração `<LastUsedBuildConfiguration>` é especial e não deve ser substituída em um arquivo do MSBuild importado. Essa propriedade pode ser substituída da linha de comando.

Usando a CLI do .NET Core:

```
dotnet build -c Release /p:DeployOnBuild=true /p:PublishProfile=FolderProfile
```

Usando o MSBuild:

```
msbuild /p:Configuration=Release /p:DeployOnBuild=true /p:PublishProfile=FolderProfile
```

## Publicar em um ponto de extremidade do MSDeploy da linha de comando

O exemplo a seguir usa um aplicativo Web do ASP.NET Core, criado pelo Visual Studio, denominada *AzureWebApp*. Um perfil de publicação dos aplicativos do Azure é adicionado ao Visual Studio. Para obter mais informações sobre como criar um perfil, consulte a seção [Perfis de publicação](#).

Para implantar o aplicativo usando um perfil de publicação, execute o comando `msbuild` de um **Prompt de Comando do Desenvolvedor** do Visual Studio. O prompt de comando está disponível na pasta do *Visual Studio* do menu **Página Inicial** na barra de tarefas do Windows. Para facilitar o acesso, você pode adicionar o prompt de comando ao menu **Ferramentas** no Visual Studio. Para saber mais, confira [Prompt de comando do desenvolvedor para Visual Studio](#).

O MSBuild usa a sintaxe de comando a seguir:

```
msbuild {PATH}  
/p:DeployOnBuild=true  
/p:PublishProfile={PROFILE}  
/p:Username={USERNAME}  
/p:Password={PASSWORD}
```

- {Caminho} – Caminho para o arquivo de projeto do aplicativo.
- {Perfil} – Nome do perfil de publicação.
- {Nome de Usuário} – Nome de usuário do MSDeploy. O {Nome de Usuário} pode ser encontrado no perfil de publicação.
- {Senha} – Senha do MSDeploy. Obtenha a {Senha} do arquivo {Perfil}.PublishSettings. Baixe o arquivo .PublishSettings de uma das seguintes opções:
  - Gerenciador de Soluções: Selecione **Exibição > Cloud Explorer**. Conecte-se à sua assinatura do Azure. Abra **Serviços de Aplicativos**. Clique com o botão direito do mouse no aplicativo. Selecione **Baixar Perfil de Publicação**.
  - Portal do Azure: selecione **Obter perfil de publicação** no painel **Visão geral** de seu aplicativo Web.

O exemplo a seguir usa um perfil de publicação denominado *AzureWebApp – Implantação da Web*:

```
msbuild "AzureWebApp.csproj"  
/p:DeployOnBuild=true  
/p:PublishProfile="AzureWebApp - Web Deploy"  
/p:Username="$AzureWebApp"  
/p:Password="....."
```

Um perfil de publicação também pode ser usado com o comando [dotnet msbuild](#) da CLI do .NET Core em um prompt de comando do Windows:

```
dotnet msbuild "AzureWebApp.csproj"  
/p:DeployOnBuild=true  
/p:PublishProfile="AzureWebApp - Web Deploy"  
/p:Username="$AzureWebApp"  
/p:Password="....."
```

#### NOTE

O comando [dotnet msbuild](#) está disponível em várias plataformas e pode compilar aplicativos ASP.NET Core no macOS e Linux. No entanto, o MSBuild no macOS e Linux não é capaz de implantar um aplicativo no Azure ou em outro ponto de extremidade do MSDeploy. O MSDeploy está disponível apenas no Windows.

## Definir o ambiente

Inclua a propriedade `<EnvironmentName>` no perfil de publicação (`.pubxml`) ou no arquivo de projeto para definir o [ambiente](#) do aplicativo:

```
<PropertyGroup>  
<EnvironmentName>Development</EnvironmentName>  
</PropertyGroup>
```

## Excluir arquivos

Ao publicar aplicativos Web do ASP.NET Core, os artefatos de build e o conteúdo da pasta `wwwroot` são

incluídos. `msbuild` dá suporte a [padrões de caractere curinga](#). Por exemplo, o elemento `<Content>` a seguir exclui todos os arquivos de texto (.txt) da pasta `wwwroot/content` e de todas as subpastas.

```
<ItemGroup>
  <Content Update="wwwroot/content/**/*.*" CopyToPublishDirectory="Never" />
</ItemGroup>
```

A marcação anterior pode ser adicionada a um perfil de publicação ou ao arquivo `.csproj`. Quando adicionada ao arquivo `.csproj`, a regra será adicionada a todos os perfis de publicação no projeto.

O elemento `<MsDeploySkipRules>` a seguir exclui todos os arquivos da pasta `wwwroot/content`:

```
<ItemGroup>
  <MsDeploySkipRules Include="CustomSkipFolder">
    <ObjectName>dirPath</ObjectName>
    <AbsolutePath>wwwroot\\content</AbsolutePath>
  </MsDeploySkipRules>
</ItemGroup>
```

`<MsDeploySkipRules>` não exclui os destinos de *ignorados* do site de implantação. Pastas e arquivos de destino de `<Content>` são excluídos do site de implantação. Por exemplo, suponha que um aplicativo Web implantado tinha os seguintes arquivos:

- `Views/Home/About1.cshtml`
- `Views/Home/About2.cshtml`
- `Views/Home/About3.cshtml`

Nos elementos `<MsDeploySkipRules>` a seguir, esses arquivos não seriam excluídos no site de implantação.

```
<ItemGroup>
  <MsDeploySkipRules Include="CustomSkipFile">
    <ObjectName>filePath</ObjectName>
    <AbsolutePath>Views\\Home\\About1.cshtml</AbsolutePath>
  </MsDeploySkipRules>

  <MsDeploySkipRules Include="CustomSkipFile">
    <ObjectName>filePath</ObjectName>
    <AbsolutePath>Views\\Home\\About2.cshtml</AbsolutePath>
  </MsDeploySkipRules>

  <MsDeploySkipRules Include="CustomSkipFile">
    <ObjectName>filePath</ObjectName>
    <AbsolutePath>Views\\Home\\About3.cshtml</AbsolutePath>
  </MsDeploySkipRules>
</ItemGroup>
```

Os elementos `<MsDeploySkipRules>` anteriores impedem que os arquivos *ignorados* sejam implantados. Ele não excluirá esses arquivos depois que eles forem implantados.

O elemento `<Content>` a seguir exclui os arquivos de destino no site de implantação:

```
<ItemGroup>
  <Content Update="Views/Home/About?.cshtml" CopyToPublishDirectory="Never" />
</ItemGroup>
```

O uso da implantação de linha de comando com o elemento `<Content>` anterior produz a seguinte saída:

```
MSDeployPublish:  
  Starting Web deployment task from source:  
  manifest(C:\Webs\Web1\obj\Release\netcoreapp1.1\PubTmp\Web1.SourceManifest.  
  xml) to Destination: auto().  
  Deleting file (Web11112\Views\Home\About1.cshtml).  
  Deleting file (Web11112\Views\Home\About2.cshtml).  
  Deleting file (Web11112\Views\Home\About3.cshtml).  
  Updating file (Web11112\web.config).  
  Updating file (Web11112\Web1.deps.json).  
  Updating file (Web11112\Web1.dll).  
  Updating file (Web11112\Web1.pdb).  
  Updating file (Web11112\Web1.runtimeconfig.json).  
  Successfully executed Web deployment task.  
  Publish Succeeded.  
 Done Building Project "C:\Webs\Web1\Web1.csproj" (default targets).
```

## Incluir arquivos

A marcação a seguir inclui uma pasta *images* fora do diretório do projeto para a pasta *wwwroot/images* do site de publicação:

```
<ItemGroup>  
  <_CustomFiles Include="$(MSBuildProjectDirectory)/../images/**/*" />  
  <DotnetPublishFiles Include="@(_CustomFiles)">  
    <DestinationRelativePath>wwwroot/images/%(RecursiveDir)%{Filename}%(Extension)</DestinationRelativePath>  
  </DotnetPublishFiles>  
</ItemGroup>
```

A marcação pode ser adicionada ao arquivo *.csproj* ou ao perfil de publicação. Se ela for adicionada ao arquivo *.csproj*, ela será incluída em cada perfil de publicação no projeto.

A marcação realçada a seguir mostra como:

- Copiar um arquivo de fora do projeto para a pasta *wwwroot*.
- Excluir a pasta *wwwroot\Content*.
- Excluir *Views\Home\About2.cshtml*.

```

<?xml version="1.0" encoding="utf-8"?>
<!--
This file is used by the publish/package process of your Web project.
You can customize the behavior of this process by editing this
MSBuild file.
-->
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <WebPublishMethod>FileSystem</WebPublishMethod>
    <PublishProvider>FileSystem</PublishProvider>
    <LastUsedBuildConfiguration>Release</LastUsedBuildConfiguration>
    <LastUsedPlatform>Any CPU</LastUsedPlatform>
    <SiteUrlToLaunchAfterPublish />
    <LaunchSiteAfterPublish>True</LaunchSiteAfterPublish>
    <ExcludeApp_Data>False</ExcludeApp_Data>
    <PublishFramework />
    <ProjectGuid>afa9f185-7ce0-4935-9da1-ab676229d68a</ProjectGuid>
    <publishUrl>bin\Release\PublishOutput</publishUrl>
    <DeleteExistingFiles>False</DeleteExistingFiles>
  </PropertyGroup>
  <ItemGroup>
    <ResolvedFileToPublish Include="..\ReadMe2.MD">
      <RelativePath>wwwroot\ReadMe2.MD</RelativePath>
    </ResolvedFileToPublish>

    <Content Update="wwwroot\Content\**\*" CopyToPublishDirectory="Never" />
    <Content Update="Views\Home\About2.cshtml" CopyToPublishDirectory="Never" />
  </ItemGroup>
</Project>

```

Consulte o [Leia me do WebSDK](#) para obter mais amostras de implantação.

## Executar um destino antes ou depois da publicação

Os destinos `BeforePublish` e `AfterPublish` internos executam um destino antes ou após o destino de publicação. Adicione os elementos a seguir ao perfil de publicação para registrar em log as mensagens de console antes e após a publicação:

```

<Target Name="CustomActionsBeforePublish" BeforeTargets="BeforePublish">
  <Message Text="Inside BeforePublish" Importance="high" />
</Target>
<Target Name="CustomActionsAfterPublish" AfterTargets="AfterPublish">
  <Message Text="Inside AfterPublish" Importance="high" />
</Target>

```

## Publicar em um servidor usando um certificado não confiável

Adicione a propriedade `<AllowUntrustedCertificate>` com um valor `True` ao perfil de publicação:

```

<PropertyGroup>
  <AllowUntrustedCertificate>True</AllowUntrustedCertificate>
</PropertyGroup>

```

## O serviço Kudu

Para exibir os arquivos em uma implantação de aplicativo Web do Serviço de Aplicativo do Azure, use o [serviço Kudu](#). Acrescente o token `scm` ao nome do aplicativo Web. Por exemplo:

URL	RESULTADO
<code>http://mysite.azurewebsites.net/</code>	Aplicativo Web
<code>http://mysite.scm.azurewebsites.net/</code>	Serviço Kudu

Selecione o item de menu [Console de Depuração](#) para exibir, editar, excluir ou adicionar arquivos.

## Recursos adicionais

- A [Implantação da Web](#) (MSDeploy) simplifica a implantação de aplicativos Web e sites da Web em servidores IIS.
- <https://github.com/aspnet/websdk>: problemas de arquivos e recursos de solicitação para implantação.
- [Publicar um aplicativo Web ASP.NET em uma VM do Azure usando o Visual Studio](#)

# Estrutura do diretório do ASP.NET Core

21/01/2019 • 4 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

O diretório *publish* contém os ativos implantáveis do aplicativo produzidos pelo comando [dotnet publish](#). O diretório contém:

- Arquivos de aplicativo
- Arquivos de configuração
- Ativos estáticos
- Pacotes
- Um tempo de execução ([somente implantação autocontida](#))

TIPO DE APLICATIVO	ESTRUTURA DE DIRETÓRIOS
<a href="#">Implantação dependente de estrutura</a>	<ul style="list-style-type: none"><li>• publish†<ul style="list-style-type: none"><li>◦ Log† (opcional, a menos que necessário para receber logs de stdout)</li><li>◦ Exibições† (aplicativos MVC; se as exibições não são pré-compiladas)</li><li>◦ Páginas† (aplicativos de Páginas do Razor ou MVC, se as páginas não são pré-compiladas)</li><li>◦ wwwroot†</li><li>◦ arquivos *.dll</li><li>◦ {NOME DO ASSEMBLY}.deps.json</li><li>◦ {NOME DO ASSEMBLY}.dll</li><li>◦ {NOME DO ASSEMBLY}.pdb</li><li>◦ {NOME DO ASSEMBLY}.Views.dll</li><li>◦ {NOME DO ASSEMBLY}.Views.pdb</li><li>◦ {NOME DO ASSEMBLY}.runtimeconfig.json</li><li>◦ web.config (implantações do IIS)</li></ul></li></ul>
<a href="#">Implantação autossuficiente</a>	<ul style="list-style-type: none"><li>• publish†<ul style="list-style-type: none"><li>◦ Log† (opcional, a menos que necessário para receber logs de stdout)</li><li>◦ Exibições† (aplicativos MVC; se as exibições não são pré-compiladas)</li><li>◦ Páginas† (aplicativos de Páginas do Razor ou MVC, se as páginas não são pré-compiladas)</li><li>◦ wwwroot†</li><li>◦ arquivos *.dll</li><li>◦ {NOME DO ASSEMBLY}.deps.json</li><li>◦ {NOME DO ASSEMBLY}.dll</li><li>◦ {NOME DO ASSEMBLY}.exe</li><li>◦ {NOME DO ASSEMBLY}.pdb</li><li>◦ {NOME DO ASSEMBLY}.Views.dll</li><li>◦ {NOME DO ASSEMBLY}.Views.pdb</li><li>◦ {NOME DO ASSEMBLY}.runtimeconfig.json</li><li>◦ web.config (implantações do IIS)</li></ul></li></ul>

†Indica um diretório

O diretório `publish` representa o *caminho raiz de conteúdo* (também chamado de *caminho base do aplicativo*) da implantação. Qualquer que seja o nome fornecido para o diretório `publish` do aplicativo implantado no servidor, o local dele serve como o caminho físico do servidor para o aplicativo hospedado.

O diretório `wwwroot`, se presente, contém somente ativos estáticos.

Um diretório `Logs` pode ser criado para a implantação usando uma das duas abordagens a seguir:

- Adicione o seguinte elemento `<Target>` ao arquivo de projeto:

```
<Target Name="CreateLogsFolder" AfterTargets="Publish">
  <MakeDir Directories="$(PublishDir)Logs"
    Condition="!Exists('$(PublishDir)Logs')" />
  <WriteLinesToFile File="$(PublishDir)Logs\log"
    Lines="Generated file"
    Overwrite="True"
    Condition="!Exists('$(PublishDir)Logs\log')" />
</Target>
```

O elemento `<MakeDir>` cria uma pasta `Logs` vazia na saída publicada. O elemento usa a propriedade `PublishDir` para determinar o local de destino no qual criar a pasta. Vários métodos de implantação, como a Implantação da Web, ignoram pastas vazias durante a implantação. O elemento `<WriteLinesToFile>` gera um arquivo na pasta `Logs`, o que garante a implantação da pasta no servidor. A criação de pasta usando essa abordagem poderá falhar se o processo de trabalho não tiver acesso de gravação para a pasta de destino.

- Crie fisicamente o diretório `Logs` no servidor na implantação.

O diretório de implantação requer permissões de leitura/execução. O diretório `Logs` requer permissões de leitura/gravação. Diretórios adicionais em que os arquivos são gravados exigem permissões de leitura/gravação.

[Log de stdout do Módulo do ASP.NET Core](#) não exige uma pasta `Logs` na implantação. O módulo é capaz de criar pastas no caminho `stdoutLogFile` quando o arquivo de log é criado. Criar uma pasta `Logs` é útil para o [log de depuração aprimorado do Módulo do ASP.NET Core](#). As pastas no caminho fornecido para o valor `<handlerSetting>` não são criadas automaticamente pelo módulo e devem existir previamente na implantação para permitir que o módulo grave o log de depuração.

## Recursos adicionais

- [dotnet publish](#)
- [Implantação de aplicativos do .NET Core](#)
- [Estruturas de destino](#)
- [Catálogo de RIDs do .NET Core](#)

# Verificações de integridade no ASP.NET Core

10/01/2019 • 35 minutes to read • [Edit Online](#)

Por [Luke Latham](#) e [Glenn Condron](#)

O ASP.NET Core oferece o Middleware de Verificação de Integridade e bibliotecas para relatar a integridade de componentes de infraestrutura do aplicativo.

As verificações de integridade são expostas por um aplicativo como pontos de extremidade HTTP. Os pontos de extremidade de verificação de integridade podem ser configurados para uma variedade de cenários de monitoramento em tempo real:

- As investigações de integridade podem ser usadas por orquestradores de contêineres e balanceadores de carga para verificar o status de um aplicativo. Por exemplo, um orquestrador de contêineres pode responder a uma verificação de integridade com falha interrompendo uma implantação sem interrupção ou reiniciando um contêiner. Um balanceador de carga pode reagir a um aplicativo não íntegro encaminhando o tráfego para fora da instância com falha para uma instância íntegra.
- O uso de memória, disco e outros recursos de servidor físico pode ser monitorado quanto ao status de integridade.
- As verificações de integridade podem testar as dependências do aplicativo, como bancos de dados e pontos de extremidade de serviço externo, para confirmar a disponibilidade e o funcionamento normal.

## [Exibir ou baixar código de exemplo \(como baixar\)](#)

O aplicativo de exemplo inclui exemplos dos cenários descritos neste tópico. Para executar o aplicativo de exemplo para determinado cenário, use o comando `dotnet run` na pasta do projeto em um shell de comando. Confira o arquivo `README.md` do aplicativo de exemplo e as descrições de cenários neste tópico para obter detalhes sobre como usar o aplicativo de exemplo.

## Pré-requisitos

As verificações de integridade são normalmente usadas com um orquestrador de contêineres ou um serviço de monitoramento externo para verificar o status de um aplicativo. Antes de adicionar verificações de integridade a um aplicativo, decida qual sistema de monitoramento será usado. O sistema de monitoramento determina quais tipos de verificações de integridade serão criados e como configurar seus pontos de extremidade.

Referencie o [metapacote Microsoft.AspNetCore.App](#) ou adicione uma referência de pacote ao pacote [Microsoft.AspNetCore.Diagnostics.HealthChecks](#).

O aplicativo de exemplo fornece um código de inicialização para demonstrar verificações de integridade para vários cenários. O cenário [investigação de banco de dados](#) verifica a integridade de uma conexão de banco de dados usando o [BeatPulse](#). O cenário [investigação de DbContext](#) verifica um banco de dados usando um [DbContext](#) do EF Core. Para explorar os cenários de banco de dados, o aplicativo de exemplo:

- Cria um banco de dados e fornece sua cadeia de conexão no arquivo `appsettings.json`.
- Tem as seguintes referências de pacote em seu arquivo de projeto:
  - [AspNetCore.HealthChecks.SqlServer](#)
  - [Microsoft.Extensions.Diagnostics.HealthChecks.EntityFrameworkCore](#)

#### NOTE

O [BeatPulse](#) não é mantido pela Microsoft nem tem o suporte da Microsoft.

Outro cenário de verificação de integridade demonstra como filtrar verificações de integridade para uma porta de gerenciamento. O aplicativo de exemplo exige a criação de um arquivo `Properties/launchSettings.json` que inclui a URL de gerenciamento e a porta de gerenciamento. Para obter mais informações, confira a seção [Filtrar por porta](#).

## Investigação de integridade básica

Para muitos aplicativos, uma configuração básica de investigação de integridade que relata a disponibilidade do aplicativo para processar solicitações (*atividade*) é suficiente para descobrir o status do aplicativo.

A configuração básica regista os serviços de verificação de integridade e chama o Middleware de Verificação de Integridade para responder a um ponto de extremidade de URL com uma resposta de integridade. Por padrão, nenhuma verificação de integridade específica é registrada para testar qualquer dependência ou subsistema específico. O aplicativo é considerado íntegro se consegue responder na URL do ponto de extremidade de integridade. O gravador de resposta padrão grava o status (`HealthStatus`) como uma resposta de texto não criptografado no cliente, indicando um status `HealthStatus.Healthy`, `HealthStatus.Degraded` ou `HealthStatus.Unhealthy`.

Registre os serviços de verificação de integridade com `AddHealthChecks` em `Startup.ConfigureServices`. Adicione o Middleware de Verificação de Integridade com `UseHealthChecks` ao pipeline de processamento de solicitação de `Startup.Configure`.

No aplicativo de exemplo, o ponto de extremidade de verificação de integridade é criado em `/health` (`BasicStartup.cs`):

```
public class BasicStartup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddHealthChecks();
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.UseHealthChecks("/health");

        app.Run(async (context) =>
        {
            await context.Response.WriteAsync(
                "Navigate to /health to see the health status.");
        });
    }
}
```

Para executar o cenário de configuração básica usando o aplicativo de exemplo, execute o seguinte comando na pasta do projeto em um shell de comando:

```
dotnet run --scenario basic
```

## Exemplo do Docker

O [Docker](#) oferece uma diretiva `HEALTHCHECK` interna que pode ser usada para verificar o status de um aplicativo que usa a configuração básica de verificação de integridade:

```
HEALTHCHECK CMD curl --fail http://localhost:5000/health || exit
```

## Criar verificações de integridade

As verificações de integridade são criadas pela implementação da interface `IHealthCheck`. O método `IHealthCheck.CheckHealthAsync` retorna um `Task<HealthCheckResult>` que indica a integridade como `Healthy`, `Degraded` ou `Unhealthy`. O resultado é gravado como uma resposta de texto sem formatação com um código de status configurável (a configuração é descrita na seção [Opções de verificação de integridade](#)). `HealthCheckResult` também pode retornar pares chave-valor opcionais.

### Verificação de integridade de exemplo

A seguinte classe `ExampleHealthCheck` demonstra o layout de uma verificação de integridade:

```
public class ExampleHealthCheck : IHealthCheck
{
    public ExampleHealthCheck()
    {
        // Use dependency injection (DI) to supply any required services to the
        // health check.
    }

    public Task<HealthCheckResult> CheckHealthAsync(
        HealthCheckContext context,
        CancellationToken cancellationToken = default(CancellationToken))
    {
        // Execute health check logic here. This example sets a dummy
        // variable to true.
        var healthCheckResultHealthy = true;

        if (healthCheckResultHealthy)
        {
            return Task.FromResult(
                HealthCheckResult.Healthy("The check indicates a healthy result."));
        }

        return Task.FromResult(
            HealthCheckResult.Unhealthy("The check indicates an unhealthy result."));
    }
}
```

### Registrar os serviços de verificação de integridade

O tipo `ExampleHealthCheck` é adicionado aos serviços de verificação de integridade com `AddCheck`:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddHealthChecks()
        .AddCheck<ExampleHealthCheck>("example_health_check");
}
```

A sobrecarga `AddCheck` mostrada no exemplo a seguir define o status de falha (`HealthStatus`) como relatório quando a verificação de integridade relata uma falha. Se o status de falha é definido como `null` (padrão), `HealthStatus.Unhealthy` é relatado. Essa sobrecarga é um cenário útil para autores de biblioteca, em que o status de falha indicado pela biblioteca é imposto pelo aplicativo quando uma falha de verificação de integridade ocorre se a implementação da verificação de integridade respeita a configuração.

*Marcas* podem ser usadas para filtrar as verificações de integridade (descritas posteriormente na seção [Filtrar](#)

verificações de integridade).

```
services.AddHealthChecks()
    .AddCheck<ExampleHealthCheck>(
        "example_health_check",
        failureStatus: HealthStatus.Degraded,
        tags: new[] { "example" });
```

`AddCheck` também pode executar uma função lambda. No seguinte exemplo, o nome da verificação de integridade é especificado como `Example` e a verificação sempre retorna um estado íntegro:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddHealthChecks()
        .AddCheck("Example", () =>
            HealthCheckResult.Healthy("Example is OK!"), tags: new[] { "example" })
}
```

## Usar o Middleware de Verificações de Integridade

Em `Startup.Configure`, chame `UseHealthChecks` do pipeline de processamento com a URL de ponto de extremidade ou o caminho relativo:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseHealthChecks("/health");
}
```

Se as verificações de integridade precisarem escutar uma porta específica, use uma sobrecarga igual a `UseHealthChecks` para definir a porta (descrito posteriormente na seção [Filtrar por porta](#)):

```
app.UseHealthChecks("/health", port: 8000);
```

O Middleware de Verificações de Integridade é um *middleware de terminal* no pipeline de processamento de solicitação do aplicativo. O primeiro ponto de extremidade de verificação de integridade encontrado que seja uma correspondência exata da URL de solicitação é executado e causa curto-circuito no restante do pipeline de middleware. Quando ocorre o curto-circuito, nenhum middleware após a verificação de integridade correspondente é executado.

## Opções de verificação de integridade

`HealthCheckOptions` oferece uma oportunidade de personalizar o comportamento de verificação de integridade:

- [Filtrar verificações de integridade](#)
- [Personalizar o código de status HTTP](#)
- [Suprimir os cabeçalhos de cache](#)
- [Personalizar a saída](#)

### Filtrar verificações de integridade

Por padrão, o Middleware de Verificação de Integridade executa todas as verificações de integridade registradas. Para executar um subconjunto das verificações de integridade, forneça uma função que retorne um booleano para a opção `Predicate`. No seguinte exemplo, a verificação de integridade `Bar` é filtrada por sua marca (`bar_tag`) na instrução condicional da função, em que `true` só é retornado se a propriedade `Tag` da verificação de integridade corresponde a `foo_tag` ou `baz_tag`:

```

using System.Threading.Tasks;
using Microsoft.AspNetCore.Diagnostics.HealthChecks;
using Microsoft.Extensions.Diagnostics.HealthChecks;

public void ConfigureServices(IServiceCollection services)
{
    services.AddHealthChecks()
        .AddCheck("Foo", () =>
            HealthCheckResult.Healthy("Foo is OK!"), tags: new[] { "foo_tag" })
        .AddCheck("Bar", () =>
            HealthCheckResult.Unhealthy("Bar is unhealthy!"), tags: new[] { "bar_tag" })
        .AddCheck("Baz", () =>
            HealthCheckResult.Healthy("Baz is OK!"), tags: new[] { "baz_tag" });
}

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseHealthChecks("/health", new HealthCheckOptions()
    {
        // Filter out the 'Bar' health check. Only Foo and Baz execute.
        Predicate = (check) => check.Tags.Contains("foo_tag") ||
            check.Tags.Contains("baz_tag")
    });
}

```

## Personalizar o código de status HTTP

Use `ResultStatusCodes` para personalizar o mapeamento de status da integridade para códigos de status HTTP. As atribuições `StatusCodes` a seguir são os valores padrão usados pelo middleware. Altere os valores do código de status de acordo com suas necessidades.

```

using Microsoft.AspNetCore.Diagnostics.HealthChecks;
using Microsoft.Extensions.Diagnostics.HealthChecks;

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseHealthChecks("/health", new HealthCheckOptions()
    {
        // The following StatusCodes are the default assignments for
        // the HealthStatus properties.
        ResultStatusCodes =
        {
            [HealthStatus.Healthy] = StatusCodes.Status200OK,
            [HealthStatus.Degraded] = StatusCodes.Status200OK,
            [HealthStatus.Unhealthy] = StatusCodes.Status503ServiceUnavailable
        }
    });
}

```

## Suprimir os cabeçalhos de cache

`AllowCachingResponses` controla se o Middleware de Verificação de Integridade adiciona cabeçalhos HTTP a uma resposta de investigação para prevenir o cache de resposta. Se o valor é `false` (padrão), o middleware define ou substitui os cabeçalhos `Cache-Control`, `Expires` e `Pragma` para prevenir o cache de resposta. Se o valor é `true`, o middleware não modifica os cabeçalhos de cache da resposta.

```

using Microsoft.AspNetCore.Diagnostics.HealthChecks;
using Microsoft.Extensions.Diagnostics.HealthChecks;

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseHealthChecks("/health", new HealthCheckOptions()
    {
        // The default value is false.
        AllowCachingResponses = false
    });
}

```

## Personalizar a saída

A opção `ResponseWriter` obtém ou define um representante usado para gravar a resposta. O representante padrão grava uma resposta mínima de texto sem formatação com o valor de cadeia de caracteres `HealthReport.Status`.

```

using Microsoft.AspNetCore.Diagnostics.HealthChecks;
using Microsoft.Extensions.Diagnostics.HealthChecks;

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseHealthChecks("/health", new HealthCheckOptions()
    {
        // WriteResponse is a delegate used to write the response.
        ResponseWriter = WriteResponse
    });
}

private static Task WriteResponse(HttpContext httpContext,
    HealthReport result)
{
    httpContext.Response.ContentType = "application/json";

    var json = new JObject(
        new JProperty("status", result.Status.ToString()),
        new JProperty("results", new JObject(result.Entries.Select(pair =>
            new JProperty(pair.Key, new JObject(
                new JProperty("status", pair.Value.Status.ToString()),
                new JProperty("description", pair.Value.Description),
                new JProperty("data", new JObject(pair.Value.Data.Select(
                    p => new JProperty(p.Key, p.Value))))))))));
    return httpContext.Response.WriteAsync(
        json.ToString(Formatting.Indented));
}

```

## Investigação de banco de dados

Uma verificação de integridade pode especificar uma consulta de banco de dados a ser executada como um teste booleano para indicar se o banco de dados está respondendo normalmente.

O aplicativo de exemplo usa o [BeatPulse](#), uma biblioteca de verificação de integridade para aplicativos ASP.NET Core, para executar uma verificação de integridade em um banco de dados do SQL Server. O BeatPulse executa uma consulta `SELECT 1` no banco de dados para confirmar se a conexão ao banco de dados está íntegra.

### WARNING

Ao verificar uma conexão de banco de dados com uma consulta, escolha uma consulta que retorne rapidamente. A abordagem de consulta corre o risco de sobrecarregar o banco de dados e prejudicar o desempenho. Na maioria dos casos, a execução de uma consulta de teste não é necessária. É suficiente apenas estabelecer uma conexão bem-sucedida ao banco de dados. Se você achar necessário executar uma consulta, escolha uma consulta SELECT simples, como `SELECT 1`.

Para usar a biblioteca do BeatPulse, inclua uma referência de pacote a `AspNetCore.HealthChecks.SqlServer`.

Forneça uma cadeia de conexão de banco de dados válida no arquivo `appsettings.json` do aplicativo de exemplo. O aplicativo usa um banco de dados do SQL Server chamado `HealthCheckSample`:

```
{  
  "ConnectionStrings": {  
    "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=HealthCheckSample;Trusted_Connection=True;MultipleActiveResultSets=true;Conn  
ectRetryCount=0"  
  },  
  "Logging": {  
    "LogLevel": {  
      "Default": "Debug"  
    },  
    "Console": {  
      "IncludeScopes": "true"  
    }  
  }  
}
```

Registre os serviços de verificação de integridade com `AddHealthChecks` em `Startup.ConfigureServices`. O aplicativo de exemplo chama o método `AddSqlServer` do BeatPulse com a cadeia de conexão do banco de dados (`DbHealthStartup.cs`):

```
public void ConfigureServices(IServiceCollection services)  
{  
  services.AddHealthChecks()  
    .AddSqlServer(Configuration["ConnectionStrings:DefaultConnection"]);  
}
```

Chame o Middleware de Verificação de Integridade do pipeline de processamento de aplicativo em `Startup.Configure`:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)  
{  
  app.UseHealthChecks("/health");  
}
```

Para executar o cenário de investigação de banco de dados usando o aplicativo de exemplo, execute o seguinte comando na pasta do projeto em um shell de comando:

```
dotnet run --scenario db
```

### NOTE

O [BeatPulse](#) não é mantido pela Microsoft nem tem o suporte da Microsoft.

# Investigação de DbContext do Entity Framework Core

A verificação `DbContext` confirma se o aplicativo pode se comunicar com o banco de dados configurado para um `DbContext` do EF Core. A verificação `DbContext` é compatível em aplicativos que:

- Usam o [EF \(Entity Framework\) Core](#).
- Incluem uma referência de pacote para [Microsoft.Extensions.Diagnostics.HealthChecks.EntityFrameworkCore](#).

`AddDbContextCheck<TContext>` registra uma verificação de integridade para um `DbContext`. O `DbContext` é fornecido como o `TContext` para o método. Uma sobrecarga está disponível para configurar o status de falha, marcas e uma consulta de teste personalizada.

Por padrão:

- O `DbContextHealthCheck` chama o método `CanConnectAsync` do EF Core. Você pode personalizar qual operação é executada durante a verificação de integridade usando sobrecargas do método `AddDbContextCheck`.
- O nome da verificação de integridade é o nome do tipo `TContext`.

No aplicativo de exemplo, `AppDbContext` é fornecido para `AddDbContextCheck` e registrado como um serviço em `Startup.ConfigureServices`.

*DbContextHealthStartup.cs*:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddHealthChecks()
        .AddDbContextCheck<AppDbContext>();

    services.AddDbContext<AppDbContext>(options =>
    {
        options.UseSqlServer(
            Configuration["ConnectionStrings:DefaultConnection"]);
    });
}
```

No aplicativo de exemplo, `UseHealthChecks` adiciona o Middleware de Verificação de Integridade em `Startup.Configure`.

*DbContextHealthStartup.cs*:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseHealthChecks("/health");
```

Para executar o cenário de investigação `DbContext` usando o aplicativo de exemplo, confirme se o banco de dados especificado pela cadeia de conexão não existe na instância do SQL Server. Se o banco de dados existir, exclua-o.

Execute o seguinte comando na pasta do projeto em um shell de comando:

```
dotnet run --scenario dbcontext
```

Depois que o aplicativo estiver em execução, verifique o status da integridade fazendo uma solicitação para o ponto de extremidade `/health` em um navegador. O banco de dados e `AppDbContext` não existem e, portanto, o aplicativo fornece a seguinte resposta:

Unhealthy

Dispare o aplicativo de exemplo para criar o banco de dados. Faça uma solicitação para `/createdatabase`. O aplicativo responde:

```
Creating the database...
Done!
Navigate to /health to see the health status.
```

Faça uma solicitação ao ponto de extremidade `/health`. O banco de dados e o contexto existem e, portanto, o aplicativo responde:

Healthy

Dispare o aplicativo de exemplo para excluir o banco de dados. Faça uma solicitação para `/deletedatabase`. O aplicativo responde:

```
Deleting the database...
Done!
Navigate to /health to see the health status.
```

Faça uma solicitação ao ponto de extremidade `/health`. O aplicativo fornece uma resposta não íntegra:

Unhealthy

## Investigações de preparação e atividade separadas

Em alguns cenários de hospedagem, é usado um par de verificações de integridade que distingue dois estados de aplicativo:

- O aplicativo está funcionando, mas ainda não está pronto para receber solicitações. Esse estado é a *preparação* do aplicativo.
- O aplicativo está funcionando e respondendo a solicitações. Esse estado é a *atividade* do aplicativo.

A verificação de preparação geralmente executa um conjunto mais amplo e demorado de verificações para determinar se todos os recursos e subsistemas do aplicativo estão disponíveis. Uma verificação de atividade apenas executa uma verificação rápida para determinar se o aplicativo está disponível para processar solicitações. Depois que o aplicativo é aprovado na verificação de preparação, não há nenhuma necessidade de sobreregar o aplicativo com o conjunto caro de verificações de preparação – as verificações adicionais exigem somente a verificação de atividade.

O aplicativo de exemplo contém uma verificação de integridade para relatar a conclusão da tarefa de inicialização de execução longa em um [Serviço Hospedado](#). A `StartupHostedServiceHealthCheck` expõe uma propriedade `StartupTaskCompleted`, que o serviço hospedado poderá definir como `true` quando sua tarefa de execução longa estiver concluída (`StartupHostedServiceHealthCheck.cs`):

```
public class StartupHostedServiceHealthCheck : IHealthCheck
{
    public string Name => "slow_dependency_check";

    public bool StartupTaskCompleted { get; set; } = false;

    public Task<HealthCheckResult> CheckHealthAsync(
        HealthCheckContext context,
        CancellationToken cancellationToken = default(CancellationToken))
    {
        if (StartupTaskCompleted)
        {
            return Task.FromResult(
                HealthCheckResult.Healthy("The startup task is finished."));
        }

        return Task.FromResult(
            HealthCheckResult.Unhealthy("The startup task is still running."));
    }
}
```

A tarefa em segundo plano de execução longa é iniciada por um [Serviço Hospedado](#) (*Services/StartupHostedService*). Após a conclusão da tarefa,

`StartupHostedServiceHealthCheck.StartupTaskCompleted` é definido como `true`:

```

public class StartupHostedService : IHostedService, IDisposable
{
    private readonly int _delaySeconds = 15;
    private readonly ILogger _logger;
    private readonly StartupHostedServiceHealthCheck _startupHostedServiceHealthCheck;

    public StartupHostedService(ILogger<StartupHostedService> logger,
        StartupHostedServiceHealthCheck startupHostedServiceHealthCheck)
    {
        _logger = logger;
        _startupHostedServiceHealthCheck = startupHostedServiceHealthCheck;
    }

    public Task StartAsync(CancellationToken cancellationToken)
    {
        _logger.LogInformation($"Startup Background Service is starting.");

        // Simulate the effect of a long-running startup task.
        Task.Run(async () =>
        {
            await Task.Delay(_delaySeconds * 1000);

            _startupHostedServiceHealthCheck.StartupTaskCompleted = true;

            _logger.LogInformation($"Startup Background Service has started.");
        });
    }

    return Task.CompletedTask;
}

public Task StopAsync(CancellationToken cancellationToken)
{
    _logger.LogInformation("Startup Background Service is stopping.");

    return Task.CompletedTask;
}

public void Dispose()
{
}
}

```

A verificação de integridade é registrada em `AddCheck` no `Startup.ConfigureServices` juntamente com o serviço hospedado. Como o serviço hospedado precisa definir a propriedade na verificação de integridade, a verificação de integridade também é registrada no contêiner de serviço (*LiveabilityProbeStartup.cs*):

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddHostedService<StartupHostedService>();
    services.AddSingleton<StartupHostedServiceHealthCheck>();

    services.AddHealthChecks()
        .AddCheck<StartupHostedServiceHealthCheck>(
            "hosted_service_startup",
            failureStatus: HealthStatus.Degraded,
            tags: new[] { "ready" });
}

```

Chame o Middleware de Verificação de Integridade no pipeline de processamento de aplicativo em `Startup.Configure`. No aplicativo de exemplo, os pontos de extremidade de verificação de integridade são criados em `/health/ready` para a verificação de preparação e em `/health/live` para a verificação de atividade. A verificação de preparação filtra as verificações de integridade para a verificação de integridade com a marca

`ready`. A verificação de atividade filtra a `StartupHostedServiceHealthCheck` retornando `false` no `HealthCheckOptions.Predicate` (para obter mais informações, confira [Filtrar verificações de integridade](#)):

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    // The readiness check uses all registered checks with the 'ready' tag.
    app.UseHealthChecks("/health/ready", new HealthCheckOptions()
    {
        Predicate = (check) => check.Tags.Contains("ready"),
    });

    app.UseHealthChecks("/health/live", new HealthCheckOptions()
    {
        // Exclude all checks and return a 200-Ok.
        Predicate = (_) => false
    });
}
```

Para executar o cenário de configuração de preparação/atividade usando o aplicativo de exemplo, execute o seguinte comando na pasta do projeto em um shell de comando:

```
dotnet run --scenario liveness
```

Em um navegador, visite `/health/ready` várias vezes até terem decorrido 15 segundos. A verificação de integridade relata `Unhealthy` para os primeiros 15 segundos. Após 15 segundos, o ponto de extremidade relata `Healthy`, que reflete a conclusão da tarefa de execução longa pelo serviço hospedado.

### Exemplo do Kubernetes

O uso de verificações de preparação e atividade separadas é útil em um ambiente como o [Kubernetes](#). No Kubernetes, um aplicativo pode precisar executar um trabalho de inicialização demorado antes de aceitar solicitações, como um teste da disponibilidade do banco de dados subjacente. O uso de verificações separadas permite que o orquestrador distinga se o aplicativo está funcionando, mas ainda não está pronto, ou se o aplicativo falhou ao ser iniciado. Para obter mais informações sobre as investigações de preparação e atividade no Kubernetes, confira [Configurar investigações de preparação e atividade](#) na documentação do Kubernetes.

O seguinte exemplo demonstra uma configuração de investigação de preparação do Kubernetes:

```
spec:
  template:
    spec:
      readinessProbe:
        # an http probe
        httpGet:
          path: /health/ready
          port: 80
        # length of time to wait for a pod to initialize
        # after pod startup, before applying health checking
        initialDelaySeconds: 30
        timeoutSeconds: 1
      ports:
        - containerPort: 80
```

## Investigação baseada em métrica com um gravador de resposta personalizada

O aplicativo de exemplo demonstra uma verificação de integridade da memória com um gravador de resposta personalizada.

`MemoryHealthCheck` relata um estado degradado se o aplicativo usa mais de determinado limite de memória (1 GB no aplicativo de exemplo). O `HealthCheckResult` inclui informações de GC (Coletor de Lixo) para o aplicativo (`MemoryHealthCheck.cs`):

```
public class MemoryHealthCheck : IHealthCheck
{
    private readonly IOptionsMonitor<MemoryCheckOptions> _options;

    public MemoryHealthCheck(IOptionsMonitor<MemoryCheckOptions> options)
    {
        _options = options;
    }

    public string Name => "memory_check";

    public Task<HealthCheckResult> CheckHealthAsync(
        HealthCheckContext context,
        CancellationToken cancellationToken = default(CancellationToken))
    {
        var options = _options.Get(context.Registration.Name);

        // Include GC information in the reported diagnostics.
        var allocated = GC.GetTotalMemory(forceFullCollection: false);
        var data = new Dictionary<string, object>()
        {
            { "AllocatedBytes", allocated },
            { "Gen0Collections", GC.CollectionCount(0) },
            { "Gen1Collections", GC.CollectionCount(1) },
            { "Gen2Collections", GC.CollectionCount(2) },
        };

        var status = (allocated < options.Threshold) ?
            HealthStatus.Healthy : HealthStatus.Unhealthy;

        return Task.FromResult(new HealthCheckResult(
            status,
            description: "Reports degraded status if allocated bytes " +
                $">= {options.Threshold} bytes.",
            exception: null,
            data: data));
    }
}
```

Registre os serviços de verificação de integridade com `AddHealthChecks` em `Startup.ConfigureServices`. Em vez de permitir a verificação de integridade passando-a para `AddCheck`, a `MemoryHealthCheck` é registrada como um serviço. Todos os serviços registrados da `IHealthCheck` estão disponíveis para os serviços de verificação de integridade e middleware. Recomendamos registrar os serviços de verificação de integridade como serviços Singleton.

`CustomWriterStartup.cs`:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddHealthChecks()
        .AddMemoryHealthCheck("memory");
}
```

Chame o Middleware de Verificação de Integridade no pipeline de processamento de aplicativo em `Startup.Configure`. Um representante `WriteResponse` é fornecido para a propriedade `ResponseWriter` para gerar uma resposta JSON personalizada quando a verificação de integridade é executada:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseHealthChecks("/health", new HealthCheckOptions()
    {
        // This custom writer formats the detailed status as JSON.
        ResponseWriter = WriteResponse
    });
}
```

O método `WriteResponse` formata o `CompositeHealthCheckResult` em um objeto JSON e produz a saída JSON para a resposta da verificação de integridade:

```
private static Task WriteResponse(HttpContext httpContext,
    HealthReport result)
{
    httpContext.Response.ContentType = "application/json";

    var json = new JObject(
        new JProperty("status", result.Status.ToString()),
        new JProperty("results", new JObject(result.Entries.Select(pair =>
            new JObject(pair.Key, new JObject(
                new JProperty("status", pair.Value.Status.ToString()),
                new JProperty("description", pair.Value.Description),
                new JProperty("data", new JObject(pair.Value.Data.Select(
                    p => new JProperty(p.Key, p.Value))))))))));
    return httpContext.Response.WriteAsync(
        json.ToString(Formatting.Indented));
}
```

Para executar a investigação baseada em métrica com a saída do gravador de resposta personalizada usando o aplicativo de exemplo, execute o seguinte comando na pasta do projeto em um shell de comando:

```
dotnet run --scenario writer
```

#### NOTE

O [BeatPulse](#) inclui cenários de verificação de integridade baseada em métrica, incluindo verificações de atividade de valor máximo e armazenamento em disco.

O [BeatPulse](#) não é mantido pela Microsoft nem tem o suporte da Microsoft.

## Filtrar por porta

A chamada a `UseHealthChecks` com uma porta restringe as solicitações de verificação de integridade à porta especificada. Isso normalmente é usado em um ambiente de contêiner para expor uma porta para os serviços de monitoramento.

O aplicativo de exemplo configura a porta usando o [Provedor de Configuração de Variáveis de Ambiente](#). A porta é definida no arquivo `launchSettings.json` e passada para o provedor de configuração por meio de uma variável de ambiente. Você também precisa configurar o servidor para escutar as solicitações na porta de gerenciamento.

Para usar o aplicativo de exemplo para demonstrar a configuração de porta de gerenciamento, crie o arquivo `launchSettings.json` em uma pasta `Properties`.

O arquivo `launchSettings.json` a seguir não está incluído nos arquivos de projeto do aplicativo de exemplo e precisa ser criado manualmente.

`Properties/launchSettings.json`:

```
{  
  "profiles": {  
    "SampleApp": {  
      "commandName": "Project",  
      "commandLineArgs": "",  
      "launchBrowser": true,  
      "environmentVariables": {  
        "ASPNETCORE_ENVIRONMENT": "Development",  
        "ASPNETCORE_URLS": "http://localhost:5000/;http://localhost:5001/",  
        "ASPNETCORE_MANAGEMENTPORT": "5001"  
      },  
      "applicationUrl": "http://localhost:5000/"  
    }  
  }  
}
```

Registre os serviços de verificação de integridade com `AddHealthChecks` em `Startup.ConfigureServices`. A chamada a `UseHealthChecks` especifica a porta de gerenciamento (*ManagementPortStartup.cs*):

```
public class ManagementPortStartup  
{  
  public ManagementPortStartup(IConfiguration configuration)  
  {  
    Configuration = configuration;  
  }  
  
  public IConfiguration Configuration { get; }  
  
  public void ConfigureServices(IServiceCollection services)  
  {  
    services.AddHealthChecks();  
  }  
  
  public void Configure(IApplicationBuilder app, IHostingEnvironment env)  
  {  
  
    app.UseHealthChecks("/health", port: Configuration["ManagementPort"]);  
  
    app.Run(async (context) =>  
    {  
      await context.Response.WriteAsync(  
        "Navigate to " +  
        $"http://localhost:{Configuration["ManagementPort"]}/health " +  
        "to see the health status.");  
    });  
  }  
}
```

#### NOTE

Evite a criação do arquivo `launchSettings.json` no aplicativo de exemplo definindo as URLs e a porta de gerenciamento explicitamente no código. Em `Program.cs`, em que o `WebHostBuilder` é criado, adicione uma chamada a `UseUrls` e forneça o ponto de extremidade da resposta normal do aplicativo e o ponto de extremidade da porta de gerenciamento. Em `ManagementPortStartup.cs`, em que `useHealthChecks` é chamado, especifique a porta de gerenciamento explicitamente.

`Program.cs`:

```
return new WebHostBuilder()
    .UseConfiguration(config)
    .UseUrls("http://localhost:5000/;http://localhost:5001/")
    .ConfigureLogging(builder =>
{
    builder.SetMinimumLevel(LogLevel.Trace);
    builder.AddConfiguration(config);
    builder.AddConsole();
})
    .UseKestrel()
    .UseStartup(startupType)
    .Build();
```

`ManagementPortStartup.cs`:

```
app.UseHealthChecks("/health", port: 5001);
```

Para executar o cenário de configuração de porta de gerenciamento usando o aplicativo de exemplo, execute o seguinte comando na pasta do projeto em um shell de comando:

```
dotnet run --scenario port
```

## Distribuir uma biblioteca de verificação de integridade

Para distribuir uma verificação de integridade como uma biblioteca:

1. Escreva uma verificação de integridade que implementa a interface `IHealthCheck` como uma classe autônoma. A classe pode depender da [DI \(injeção de dependência\)](#), da ativação de tipo e das [opções nomeadas](#) para acessar os dados de configuração.

```

using System;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Extensions.Diagnostics.HealthChecks;

namespace SampleApp
{
    public class ExampleHealthCheck : IHealthCheck
    {
        private readonly string _data1;
        private readonly int? _data2;

        public ExampleHealthCheck(string data1, int? data2)
        {
            _data1 = data1 ?? throw new ArgumentNullException(nameof(data1));
            _data2 = data2 ?? throw new ArgumentNullException(nameof(data2));
        }

        public async Task<HealthCheckResult> CheckHealthAsync(
            HealthCheckContext context, CancellationToken cancellationToken)
        {
            try
            {
                // Health check logic
                //
                // data1 and data2 are used in the method to
                // run the probe's health check logic.

                // Assume that it's possible for this health check
                // to throw an AccessViolationException.

                return HealthCheckResult.Healthy();
            }
            catch (AccessViolationException ex)
            {
                return new HealthCheckResult(
                    context.Registration.FailureStatus,
                    description: "An access violation occurred during the check.",
                    exception: ex,
                    data: null);
            }
        }
    }
}

```

2. Escreva um método de extensão com parâmetros que o aplicativo de consumo chama em seu método `Startup.Configure`. No seguinte exemplo, suponha a seguinte assinatura de método de verificação de integridade:

```
ExampleHealthCheck(string, string, int )
```

A assinatura anterior indica que a `ExampleHealthCheck` exige dados adicionais para processar a lógica de investigação de verificação de integridade. Os dados são fornecidos para o representante usado para criar a instância de verificação de integridade quando a verificação de integridade é registrada em um método de extensão. No seguinte exemplo, o chamador especifica itens opcionais:

- nome da verificação de integridade (`name`). Se `null`, `example_health_check` é usado.
- ponto de dados de cadeia de caracteres para a verificação de integridade (`data1`).
- ponto de dados de inteiro para a verificação de integridade (`data2`). Se `null`, `1` é usado.
- status de falha (`HealthStatus`). O padrão é `null`. Se `null`, `HealthStatus.Unhealthy` é relatado para um status de falha.

- marcas (`IEnumerable<string>`).

```
using System.Collections.Generic;
using Microsoft.Extensions.Diagnostics.HealthChecks;

public static class ExampleHealthCheckBuilderExtensions
{
    const string NAME = "example_health_check";

    public static IHealthChecksBuilder AddExampleHealthCheck(
        this IHealthChecksBuilder builder,
        string name = default,
        string data1,
        int data2 = 1,
        HealthStatus? failureStatus = default,
        IEnumerable<string> tags = default)
    {
        return builder.Add(new HealthCheckRegistration(
            name ?? NAME,
            sp => new ExampleHealthCheck(data1, data2),
            failureStatus,
            tags));
    }
}
```

## Publicador de Verificação de Integridade

Quando um `IHealthCheckPublisher` é adicionado ao contêiner de serviços, o sistema de verificação de integridade periodicamente executa sua verificação de integridade e chama `PublishAsync` com o resultado. Isso é útil em um cenário de sistema de monitoramento de integridade baseada em push que espera que cada processo chame o sistema de monitoramento periodicamente para determinar a integridade.

A interface `IHealthCheckPublisher` tem um único método:

```
Task PublishAsync(HealthReport report, CancellationToken cancellationToken);
```

### NOTE

O [BeatPulse](#) inclui publicadores para vários sistemas, incluindo o [Application Insights](#).

O [BeatPulse](#) não é mantido pela Microsoft nem tem o suporte da Microsoft.

# Hospedar e implantar Componentes Razor

05/02/2019 • 25 minutes to read • [Edit Online](#)

Por [Luke Latham](#), [Rainer Stropek](#) e [Daniel Roth](#)

## NOTE

O Razor Components do ASP.NET Core é compatível com ASP.NET Core 3.0 ou posterior.

Blazor é uma estrutura da Web sem suporte e experimental que não deve ser usada para cargas de trabalho de produção no momento.

## Publique o aplicativo

Os aplicativos são publicados para implantação na configuração de versão com o comando `dotnet publish`. Um IDE pode manipular a execução do comando `dotnet publish` automaticamente usando recursos de publicação internos, para que não seja necessário executar o comando manualmente usando um prompt de comando, dependendo das ferramentas de desenvolvimento em uso.

```
dotnet publish -c Release
```

O `dotnet publish` dispara uma [restauração](#) das dependências do projeto e [compila](#) o projeto antes de criar os ativos para implantação. Como parte do processo de build, os assemblies e métodos não usados são removidos para reduzir o tamanho de download do aplicativo e os tempos de carregamento. A implantação é criada na pasta `/bin/Release/<target-framework>/publish`.

Os ativos na pasta `publish` são implantados no servidor Web. A implantação pode ser um processo manual ou automatizado, dependendo das ferramentas de desenvolvimento em uso.

## Valores de configuração do host

Os aplicativos dos Componentes Razor que usam o [modelo de hospedagem do lado do servidor](#) podem aceitar os [valores de configuração do host da Web](#).

Os aplicativos do Blazor que usam o [modelo de hospedagem do lado do cliente](#) podem aceitar os seguintes valores de configuração do host como argumentos de linha de comando no tempo de execução no ambiente de desenvolvimento.

### Raiz do conteúdo

O argumento `--contentroot` define o caminho absoluto para o diretório que contém os arquivos de conteúdo do aplicativo.

- Passe o argumento ao executar o aplicativo localmente em um prompt de comando. No diretório do aplicativo, execute:

```
dotnet run --contentroot=/<content-root>
```

- Adicione uma entrada ao arquivo `launchSettings.json` do aplicativo no perfil do **IIS Express**. Essa configuração é obtida ao executar o aplicativo com o Depurador do Visual Studio e ao executar o aplicativo em um prompt de comando com `dotnet run`.

```
"commandLineArgs": "--contentroot=<content-root>"
```

- No Visual Studio, especifique o argumento em **Propriedades > Depurar > Argumentos de aplicativo**. A configuração do argumento na página de propriedades do Visual Studio adiciona o argumento ao arquivo *launchSettings.json*.

```
--contentroot=<content-root>
```

## Caminho base

O argumento `--pathbase` define o caminho base do aplicativo para um aplicativo executado localmente com um caminho virtual não raiz (a tag `href` `<base>` é definida como um caminho diferente de `/` para preparo e produção). Para obter mais informações, confira a seção [Caminho base do aplicativo](#).

### IMPORTANT

Ao contrário do caminho fornecido ao `href` da tag `<base>`, não inclua uma barra à direita (`/`) ao passar o valor do argumento `--pathbase`. Se o caminho base do aplicativo for fornecido na tag `<base>` como `<base href="/CoolApp/" />` (incluir uma barra à direita), passe o valor do argumento de linha de comando como `--pathbase=/CoolApp` (nenhuma barra à direita).

- Passe o argumento ao executar o aplicativo localmente em um prompt de comando. No diretório do aplicativo, execute:

```
dotnet run --pathbase=<virtual-path>
```

- Adicione uma entrada ao arquivo *launchSettings.json* do aplicativo no perfil do **IIS Express**. Essa configuração é obtida ao executar o aplicativo com o Depurador do Visual Studio e ao executar o aplicativo em um prompt de comando com `dotnet run`.

```
"commandLineArgs": "--pathbase=<virtual-path>"
```

- No Visual Studio, especifique o argumento em **Propriedades > Depurar > Argumentos de aplicativo**. A configuração do argumento na página de propriedades do Visual Studio adiciona o argumento ao arquivo *launchSettings.json*.

```
--pathbase=<virtual-path>
```

## URLs

O argumento `--urls` indica os endereços IP ou os endereços de host com portas e protocolos para escutar solicitações.

- Passe o argumento ao executar o aplicativo localmente em um prompt de comando. No diretório do aplicativo, execute:

```
dotnet run --urls=http://127.0.0.1:0
```

- Adicione uma entrada ao arquivo *launchSettings.json* do aplicativo no perfil do **IIS Express**. Essa configuração é obtida ao executar o aplicativo com o Depurador do Visual Studio e ao executar o aplicativo em um prompt de comando com `dotnet run`.

```
"commandLineArgs": "--urls=http://127.0.0.1:0"
```

- No Visual Studio, especifique o argumento em **Propriedades > Depurar > Argumentos de aplicativo**. A configuração do argumento na página de propriedades do Visual Studio adiciona o argumento ao arquivo `launchSettings.json`.

```
--urls=http://127.0.0.1:0
```

## Implantar um aplicativo do Blazor do lado do cliente

Com o [modelo de hospedagem do lado do cliente](#):

- O aplicativo do Blazor, suas dependências e o tempo de execução do .NET são baixados no navegador.
- O aplicativo é executado diretamente no thread da interface do usuário do navegador. Há suporte para todas as estratégias a seguir:
  - O aplicativo do Blazor é atendido por um aplicativo ASP.NET Core. Abordado na seção [Implantação hospedada do Blazor do lado do cliente com o ASP.NET Core](#).
  - O aplicativo do Blazor é colocado em um serviço ou um servidor Web de hospedagem estático, em que o .NET não é usado para atender ao aplicativo do Blazor. Abordado na seção [Implantação autônoma do Blazor do lado do cliente](#).

### Configurar o vinculador

O Blazor executa a vinculação de IL (linguagem intermediária) em cada build para remover a IL desnecessária dos assemblies de saída. Você pode controlar a vinculação de assembly no build. Para obter mais informações, consulte [Configurar o Vinculador para o Blazor](#).

### Reescrever as URLs para obter o roteamento correto

O roteamento de solicitações para componentes de página em um aplicativo do lado do cliente não é tão simples quanto o roteamento de solicitações para um aplicativo hospedado do lado do servidor. Considere um aplicativo do lado do cliente com duas páginas:

- **Main.cshtml** – É carregado na raiz do aplicativo e contém um link para a página Sobre (`href="About"`).
- **About.cshtml** – Página Sobre.

Quando o documento padrão do aplicativo é solicitado usando a barra de endereços do navegador (por exemplo, `https://www.contoso.com/`):

1. O navegador faz uma solicitação.
2. A página padrão é retornada, que é geralmente `index.html`.
3. A `index.html` inicia o aplicativo.
4. O roteador do Blazor é carregado e a página Principal do Razor (`Main.cshtml`) é exibida.

Na página Principal, é possível carregar a página Sobre selecionando o link para ela. A seleção do link para a página Sobre funciona no cliente porque o roteador do Blazor impede que o navegador faça uma solicitação na Internet para `www.contoso.com` de `About` e atende à própria página Sobre. Todas as solicitações de páginas internas *no aplicativo do lado do cliente* funcionam da mesma maneira: Não são disparadas solicitações baseadas em navegador para os recursos hospedados no servidor na Internet. O roteador trata das solicitações internamente.

Se uma solicitação for feita usando a barra de endereços do navegador para `www.contoso.com/About`, a solicitação falhará. Esse recurso não existe no host do aplicativo na Internet, portanto, uma resposta *404 Não Encontrado* é retornada.

Como os navegadores fazem solicitações aos hosts baseados na Internet de páginas do lado do cliente, os servidores Web e os serviços de hospedagem precisam reescrever todas as solicitações de recursos que não estão fisicamente no servidor para a página `index.html`. Quando a `index.html` for retornada, o roteador do lado do cliente do aplicativo assumirá o controle e responderá com o recurso correto.

## Caminho base do aplicativo

O caminho base do aplicativo é o caminho raiz do aplicativo virtual no servidor. Por exemplo, um aplicativo que reside no servidor Contoso em uma pasta virtual em `/CoolApp/` é acessado em `https://www.contoso.com/CoolApp` e tem o caminho base virtual `/CoolApp/`. Quando o caminho base do aplicativo é definido como `CoolApp/`, o aplicativo fica ciente de onde ele reside virtualmente no servidor. O aplicativo pode usar o caminho base para construir URLs relativas à raiz do aplicativo de um componente que não esteja no diretório raiz. Isso permite que os componentes que existem em diferentes níveis da estrutura de diretório criem links para outros recursos em locais em todo o aplicativo. O caminho base do aplicativo também é usado para interceptar cliques em hiperlink em que o destino `href` do link está dentro do espaço do URI do caminho base do aplicativo. O roteador do Blazor manipula a navegação interna.

Em muitos cenários de hospedagem, o caminho virtual do servidor para o aplicativo é a raiz do aplicativo. Nesses casos, o caminho base do aplicativo é uma barra invertida (`<base href="/" />`), que é a configuração padrão de um aplicativo. Em outros cenários de hospedagem, como Páginas do GitHub e diretórios virtuais ou subaplicativos do IIS, o caminho base do aplicativo precisa ser definido como o caminho virtual do servidor para o aplicativo. Para definir o caminho base do aplicativo, adicione ou atualize a tag `<base>` na `index.html` encontrada nos elementos da tag `<head>`. Defina o valor de atributo `href` como `<virtual-path>/` (a barra à direita é necessária), em que `<virtual-path>/` é o caminho raiz do aplicativo virtual completo no servidor do aplicativo. No exemplo anterior, o caminho virtual é definido como `CoolApp/`; `<base href="CoolApp/" />`.

No caso de um aplicativo com um caminho virtual não raiz configurado (por exemplo, `<base href="CoolApp/" />`), o aplicativo não consegue localizar seus recursos *quando é executado localmente*. Para superar esse problema durante o desenvolvimento e os testes locais, você pode fornecer um argumento *base de caminho* que corresponde ao valor de `href` da tag `<base>` no tempo de execução.

Para passar o argumento base de caminho com o caminho raiz (`/`) ao executar o aplicativo localmente, execute o seguinte comando no diretório do aplicativo:

```
dotnet run --pathbase=/CoolApp
```

O aplicativo responde localmente em `http://localhost:port/CoolApp`.

Para obter mais informações, confira a seção de [valor de configuração do host base de caminho](#).

## IMPORTANT

Se um aplicativo usa o [modelo de hospedagem do lado do cliente](#) (com base no modelo de projeto **Blazor**) e é hospedado como um subaplicativo do IIS em um aplicativo do ASP.NET Core, é importante desabilitar o manipulador de módulo do ASP.NET Core herdado ou verificar se a seção `<handlers>` do aplicativo raiz (pai) no arquivo `web.config` não é herdada pelo subaplicativo.

Remova o manipulador do arquivo `web.config` publicado do aplicativo adicionando uma seção `<handlers>` ao arquivo:

```
<handlers>
  <remove name="aspNetCore" />
</handlers>
```

Como alternativa, desabilite a herança da seção `<system.webServer>` do aplicativo raiz (pai) usando um elemento

```
<location> com <inheritInChildApplications> definido como <false> :
```

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <location path=". " inheritInChildApplications="false">
    <system.webServer>
      <handlers>
        <add name="aspNetCore" ... />
      </handlers>
      <aspNetCore ... />
    </system.webServer>
  </location>
</configuration>
```

A ação de remover o manipulador ou desabilitar a herança é realizada além da ação de configurar o caminho base do aplicativo, conforme descrito nesta seção. Defina o caminho base do aplicativo no arquivo `index.html` do aplicativo do alias do IIS usado ao configurar o subaplicativo no IIS.

## Implantação hospedada do Blazor do lado do cliente com o ASP.NET Core

Uma *implantação hospedada* atende ao aplicativo do Blazor do lado do cliente para navegadores de um [aplicativo do ASP.NET Core](#) que é executado em um servidor.

O aplicativo do Blazor é incluído com o aplicativo do ASP.NET Core na saída publicada para que ambos sejam implantados juntos. É necessário um servidor Web capaz de hospedar um aplicativo do ASP.NET Core. Para uma implantação hospedada, o Visual Studio inclui o modelo de projeto **Blazor (hospedado no ASP.NET Core)** (modelo `blazorhosted` ao usar o comando `dotnet new`).

Para obter mais informações sobre a implantação e a hospedagem de aplicativo do ASP.NET Core, confira [Hospedar e implantar o ASP.NET Core](#).

Para obter informações de como implantar o Serviço de Aplicativo do Azure, confira os tópicos a seguir:

[Publicar um aplicativo ASP.NET Core no Azure com o Visual Studio](#)

Aprenda como publicar um aplicativo ASP.NET Core no Serviço de Aplicativo do Azure usando o Visual Studio.

## Implantação autônoma do Blazor do lado do cliente

Uma *implantação autônoma* atende ao aplicativo do Blazor do lado do cliente como um conjunto de arquivos estáticos que são solicitados diretamente pelos clientes. Não é usado é um servidor Web para atender ao aplicativo do Blazor.

### Hospedagem autônoma do Blazor do lado do cliente com o IIS

O IIS é um servidor de arquivos estático com capacidade para aplicativos do Blazor. Para configurar o IIS para hospedar o Blazor, confira [Build a Static Website on IIS](#) (Criar um site estático no IIS).

Os ativos publicados são criados na pasta `\bin\Release\<target-framework>\publish`. Hospede o conteúdo da

pasta *publish* no servidor Web ou no serviço de hospedagem.

## web.config

Quando um projeto Blazor é publicado, um arquivo *web.config* é criado com a seguinte configuração do IIS:

- Os tipos MIME são definidos para as seguintes extensões de arquivo:
  - \*.dll: application/octet-stream
  - \*.json: application/json
  - \*.wasm: application/wasm
  - \*.woff: application/font-woff
  - \*.woff2: application/font-woff2
- A compactação HTTP está habilitada para os seguintes tipos MIME:
  - application/octet-stream
  - application/wasm
- As regras do Módulo de Reescrita de URL são estabelecidas:
  - Atender ao subdiretório em que residem os ativos estáticos do aplicativo (*<assembly\_name>\dist\<path\_requested>*).
  - Criar o roteamento de fallback do SPA, de modo que as solicitações de ativos que não sejam arquivos sejam redirecionadas ao documento padrão do aplicativo na pasta de ativos estáticos dele (*<assembly\_name>\dist\index.html*).

## Instalar o Módulo de Reescrita de URL

O [Módulo de Reescrita de URL](#) é necessário para reescrever URLs. O módulo não está instalado por padrão e não está disponível para instalação como um recurso do serviço de função do servidor Web (IIS). O módulo precisa ser baixado do site do IIS. Use o Web Platform Installer para instalar o módulo:

1. Localmente, navegue até a [página de downloads do Módulo de Reescrita de URL](#). Para obter a versão em inglês, selecione **WebPI** para baixar o instalador do WebPI. Para outros idiomas, selecione a arquitetura adequada para o servidor (x86/x64) para baixar o instalador.
2. Copie o instalador para o servidor. Execute o instalador. Selecione o botão **Instalar** e aceite os termos de licença. Uma reinicialização do servidor não será necessária após a conclusão da instalação.

## Configurar o site

Defina o **Caminho físico** do site como a pasta do aplicativo. A pasta contém:

- O arquivo *web.config* que o IIS usa para configurar o site, incluindo as regras de redirecionamento e os tipos de conteúdo do arquivo necessários.
- A pasta de ativos estática do aplicativo.

## Solução de problemas

Se um *500 Erro Interno do Servidor* for recebido e o Gerenciador do IIS gerar erros ao tentar acessar a configuração do site, confirme se o Módulo de Reescrita de URL está instalado. Quando o módulo não estiver instalado, o arquivo *web.config* não poderá ser analisado pelo IIS. Isso impede que o Gerenciador do IIS carregue a configuração do site e que o site atenda aos arquivos estáticos do Blazor.

Para obter mais informações de como solucionar problemas de implantações no IIS, confira [Solucionar problemas do ASP.NET Core no IIS](#).

## Hospedagem autônoma do Blazor do lado do cliente com o Nginx

O arquivo *nginx.conf* a seguir é simplificado para mostrar como configurar o Nginx para enviar o arquivo *Index.html* sempre que ele não puder encontrar um arquivo correspondente no disco.

```
events { }
http {
    server {
        listen 80;

        location / {
            root /usr/share/nginx/html;
            try_files $uri $uri/ /Index.html =404;
        }
    }
}
```

Para obter mais informações sobre a configuração do servidor Web Nginx de produção, confira [Creating NGINX Plus and NGINX Configuration Files](#) (Criando arquivos de configuração do NGINX Plus e do NGINX).

#### Hospedagem autônoma do Blazor do lado do cliente no Docker

Para hospedar o Blazor no Docker usando o Nginx, configure o Dockerfile para usar a imagem do Nginx baseada no Alpine. Atualize o Dockerfile para copiar o arquivo `nginx.config` no contêiner.

Adicione uma linha ao Dockerfile, conforme é mostrado no exemplo a seguir:

```
FROM nginx:alpine
COPY ./bin/Release/netstandard2.0/publish /usr/share/nginx/html/
COPY nginx.conf /etc/nginx/nginx.conf
```

#### Hospedagem autônoma do Blazor do lado do cliente com as Páginas do GitHub

Para lidar com as reescritas de URL, adicione um arquivo `404.html` com um script que manipule o redirecionamento de solicitação para a página `index.html`. Para obter uma implementação de exemplo fornecida pela comunidade, confira [Single Page Apps for GitHub Pages](#) (Aplicativos de página única das Páginas do GitHub) ([raflex/spa-github-pages no GitHub](#)). Um exemplo usando a abordagem da comunidade pode ser visto em [blazor-demo/blazor-demo.github.io no GitHub](#) (site dinâmico).

Ao usar um site de projeto em vez de um site de empresa, adicione ou atualize a tag `<base>` no `index.html`. Defina o valor do atributo `<repository-name>/` do `href`, em que `<repository-name>/` é o nome do repositório do GitHub.

## Implantar um aplicativo dos Componentes Razor do lado do servidor

Com o [modelo de hospedagem do lado do servidor](#), os Componentes Razor são executados no servidor de dentro de um aplicativo ASP.NET Core. As atualizações da interface do usuário, a manipulação de eventos e as chamadas de JavaScript são realizadas por uma conexão SignalR.

O aplicativo é incluído com o aplicativo do ASP.NET Core na saída publicada para que os dois aplicativos sejam implantados juntos. É necessário um servidor Web capaz de hospedar um aplicativo do ASP.NET Core. Para uma implantação do lado do servidor, o Visual Studio inclui o modelo de projeto **Blazor (do lado do servidor no ASP.NET Core)** (modelo `blazorserver` ao usar o comando `dotnet new`).

Quando o aplicativo do ASP.NET Core é publicado, o aplicativo dos Componentes Razor é incluído na saída publicada para que ambos possam ser implantados juntos. Para obter mais informações sobre a implantação e a hospedagem de aplicativo do ASP.NET Core, confira [Hospedar e implantar o ASP.NET Core](#).

Para obter informações de como implantar o Serviço de Aplicativo do Azure, confira os tópicos a seguir:

#### Publicar um aplicativo ASP.NET Core no Azure com o Visual Studio

Aprenda como publicar um aplicativo ASP.NET Core no Serviço de Aplicativo do Azure usando o Visual Studio.

# Configurar o Vinculador para o Blazor

06/02/2019 • 2 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

## NOTE

O Razor Components do ASP.NET Core é compatível com ASP.NET Core 3.0 ou posterior.

Blazor é uma estrutura da Web sem suporte e experimental que não deve ser usada para cargas de trabalho de produção no momento.

O Blazor executa a vinculação de [IL \(linguagem intermediária\)](#) durante cada build do Modo de Versão para remover IL desnecessária dos assemblies de saída.

Você pode controlar a vinculação do assembly com uma das seguintes abordagens:

- Desabilite a vinculação globalmente com uma propriedade MSBuild.
- Controle a vinculação por assembly usando um arquivo de configuração.

## Desabilitar a vinculação com uma propriedade MSBuild

A vinculação é habilitada por padrão no Modo de Versão quando um aplicativo é criado, o que inclui publicação.

Para desabilitar a vinculação para todos os assemblies, defina a propriedade `<BlazorLinkOnBuild>` MSBuild como `false` no arquivo de projeto:

```
<PropertyGroup>
  <BlazorLinkOnBuild>false</BlazorLinkOnBuild>
</PropertyGroup>
```

## Controlar a vinculação com um arquivo de configuração

A vinculação pode ser controlada por assembly fornecendo um arquivo de configuração XML e especificando o arquivo como um item MSBuild no arquivo de projeto.

Veja a seguir um exemplo de arquivo de configuração (*Linker.xml*):

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
  This file specifies which parts of the BCL or Blazor packages must not be
  stripped by the IL Linker even if they aren't referenced by user code.
-->
<linker>
  <assembly fullname="mscorlib">
    <!--
      Preserve the methods in WasmRuntime because its methods are called by
      JavaScript client-side code to implement timers.
      Fixes: https://github.com/aspnet/Blazor/issues/239
    -->
    <type fullname="System.Threading.WasmRuntime" />
  </assembly>
  <assembly fullname="System.Core">
    <!--
      System.Linq.Expressions* is required by Json.NET and any
      expression.Compile caller. The assembly isn't stripped.
    -->
    <type fullname="System.Linq.Expressions*" />
  </assembly>
  <!--
    In this example, the app's entry point assembly is listed. The assembly
    isn't stripped by the IL Linker.
  -->
  <assembly fullname="MyCoolBlazorApp" />
</linker>
```

Para saber mais sobre o formato de arquivo para o arquivo de configuração, veja [Vinculador de IL: sintaxe do descriptor de xml](#).

Especifique o arquivo de configuração no arquivo de projeto com o item `BlazorLinkerDescriptor`:

```
<ItemGroup>
  <BlazorLinkerDescriptor Include="Linker.xml" />
</ItemGroup>
```

# Visão geral sobre a segurança do ASP.NET Core

09/01/2019 • 3 minutes to read • [Edit Online](#)

O ASP.NET Core permite que desenvolvedores configurem e gerenciem facilmente a segurança de seus aplicativos. O ASP.NET Core contém recursos para gerenciamento de autenticação, autorização, proteção de dados, imposição de HTTPS, segredos de aplicativo, proteção contra falsificação de solicitação e gerenciamento de CORS. Esses recursos de segurança permitem que você crie aplicativos de ASP.NET Core robustos e seguros ao mesmo tempo.

## Recursos de segurança do ASP.NET Core

O ASP.NET Core fornece várias ferramentas e bibliotecas para proteger seus aplicativos, incluindo provedores de identidade internos, mas você pode usar serviços de identidade de terceiros, como Facebook, Twitter e LinkedIn. Com o ASP.NET Core, você pode gerenciar facilmente os segredos do aplicativo, que são uma maneira de armazenar e usar informações confidenciais sem a necessidade de expô-los no código.

## Autenticação versus Autorização

A autenticação é um processo em que um usuário fornece credenciais que são comparadas àquelas armazenadas em um sistema operacional, num banco de dados, no aplicativo ou no recurso. Se elas corresponderem, os usuários se autenticarão com êxito e, assim, poderão realizar ações para as quais são autorizados, durante um processo de autorização. A autorização é o processo que determina o que um usuário pode fazer.

Outra forma de pensar na autenticação é considerá-la como uma maneira de entrar em um espaço, como um servidor, um banco de dados, um aplicativo ou um recurso, ao passo que a autorização refere-se a quais ações o usuário poderá executar em que objetos dentro desse espaço (servidor, banco de dados ou aplicativo).

## Vulnerabilidades comuns no software

O ASP.NET Core e o EF contêm recursos que ajudam a proteger seus aplicativos e impedir violações de segurança. A seguinte lista de links leva à documentação com detalhe de técnicas para evitar as vulnerabilidades de segurança mais comuns em aplicativos Web:

- [Ataques de script entre sites](#)
- [Ataques de injeção de SQL](#)
- [CSRF \(solicitação intersite forjada\)](#)
- [Ataques de redirecionamento aberto](#)

Há mais vulnerabilidades sobre as quais você deve estar atento. Para obter mais informações, veja os outros artigos na seção **Segurança e identidade** do sumário.

# Introdução à identidade do ASP.NET Core

13/02/2019 • 16 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

O ASP.NET Core Identity é um sistema de associação que adiciona a funcionalidade de logon para aplicativos ASP.NET Core. Os usuários podem criar uma conta com as informações de logon armazenadas no Identity ou eles podem usar um provedor de logon externo. Provedores de logon externo com suporte incluem [Facebook](#), [Google](#), [Account da Microsoft](#) e [Twitter](#).

Identidade pode ser configurada usando um banco de dados do SQL Server para armazenar nomes de usuário, senhas e dados de perfil. Como alternativa, outro repositório persistente pode ser usado, por exemplo, o armazenamento de tabelas do Azure.

[Exibir ou baixar o código de exemplo \(como baixar\).](#)

Neste tópico, saiba como usar a identidade para registrar, faça logon e logoff de um usuário. Para obter instruções mais detalhadas sobre como criar aplicativos que usam a identidade, consulte a seção próximas etapas no final deste artigo.

## AddDefaultIdentity e AddIdentity

[AddDefaultIdentity](#) foi introduzido no ASP.NET Core 2.1. Chamar `AddDefaultIdentity` é semelhante a chamar o seguinte:

- [AddIdentity](#)
- [AddDefaultUI](#)
- [AddDefaultTokenProviders](#)

Ver [AddDefaultIdentity fonte](#) para obter mais informações.

## Criar um aplicativo Web com autenticação

Crie um projeto de aplicativo Web ASP.NET Core com contas de usuário individuais.

- [Visual Studio](#)
- [CLI do .NET Core](#)
- Selecione **Arquivo > Novo > Projeto**.
- Selecione **Aplicativo Web ASP.NET Core**. Nomeie o projeto **WebApp1** para ter o mesmo namespace que o download do projeto. Clique em **OK**.
- Selecione um ASP.NET Core **aplicativo Web**, em seguida, selecione **alterar autenticação**.
- Selecione **contas de usuário individuais** e clique em **Okey**.

Fornece o projeto gerado [ASP.NET Core Identity](#) como um [biblioteca de classes Razor](#). A biblioteca de classes Razor identidade expõe pontos de extremidade com o `Identity` área. Por exemplo:

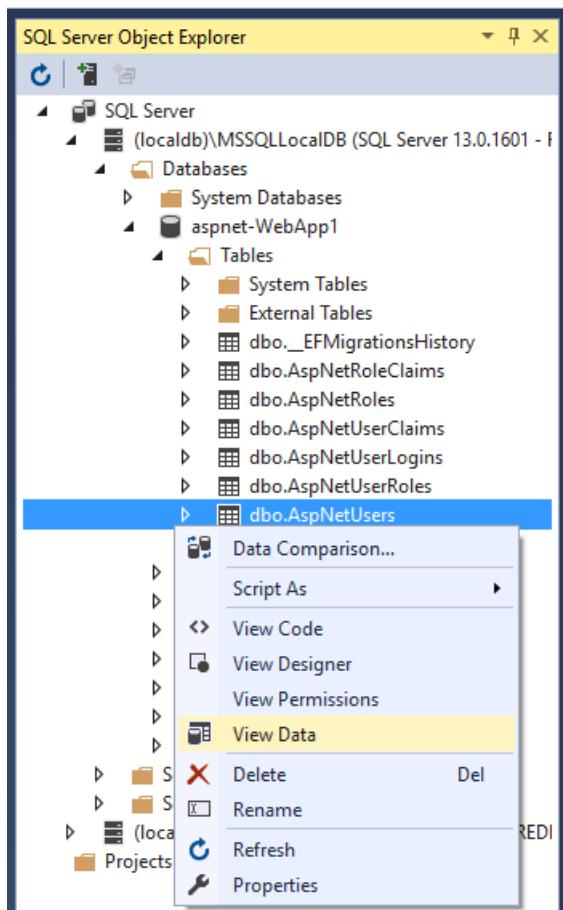
- Identidade/logon/conta
- Identidade/logoff/conta
- / Identidade/conta/gerenciar

## Registro de teste e de logon

Execute o aplicativo e registrar um usuário. Dependendo do tamanho da tela, você talvez precise selecionar o botão de alternância de navegação para ver os **registre** e **Login** links.

### Exibir o banco de dados de identidade

- [Visual Studio](#)
  - [CLI do .NET Core](#)
- Dos **modo de exibição** menu, selecione **Pesquisador de objetos do SQL Server (SSOX)**.
- Navegue até **(localdb) MSSQLLocalDB (SQL Server 13)**. Clique duas vezes em **dbo. AspNetUsers** > **exibir dados**:



### Configurar os serviços de identidade

Serviços são adicionados no `ConfigureServices`. O padrão típico consiste em chamar todos os métodos `Add{Service}` e, em seguida, chamar todos os métodos `services.Configure{Service}`.

```

public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));
    services.AddDefaultIdentity<IdentityUser>()
        .AddDefaultUI(UIFramework.Bootstrap4)
        .AddEntityFrameworkStores<ApplicationDbContext>();

    services.Configure<IdentityOptions>(options =>
    {
        // Password settings.
        options.Password.RequireDigit = true;
        options.Password.RequireLowercase = true;
        options.Password.RequireNonAlphanumeric = true;
        options.Password.RequireUppercase = true;
        options.Password.RequiredLength = 6;
        options.Password.RequiredUniqueChars = 1;

        // Lockout settings.
        options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);
        options.Lockout.MaxFailedAccessAttempts = 5;
        options.Lockout.AllowedForNewUsers = true;

        // User settings.
        options.User.AllowedUserNameCharacters =
            "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-_@+";
        options.User.RequireUniqueEmail = false;
    });

    services.ConfigureApplicationCookie(options =>
    {
        // Cookie settings
        options.Cookie.HttpOnly = true;
        options.ExpireTimeSpan = TimeSpan.FromMinutes(5);

        options.LoginPath = "/Identity/Account/Login";
        options.AccessDeniedPath = "/Identity/Account/AccessDenied";
        options SlidingExpiration = true;
    });
}

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
}

```

O código anterior configura a identidade com os valores de opção padrão. Serviços são disponibilizados para o aplicativo por meio [injeção de dependência](#).

Identidade é habilitada por meio da chamada [UseAuthentication](#). `UseAuthentication` Adiciona a autenticação [middleware](#) ao pipeline de solicitação.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseCookiePolicy();

    app.UseAuthentication();

    app.UseMvc();
}
```

```

// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity< ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores< ApplicationContext >()
        .AddDefaultTokenProviders();

    services.Configure< IdentityOptions >(options =>
    {
        // Password settings
        options.Password.RequireDigit = true;
        options.Password.RequiredLength = 8;
        options.Password.RequireNonAlphanumeric = false;
        options.Password.RequireUppercase = true;
        options.Password.RequireLowercase = false;
        options.Password.RequiredUniqueChars = 6;

        // Lockout settings
        options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(30);
        options.Lockout.MaxFailedAccessAttempts = 10;
        options.Lockout.AllowedForNewUsers = true;

        // User settings
        options.User.RequireUniqueEmail = true;
    });

    services.ConfigureApplicationCookie(options =>
    {
        // Cookie settings
        options.Cookie.HttpOnly = true;
        options.ExpireTimeSpan = TimeSpan.FromMinutes(30);
        // If the LoginPath isn't set, ASP.NET Core defaults
        // the path to /Account/Login.
        options.LoginPath = "/Account/Login";
        // If the AccessDeniedPath isn't set, ASP.NET Core defaults
        // the path to /Account/AccessDenied.
        options.AccessDeniedPath = "/Account/AccessDenied";
        options SlidingExpiration = true;
    });

    // Add application services.
    services.AddTransient< IEmailSender, EmailSender >();

    services.AddMvc();
}

```

Serviços são disponibilizados para o aplicativo por meio [injeção de dependência](#).

Identidade é habilitada para o aplicativo, chamando `UseAuthentication` no `Configure` método.

`UseAuthentication` Adiciona a autenticação [middleware](#) ao pipeline de solicitação.

```

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseBrowserLink();
        app.UseDatabaseErrorPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseStaticFiles();

    app.UseAuthentication();

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}

```

```

public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity< ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();

    services.AddMvc();

    // Configure Identity
    services.Configure<IdentityOptions>(options =>
    {
        // Password settings
        options.Password.RequireDigit = true;
        options.Password.RequiredLength = 8;
        options.Password.RequireNonAlphanumeric = false;
        options.Password.RequireUppercase = true;
        options.Password.RequireLowercase = false;

        // Lockout settings
        options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(30);
        options.Lockout.MaxFailedAccessAttempts = 10;

        // Cookie settings
        options.Cookies.ApplicationCookie.ExpireTimeSpan = TimeSpan.FromDays(150);
        options.Cookies.ApplicationCookie.LoginPath = "/Account/Login";

        // User settings
        options.User.RequireUniqueEmail = true;
    });

    // Add application services.
    services.AddTransient<IEmailSender, AuthMessageSender>();
    services.AddTransient<ISmsSender, AuthMessageSender>();
}

```

Esses serviços são disponibilizados para o aplicativo por meio [injeção de dependência](#).

Identidade é habilitada para o aplicativo, chamando `UseIdentity` no `Configure` método. `UseIdentity` adiciona a autenticação baseada em cookie [middleware](#) ao pipeline de solicitação.

```
public void Configure(IApplicationBuilder app,
    IHostingEnvironment env,
    ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole(Configuration.GetSection("Logging"));
    loggerFactory.AddDebug();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
        app.UseBrowserLink();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseStaticFiles();

    app.UseIdentity();

    // Add external authentication middleware below. To configure them please see
    https://go.microsoft.com/fwlink/?LinkID=532715

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

Para obter mais informações, consulte o [classe IdentityOptions](#) e [inicialização do aplicativo](#).

## Registre-se de Scaffold, logon e logoff

Siga o [criar o scaffolding de identidade em um projeto do Razor com autorização](#) instruções para gerar o código mostrado nesta seção.

- [Visual Studio](#)
- [CLI do .NET Core](#)

Adicione os arquivos de registro, logon e logoff.

### Registre-se de examinar

Quando um usuário clica o `register` link, o `RegisterModel.OnPostAsync` ação é invocada. O usuário é criado pelo `CreateAsync` sobre o `_userManager` objeto. `_userManager` é fornecido pela injeção de dependência):

```

public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl = returnUrl ?? Url.Content("~/");
    if (ModelState.IsValid)
    {
        var user = new IdentityUser { UserName = Input.Email, Email = Input.Email };
        var result = await _userManager.CreateAsync(user, Input.Password);
        if (result.Succeeded)
        {
            _logger.LogInformation("User created a new account with password.");

            var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
            var callbackUrl = Url.Page(
                "/Account/ConfirmEmail",
                pageHandler: null,
                values: new { userId = user.Id, code = code },
                protocol: Request.Scheme);

            await _emailSender.SendEmailAsync(Input.Email, "Confirm your email",
                $"Please confirm your account by <a href='{HtmlEncoder.Default.Encode(callbackUrl)}'>clicking here</a>.");
        }

        await _signInManager.SignInAsync(user, isPersistent: false);
        return LocalRedirect(returnUrl);
    }
    foreach (var error in result.Errors)
    {
        ModelState.AddModelError(string.Empty, error.Description);
    }
}

// If we got this far, something failed, redisplay form
return Page();
}

```

Quando um usuário clica o **registre** link, o `Register` ação é invocada em `AccountController`. O `Register` ação cria o usuário chamando `CreateAsync` sobre o `_userManager` objeto (fornecidas para `AccountController` pela injeção de dependência):

```

// 
// POST: /Account/Register
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Register(RegisterViewModel model)
{
    if (ModelState.IsValid)
    {
        var user = new ApplicationUser { UserName = model.Email, Email = model.Email };
        var result = await _userManager.CreateAsync(user, model.Password);
        if (result.Succeeded)
        {
            // For more information on how to enable account confirmation and password reset please
            // visit http://go.microsoft.com/fwlink/?LinkID=532713
            // Send an email with this link
            //var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
            //var callbackUrl = Url.Action("ConfirmEmail", "Account", new { userId = user.Id, code =
            code }, protocol: HttpContext.Request.Scheme);
            //await _emailSender.SendEmailAsync(model.Email, "Confirm your account",
            //    "Please confirm your account by clicking this link: <a href=\"" + callbackUrl +
            //    "\">link</a>");
            await _signInManager.SignInAsync(user, isPersistent: false);
            _logger.LogInformation(3, "User created a new account with password.");
            return RedirectToAction(nameof(HomeController.Index), "Home");
        }
        AddErrors(result);
    }

    // If we got this far, something failed, redisplay form
    return View(model);
}

```

Se o usuário foi criado com êxito, o usuário é conectado pela chamada para `_signInManager.SignInAsync`.

**Observação:** Ver [confirmação de conta](#) para obter as etapas impedir que o logon imediata no momento do registro.

### Fazer Logon

O formulário de logon é exibido quando:

- O **faça logon no** link é selecionado.
- Um usuário tenta acessar uma página restrita que eles não estão autorizados a acessar **ou** quando eles ainda não foram autenticados pelo sistema.

Quando o formulário na página de logon é enviado, o `OnPostAsync` ação é chamada. `PasswordSignInAsync` é chamado de `_signInManager` (fornecido pela injeção de dependência) do objeto.

```

public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl = returnUrl ?? Url.Content("~/");

    if (ModelState.IsValid)
    {
        // This doesn't count login failures towards account lockout
        // To enable password failures to trigger account lockout,
        // set lockoutOnFailure: true
        var result = await _signInManager.PasswordSignInAsync(Input.Email,
            Input.Password, Input.RememberMe, lockoutOnFailure: true);
        if (result.Succeeded)
        {
            _logger.LogInformation("User logged in.");
            return LocalRedirect(returnUrl);
        }
        if (result.RequiresTwoFactor)
        {
            return RedirectToPage("./LoginWith2fa", new { ReturnUrl = returnUrl, RememberMe =
Input.RememberMe });
        }
        if (result.IsLockedOut)
        {
            _logger.LogWarning("User account locked out.");
            return RedirectToPage("./Lockout");
        }
        else
        {
            ModelState.AddModelError(string.Empty, "Invalid login attempt.");
            return Page();
        }
    }

    // If we got this far, something failed, redisplay form
    return Page();
}

```

A base `Controller` classe expõe um `User` propriedade que você pode acessar métodos do controlador. Por exemplo, você pode enumerar `User.Claims` e tomar decisões de autorização. Para obter mais informações, consulte [Introdução à autorização no núcleo do ASP.NET](#).

O formulário de logon é exibido quando os usuários selecionam o **faça logon no** vincular ou será redirecionado ao acessar uma página que requer autenticação. Quando o usuário envia o formulário na página de login, a ação `AccountController` `Login` é chamada.

O `Login` faz chamadas `PasswordSignInAsync` no objeto `_signInManager` (fornecido pelo `AccountController` por injeção de dependência).

```

// POST: /Account/Login
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null)
{
    ViewData["ReturnUrl"] = returnUrl;
    if (ModelState.IsValid)
    {
        // This doesn't count login failures towards account lockout
        // To enable password failures to trigger account lockout, set lockoutOnFailure: true
        var result = await _signInManager.PasswordSignInAsync(model.Email,
            model.Password, model.RememberMe, lockoutOnFailure: false);
        if (result.Succeeded)
        {
            _logger.LogInformation(1, "User logged in.");
            return RedirectToLocal(returnUrl);
        }
        if (result.RequiresTwoFactor)
        {
            return RedirectToAction(nameof(SendCode), new { ReturnUrl = returnUrl, RememberMe =
model.RememberMe });
        }
        if (result.IsLockedOut)
        {
            _logger.LogWarning(2, "User account locked out.");
            return View("Lockout");
        }
        else
        {
            ModelState.AddModelError(string.Empty, "Invalid login attempt.");
            return View(model);
        }
    }

    // If we got this far, something failed, redisplay form
    return View(model);
}

```

A base de (`Controller` ou `PageModel`) classe expõe um `User` propriedade. Por exemplo, `User.Claims` podem ser enumeradas para tomar decisões de autorização.

## Fazer logoff

O **fazer logoff** link invoca o `LogoutModel.OnPost` ação.

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Logging;
using System.Threading.Tasks;

namespace WebApp1.Areas.Identity.Pages.Account
{
    [AllowAnonymous]
    public class LogoutModel : PageModel
    {
        private readonly SignInManager<IdentityUser> _signInManager;
        private readonly ILogger<LogoutModel> _logger;

        public LogoutModel(SignInManager<IdentityUser> signInManager, ILogger<LogoutModel> logger)
        {
            _signInManager = signInManager;
            _logger = logger;
        }

        public void OnGet()
        {
        }

        public async Task<IActionResult> OnPost(string returnUrl = null)
        {
            await _signInManager.SignOutAsync();
            _logger.LogInformation("User logged out.");
            if (returnUrl != null)
            {
                return LocalRedirect(returnUrl);
            }
            else
            {
                return Page();
            }
        }
    }
}

```

[SignOutAsync](#) limpa as declarações do usuário armazenadas em um cookie. Não redirecionar após chamar [SignOutAsync](#) ou o usuário irá **não** ser desconectado.

POST é especificado na *Pages/Shared/\_LoginPartial.cshtml*:

```

@using Microsoft.AspNetCore.Identity
@inject SignInManager<IdentityUser> SignInManager
@inject UserManager<IdentityUser> UserManager

<ul class="navbar-nav">
    @if (SignInManager.IsSignedIn(User))
    {
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="Identity"
               asp-page="/Account/Manage/Index"
               title="Manage">Hello@User.Identity.Name!</a>
        </li>
        <li class="nav-item">
            <form class="form-inline" asp-area="Identity" asp-page="/Account/Logout"
                  asp-route-returnUrl="@Url.Page("/", new { area = "" })"
                  method="post">
                <button type="submit" class="nav-link btn btn-link text-dark">Logout</button>
            </form>
        </li>
    }
    else
    {
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="Identity" asp-page="/Account/Register">Register</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="Identity" asp-page="/Account/Login">Login</a>
        </li>
    }
</ul>

```

Clicar a **fazer logoff** vincular chamadas a `LogOut` ação.

```

// 
// POST: /Account/LogOut
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> LogOut()
{
    await _signInManager.SignOutAsync();
    _logger.LogInformation(4, "User logged out.");
    return RedirectToAction(nameof(HomeController.Index), "Home");
}

```

O código anterior chama o `_signInManager.SignOutAsync` método. O `SignOutAsync` método limpa as declarações do usuário armazenadas em um cookie.

## Identidade de teste

Os modelos de projeto da web padrão permitem acesso anônimo para as home pages. Para testar a identidade, adicione `[Authorize]` para a página de privacidade.

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace WebApp1.Pages
{
    [Authorize]
    public class PrivacyModel : PageModel
    {
        public void OnGet()
        {
        }
    }
}
```

Se você estiver entrado, saia do serviço. Execute o aplicativo e selecione o **privacidade** link. Você será redirecionado à página de logon.

### Explore a identidade

Para explorar a identidade em mais detalhes:

- [Criar fonte de interface do usuário de identidade completa](#)
- Examine a origem de cada página e percorrer o depurador.

## Componentes de identidade

Todos os identidade NuGet pacotes dependentes são incluídos na [metapacote Microsoft](#).

O pacote principal para a identidade está [Microsoft.AspNetCore.Identity](#). Este pacote contém o conjunto principal de interfaces para ASP.NET Core Identity e é incluído por  
`Microsoft.AspNetCore.Identity.EntityFrameworkCore`.

## Migrando para o ASP.NET Core Identity

Para obter mais informações e diretrizes sobre como migrar o armazenamento de identidade existente, consulte [migrar autenticação e identidade](#).

## Definindo a força da senha

Consulte [configuração](#) para obter um exemplo que defina os requisitos mínimos de senha.

## Próximas etapas

- [Configurar o Identity](#)
- [Criar um aplicativo ASP.NET Core com os dados de usuário protegidos por autorização](#)
- [Adicionar, baixar e excluir dados de usuário à identidade em um projeto ASP.NET Core](#)
- [Habilitar a geração de código QR TOTP para aplicativos de autenticador no ASP.NET Core](#)
- [Migrar autenticação e identidade para o ASP.NET Core](#)
- [Confirmação de conta e de recuperação de senha no ASP.NET Core](#)
- [Autenticação de dois fatores com SMS no ASP.NET Core](#)
- [Hospedar o ASP.NET Core em um web farm](#)

# Identidade de Scaffold em projetos ASP.NET Core

26/10/2018 • 22 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

O ASP.NET Core 2.1 e posterior fornece [ASP.NET Core Identity](#) como um [biblioteca de classes Razor](#). Aplicativos que incluem a identidade podem aplicar o scaffoldler para seletivamente, adicione o código-fonte contido na identidade Razor classe RCL (biblioteca). Talvez você queira gerar o código-fonte para que você possa modificar o código e alterar o comportamento. Por exemplo, você pode instruir o scaffoldler a gerar o código usado no registro. Código gerado tem precedência sobre o mesmo código na RCL de Identidade. Para obter controle total da interface do usuário e usar a RCL padrão, consulte a seção [criar fonte de interface do usuário de identidade completa](#).

Aplicativos que fazem **não** incluem autenticação pode aplicar o scaffoldler para adicionar o pacote de identidade da RCL. Você tem a opção de selecionar o código de Identidade a ser gerado.

Embora o scaffoldler gera a maioria do código necessário, você precisará atualizar seu projeto para concluir o processo. Este documento explica as etapas necessárias para concluir uma atualização de scaffolding de identidade.

Quando o scaffoldler de identidade é executado, uma *Scaffoldingreadme* arquivo é criado no diretório do projeto. O *Scaffoldingreadme* arquivo contém instruções gerais sobre o que é necessário para concluir a atualização de scaffolding de identidade. Este documento contém instruções mais completas que o *Scaffoldingreadme* arquivo.

É recomendável usar um sistema de controle do código-fonte que mostra as diferenças de arquivo e permite que você faça alterações fora. Inspecione as alterações depois de executar o scaffoldler de identidade.

## NOTE

Os serviços são necessários ao usar [autenticação de dois fatores](#), [recuperação de confirmação e a senha da conta](#) e outros recursos de segurança com identidade. Serviços ou stubs de serviço não são gerados quando o scaffolding de identidade. Serviços para habilitar esses recursos devem ser adicionados manualmente. Por exemplo, consulte [exigem Email de confirmação](#).

## Identidade de Scaffold em um projeto vazio

Execute o scaffoldler de identidade:

- [Visual Studio](#)
- [CLI do .NET Core](#)
- Partir **Gerenciador de soluções**, clique com botão direito no projeto > **Add** > **New Scaffolded Item**.
- No painel à esquerda do **adicionar Scaffold** caixa de diálogo, selecione **identidade** > **adicionar**.
- No **identidade de adição** caixa de diálogo, selecione as opções desejadas.
  - Selecione a página de layout existente ou seu arquivo de layout será substituído pela marcação incorreta. Por exemplo `~/Pages/Shared/_Layout.cshtml` para as páginas Razor `~/Views/Shared/_Layout.cshtml` para projetos MVC
  - Selecione o + botão para criar um novo **classe de contexto de dados**.
- Selecione **adicionar**.

Adicione as seguintes chamadas destacadas para o `Startup` classe:

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc();
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseHsts();
        }

        app.UseHttpsRedirection();
        app.UseStaticFiles();
        app.UseAuthentication();
        app.UseMvc();
    }
}
```

`UseHsts` é recomendável, mas não é necessário. Ver [protocolo de segurança de transporte estrito HTTP](#) para obter mais informações.

Exige que o código de banco de dados de identidade gerado [migrações do Entity Framework Core](#). Criar uma migração e atualizar o banco de dados. Por exemplo, execute os seguintes comandos:

- [Visual Studio](#)
- [CLI do .NET Core](#)

No Visual Studio **Package Manager Console**:

```
Add-Migration CreateIdentitySchema
Update-Database
```

O parâmetro de nome "CreateIdentitySchema" para o `Add-Migration` comando é arbitrário.  
"CreateIdentitySchema" Descreve a migração.

## Identidade de Scaffold em um projeto do Razor sem autorização existente

Execute o scaffold de identidade:

- [Visual Studio](#)
  - [CLI do .NET Core](#)
- 
- Partir **Gerenciador de soluções**, clique com botão direito no projeto > **Add > New Scaffolded Item**.
  - No painel à esquerda do **adicionar Scaffold** caixa de diálogo, selecione **identidade** > **adicionar**.
  - No **identidade de adição** caixa de diálogo, selecione as opções desejadas.
    - Selecione a página de layout existente ou seu arquivo de layout será substituído pela marcação incorreta. Por exemplo `~/Pages/Shared/_Layout.cshtml` para as páginas Razor

`~/Views/Shared/_Layout.cshtml` para projetos MVC

- Selecione o + botão para criar um novo **classe de contexto de dados**.
- Selecione **adicionar**.

Identidade é configurada no `Areas/Identity/IdentityHostingStartup.cs`. Para obter mais informações, consulte [IHostingStartup](#).

### As migrações, UseAuthentication e layout

Exige que o código de banco de dados de identidade gerado [migrações do Entity Framework Core](#). Criar uma migração e atualizar o banco de dados. Por exemplo, execute os seguintes comandos:

- [Visual Studio](#)
- [CLI do .NET Core](#)

No Visual Studio **Package Manager Console**:

```
Add-Migration CreateIdentitySchema  
Update-Database
```

O parâmetro de nome "CreateIdentitySchema" para o `Add-Migration` comando é arbitrário.

"CreateIdentitySchema" Descreve a migração.

### Habilitar a autenticação

No `Configure` método da `Startup` classe, chame `UseAuthentication` depois `UseStaticFiles`:

```
public class Startup  
{  
    public Startup(IConfiguration configuration)  
    {  
        Configuration = configuration;  
    }  
  
    public IConfiguration Configuration { get; }  
  
    public void ConfigureServices(IServiceCollection services)  
    {  
        services.AddMvc();  
    }  
  
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)  
    {  
        if (env.IsDevelopment())  
        {  
            app.UseDeveloperExceptionPage();  
        }  
        else  
        {  
            app.UseExceptionHandler("/Error");  
            app.UseHsts();  
        }  
  
        app.UseHttpsRedirection();  
        app.UseStaticFiles();  
        app.UseAuthentication();  
  
        app.UseMvc();  
    }  
}
```

`UseHsts` é recomendável, mas não é necessário. Ver [protocolo de segurança de transporte estrito HTTP](#) para

obter mais informações.

## Alterações de layout

Opcional: Adicionar o logon parcial (`_LoginPartial`) para o arquivo de layout:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - RazorNoAuth8</title>

    <environment include="Development">
        <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
        <link rel="stylesheet" href="~/css/site.css" />
    </environment>
    <environment exclude="Development">
        <link rel="stylesheet" href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/css/bootstrap.min.css"
              asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"
              asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-test-
              value="absolute" />
        <link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true" />
    </environment>
</head>
<body>
    <nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-
collapse">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a asp-page="/Index" class="navbar-brand">RazorNoAuth8</a>
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li><a asp-page="/Index">Home</a></li>
                    <li><a asp-page="/About">About</a></li>
                    <li><a asp-page="/Contact">Contact</a></li>
                </ul>
                <partial name="_LoginPartial" />
            </div>
        </div>
    </nav>

    <partial name="_CookieConsentPartial" />

    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; 2018 - RazorNoAuth8</p>
        </footer>
    </div>

    <environment include="Development">
        <script src="~/lib/jquery/dist/jquery.js"></script>
        <script src="~/lib/bootstrap/dist/js/bootstrap.js"></script>
        <script src="~/js/site.js" asp-append-version="true"></script>
    </environment>
    <environment exclude="Development">
        <script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-3.3.1.min.js"
               asp-fallback-src="~/lib/jquery/dist/jquery.min.js"
               asp-fallback-test="window.jQuery"></script>
    </environment>
</body>
</html>
```

```

        crossorigin="anonymous"
        integrity="sha384-K+ctZQ+LL8q6tP7I94W+qzQsfRV2a+AfhII9k8z8l9ggpc8X+Ytst4yBo/hH+8Fk">
    </script>
    <script src="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/bootstrap.min.js"
        asp-fallback-src="~/lib/bootstrap/dist/js/bootstrap.min.js"
        asp-fallback-test="window.jQuery && window.jQuery.fn && window.jQuery.fn.modal"
        crossorigin="anonymous"
        integrity="sha384-sh384-Tc5IQib027qvyjSMFHjOMaLkfuvWxZxUPnCJA7l2mCWNIPG9mGCD8wGNICPD7Txa">
    </script>
    <script src("~/js/site.min.js" asp-append-version="true"></script>
</environment>

@RenderSection("Scripts", required: false)
</body>
</html>

```

## Identidade de Scaffold em um projeto do Razor com autorização

Execute o scaffolder de identidade:

- [Visual Studio](#)
- [CLI do .NET Core](#)
- Partir **Gerenciador de soluções**, clique com botão direito no projeto > **Add** > **New Scaffolded Item**.
- No painel à esquerda do **adicionar Scaffold** caixa de diálogo, selecione **identidade** > **adicionar**.
- No **identidade de adição** caixa de diálogo, selecione as opções desejadas.
  - Selecione a página de layout existente ou seu arquivo de layout será substituído pela marcação incorreta. Quando um existente `_layout.cshtml` arquivo for selecionado, ele é **não** substituídos.

Por exemplo `~/Pages/Shared/_Layout.cshtml` para as páginas Razor `~/Views/Shared/_Layout.cshtml` para projetos MVC

- Para usar o contexto de dados existente, selecione pelo menos um arquivo para substituir. Você deve selecionar pelo menos um arquivo para adicionar seu contexto de dados.
  - Selecione sua classe de contexto de dados.
  - Selecione **adicionar**.
- Para criar um novo contexto de usuário e, possivelmente, crie uma classe de usuário personalizada para a identidade:
  - Selecione o + botão para criar um novo **classe de contexto de dados**.
  - Selecione **adicionar**.

Observação: Se você estiver criando um novo contexto de usuário, você não precisa selecionar um arquivo para substituir.

Algumas opções de identidade são configuradas no `Areas/Identity/IdentityHostingStartup.cs`. Para obter mais informações, consulte [IHostingStartup](#).

## Identidade de Scaffold em um projeto do MVC sem autorização existente

Execute o scaffolder de identidade:

- [Visual Studio](#)
- [CLI do .NET Core](#)
- Partir **Gerenciador de soluções**, clique com botão direito no projeto > **Add** > **New Scaffolded Item**.
- No painel à esquerda do **adicionar Scaffold** caixa de diálogo, selecione **identidade** > **adicionar**.

- No **identidade de adição** caixa de diálogo, selecione as opções desejadas.
  - Selecione a página de layout existente ou seu arquivo de layout será substituído pela marcação incorreta. Por exemplo `~/Pages/Shared/_Layout.cshtml` para as páginas Razor `~/Views/Shared/_Layout.cshtml` para projetos MVC
  - Selecione o + botão para criar um novo **classe de contexto de dados**.
- Selecione **adicionar**.

Opcional: Adicionar o logon parcial (`_LoginPartial`) para o `Views/Shared/_Layout.cshtml` arquivo:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@ViewData["Title"] - MvcNoAuth3</title>

  <environment include="Development">
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
    <link rel="stylesheet" href="~/css/site.css" />
  </environment>
  <environment exclude="Development">
    <link rel="stylesheet" href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/css/bootstrap.min.css"
          asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"
          asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-test-
          value="absolute" />
    <link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true" />
  </environment>
</head>
<body>
  <nav class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
      <div class="navbar-header">
        <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-
collapse">
          <span class="sr-only">Toggle navigation</span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
        </button>
        <a asp-area="" asp-controller="Home" asp-action="Index" class="navbar-brand">MvcNoAuth3</a>
      </div>
      <div class="navbar-collapse collapse">
        <ul class="nav navbar-nav">
          <li><a asp-area="" asp-controller="Home" asp-action="Index">Home</a></li>
          <li><a asp-area="" asp-controller="Home" asp-action="About">About</a></li>
          <li><a asp-area="" asp-controller="Home" asp-action="Contact">Contact</a></li>
        </ul>
        <partial name="_LoginPartial" />
      </div>
    </div>
  </nav>

  <partial name="_CookieConsentPartial" />

  <div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
      <p>&copy; 2018 - MvcNoAuth3</p>
    </footer>
  </div>

  <environment include="Development">
    <script src="~/lib/jquery/dist/jquery.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.js"></script>
    <script src="~/lib/bootstrap/dist/js/npm.js"></script>
  </environment>

```

```

<script src="~/js/site.js" asp-append-version="true"></script>
</environment>
<environment exclude="Development">
    <script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-3.3.1.min.js"
        asp-fallback-src="~/lib/jquery/dist/jquery.min.js"
        asp-fallback-test="window.jQuery"
        crossorigin="anonymous"
        integrity="sha384-K+ctZQ+LL8q6tP7I94W+qzQsfRV2a+AfHIi9k8z8l9ggpc8X+Ytst4yBo/hH+8Fk">
    </script>
    <script src="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/bootstrap.min.js"
        asp-fallback-src="~/lib/bootstrap/dist/js/bootstrap.min.js"
        asp-fallback-test="window.jQuery && window.jQuery.fn && window.jQuery.fn.modal"
        crossorigin="anonymous"
        integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfUWVxZxUPnCJA7l2mCWNIPG9mGCD8wGNICPD7Txo">
    </script>
    <script src="~/js/site.min.js" asp-append-version="true"></script>
</environment>

    @RenderSection("Scripts", required: false)
</body>
</html>

```

- Mover o *Pages/Shared/\_LoginPartial.cshtml* arquivo *Views/Shared/\_LoginPartial.cshtml*

Identidade é configurada no *Areas/Identity/IdentityHostingStartup.cs*. Para obter mais informações, consulte *IHostingStartup*.

Exige que o código de banco de dados de identidade gerado [migrações do Entity Framework Core](#). Criar uma migração e atualizar o banco de dados. Por exemplo, execute os seguintes comandos:

- [Visual Studio](#)
- [CLI do .NET Core](#)

No Visual Studio **Package Manager Console**:

```

Add-Migration CreateIdentitySchema
Update-Database

```

O parâmetro de nome "CreateIdentitySchema" para o `Add-Migration` comando é arbitrário.  
`"CreateIdentitySchema"` Descreve a migração.

Chame `UseAuthentication` depois `UseStaticFiles` :

```

public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc();
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Home/Error");
            app.UseHsts();
        }

        app.UseHttpsRedirection();
        app.UseStaticFiles();
        app.UseAuthentication();
        app.UseMvcWithDefaultRoute();
    }
}

```

`UseHsts` é recomendável, mas não é necessário. Ver [protocolo de segurança de transporte estrito HTTP](#) para obter mais informações.

## Identidade de Scaffold em um projeto do MVC com autorização

Execute o scaffolder de identidade:

- [Visual Studio](#)
- [CLI do .NET Core](#)
  
- Partir **Gerenciador de soluções**, clique com botão direito no projeto > **Add > New Scaffolded Item**.
- No painel à esquerda do **adicionar Scaffold** caixa de diálogo, selecione **identidade** > **adicionar**.
- No **identidade de adição** caixa de diálogo, selecione as opções desejadas.
  - Selecione a página de layout existente ou seu arquivo de layout será substituído pela marcação incorreta. Quando um existente `_layout.cshtml` arquivo for selecionado, ele é **não** substituídos.

Por exemplo `~/Pages/Shared/_Layout.cshtml` para as páginas Razor `~/Views/Shared/_Layout.cshtml` para projetos MVC

- Para usar o contexto de dados existente, selecione pelo menos um arquivo para substituir. Você deve selecionar pelo menos um arquivo para adicionar seu contexto de dados.
  - Selecione sua classe de contexto de dados.
  - Selecione **adicionar**.
- Para criar um novo contexto de usuário e, possivelmente, crie uma classe de usuário personalizada para a identidade:
  - Selecione o + botão para criar um novo **classe de contexto de dados**.
  - Selecione **adicionar**.

Observação: Se você estiver criando um novo contexto de usuário, você não precisa selecionar um arquivo para substituir.

Excluir o `páginas/Shared` pasta e os arquivos nessa pasta.

## Criar fonte de interface do usuário de identidade completa

Para manter o controle total da interface do usuário de identidade, execute o scaffolder de identidade e selecione **substituir todos os arquivos**.

O seguinte código realçado mostra as alterações para substituir o padrão de identidade da interface do usuário com identidade em um aplicativo web do ASP.NET Core 2.1. Você talvez queira fazer isso para ter controle total sobre a interface do usuário de identidade.

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<IdentityUser, IdentityRole>()
        // services.AddDefaultIdentity<IdentityUser>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1)
        .AddRazorPagesOptions(options =>
    {
        options.AllowAreas = true;
        options.Conventions.AuthorizeAreaFolder("Identity", "/Account/Manage");
        options.Conventions.AuthorizeAreaPage("Identity", "/Account/Logout");
    });

    services.ConfigureApplicationCookie(options =>
    {
        options.LoginPath = $"/Identity/Account/Login";
        options.LogoutPath = $"/Identity/Account/Logout";
        options.AccessDeniedPath = $"/Identity/Account/AccessDenied";
    });

    // using Microsoft.AspNetCore.Identity.UI.Services;
    services.AddSingleton<IEmailSender, EmailSender>();
}
```

O padrão de identidade é substituído no código a seguir:

```
services.AddIdentity<IdentityUser, IdentityRole>()
    // services.AddDefaultIdentity<IdentityUser>()
    .AddEntityFrameworkStores<ApplicationDbContext>()
    .AddDefaultTokenProviders();
```

O seguinte código define a [LoginPath](#), [LogoutPath](#), e [AccessDeniedPath](#):

```
services.ConfigureApplicationCookie(options =>
{
    options.LoginPath = $"/Identity/Account/Login";
    options.LogoutPath = $"/Identity/Account/Logout";
    options.AccessDeniedPath = $"/Identity/Account/AccessDenied";
});
```

Registrar um `IEmailSender` implementação, por exemplo:

```
// using Microsoft.AspNetCore.Identity.UI.Services;
services.AddSingleton<IEmailSender, EmailSender>();
```

```
public class EmailSender : IEmailSender
{
    public Task SendEmailAsync(string email, string subject, string message)
    {
        return Task.CompletedTask;
    }
}
```

## Recursos adicionais

- [Alterações no código de autenticação para o ASP.NET Core 2.1 e posterior](#)

# Adicionar, baixar e excluir dados de usuário personalizada à identidade em um projeto ASP.NET Core

09/12/2018 • 10 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

Este artigo mostra como:

- Adicione dados de usuário personalizada para um aplicativo web ASP.NET Core.
- Decore o modelo de dados de usuário personalizada com o [PersonalData](#) para que ele fica automaticamente disponível para download e exclusão de atributo. Tornando os dados capazes de ser baixado e excluído ajuda a cumprir [GDPR](#) requisitos.

O exemplo de projeto é criado a partir de um aplicativo web páginas Razor, mas as instruções são semelhantes para um aplicativo web ASP.NET Core MVC.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Pré-requisitos

[SDK do .NET Core 2.1 ou posteriores](#)

## Criar um aplicativo Web do Razor

- [Visual Studio](#)
- [CLI do .NET Core](#)
- No menu **Arquivo** do Visual Studio, selecione **Novo > Projeto**. Nomeie o projeto **WebApp1** se você deseja corresponder ao namespace da [baixar exemplo](#) código.
- Selecione **aplicativo Web ASP.NET Core > Okey**
- Selecione **ASP.NET Core 2.1** na lista suspensa
- Selecione **aplicativo Web > Okey**
- Compile e execute o projeto.

## Execute o scaffolder de identidade

- [Visual Studio](#)
- [CLI do .NET Core](#)
- Partir **Gerenciador de soluções**, clique com botão direito no projeto > **Add > New Scaffolded Item**.
- No painel à esquerda do **adicionar Scaffold** caixa de diálogo, selecione **identidade > adicionar**.
- No **identidade de adição** caixa de diálogo, as seguintes opções:
  - Selecione o arquivo de layout existente `~/Pages/Shared/_Layout.cshtml`
  - Selecione os arquivos a seguir para substituir:
    - **Conta/registo**
    - **Conta/gerenciar/índice**
  - Selecione o **+** botão para criar um novo **classe de contexto de dados**. Aceite o tipo

(**WebApp1.Models.WebApp1Context** se o projeto é denominado **WebApp1**).

- Selecione o + botão para criar um novo **classe User**. Aceite o tipo (**WebApp1User** se o projeto é denominado **WebApp1**) > **adicionar**.
- Selecione **adicionar**.

Siga as instruções da [migrações](#), [UseAuthentication](#) e [layout](#) para executar as seguintes etapas:

- Criar uma migração e atualizar o banco de dados.
- Adicione `UseAuthentication` a `Startup.Configure`.
- Adicionar `<partial name="_LoginPartial" />` ao arquivo de layout.
- Teste o aplicativo:
  - Registrar um usuário
  - Selecione o novo nome de usuário (ao lado de **Logout** link). Talvez você precise expandir a janela ou selecione o ícone da barra de navegação para mostrar o nome de usuário e outros links.
  - Selecione o **dados pessoais** guia.
  - Selecione o **Baixe** botão e examinando o *PersonalData.json* arquivo.
  - Teste o **excluir** botão, que exclui o fez logon de usuário.

## Adicionar dados de usuário personalizada para o banco de dados de identidade

Atualização de `IdentityUser` derivado da classe com propriedades personalizadas. Se você nomeou o projeto WebApp1, o arquivo é nomeado *Areas/Identity/Data/WebApp1User.cs*. Atualize o arquivo com o código a seguir:

```
using Microsoft.AspNetCore.Identity;
using System;

namespace WebApp1.Areas.Identity.Data
{
    public class WebApp1User : IdentityUser
    {
        [PersonalData]
        public string Name { get; set; }
        [PersonalData]
        public DateTime DOB { get; set; }
    }
}
```

As propriedades decoradas com o [PersonalData](#) atributo são:

- Excluído, quando o *Areas/Identity/Pages/Account/Manage/DeletePersonalData.cshtml* página do Razor chama `UserManager.Delete`.
- Incluído nos dados baixados pela *Areas/Identity/Pages/Account/Manage/DownloadPersonalData.cshtml* página do Razor.

### Atualizar a página Account/Manage/Index.cshtml

Atualizar o `InputModel` na *Areas/Identity/Pages/Account/Manage/Index.cshtml.cs* com o seguinte código realçado:

```
public partial class IndexModel : PageModel
{
    private readonly UserManager<WebApp1User> _userManager;
    private readonly SignInManager<WebApp1User> _signInManager;
    private readonly IEmailSender _emailSender;

    public IndexModel(
```

```

        UserManager<WebApp1User> userManager,
        SignInManager<WebApp1User> signInManager,
        IEmailSender emailSender)
    {
        _userManager = userManager;
        _signInManager = signInManager;
        _emailSender = emailSender;
    }

    public string Username { get; set; }

    public bool IsEmailConfirmed { get; set; }

    [TempData]
    public string StatusMessage { get; set; }
    [BindProperty]
    public InputModel Input { get; set; }

    public class InputModel
    {
        [Required]
        [DataType(DataType.Text)]
        [Display(Name = "Full name")]
        public string Name { get; set; }

        [Required]
        [Display(Name = "Birth Date")]
        [DataType(DataType.Date)]
        public DateTime DOB { get; set; }

        [Required]
        [EmailAddress]
        public string Email { get; set; }

        [Phone]
        [Display(Name = "Phone number")]
        public string PhoneNumber { get; set; }
    }

    public async Task<IActionResult> OnGetAsync()
    {
        var user = await _userManager.GetUserAsync(User);
        if (user == null)
        {
            return NotFound($"Unable to load user with ID '{_userManager.GetUserId(User)}'.");
        }

        var userName = await _userManager.GetUserNameAsync(user);
        var email = await _userManager.GetEmailAsync(user);
        var phoneNumber = await _userManager.GetPhoneNumberAsync(user);

        Username = userName;

        Input = new InputModel
        {
            Name = user.Name,
            DOB = user.DOB,
            Email = email,
            PhoneNumber = phoneNumber
        };
    }

    IsEmailConfirmed = await _userManager.IsEmailConfirmedAsync(user);

    return Page();
}

public async Task<IActionResult> OnPostAsync()
{
    if (!ModelState.IsValid)

```

```

    {
        return Page();
    }

    var user = await _userManager.GetUserAsync(User);
    if (user == null)
    {
        return NotFound($"Unable to load user with ID '{_userManager.GetUserId(User)}'.");
    }

    if (Input.Name != user.Name)
    {
        user.Name = Input.Name;
    }

    if (Input.DOB != user.DOB)
    {
        user.DOB = Input.DOB;
    }

    var email = await _userManager.GetEmailAsync(user);
    if (Input.Email != email)
    {
        var setEmailResult = await _userManager.SetEmailAsync(user, Input.Email);
        if (!setEmailResult.Succeeded)
        {
            var userId = await _userManager.GetUserIdAsync(user);
            throw new InvalidOperationException($"Unexpected error occurred setting email for user with ID '{userId}'.");
        }
    }

    var phoneNumber = await _userManager.GetPhoneNumberAsync(user);
    if (Input.PhoneNumber != phoneNumber)
    {
        var setPhoneResult = await _userManager.SetPhoneNumberAsync(user, Input.PhoneNumber);
        if (!setPhoneResult.Succeeded)
        {
            var userId = await _userManager.GetUserIdAsync(user);
            throw new InvalidOperationException($"Unexpected error occurred setting phone number for user with ID '{userId}'.");
        }
    }

    await _userManager.UpdateAsync(user);

    await _signInManager.RefreshSignInAsync(user);
    StatusMessage = "Your profile has been updated";
    return RedirectToPage();
}

public async Task<IActionResult> OnPostSendVerificationEmailAsync()
{
    if (!ModelState.IsValid)
    {
        return Page();
    }

    var user = await _userManager.GetUserAsync(User);
    if (user == null)
    {
        return NotFound($"Unable to load user with ID '{_userManager.GetUserId(User)}'.");
    }

    var userId = await _userManager.GetUserIdAsync(user);
    var email = await _userManager.GetEmailAsync(user);
    var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
}

```

```
        var callbackUrl = Url.Page(
            "/Account/ConfirmEmail",
            pageHandler: null,
            values: new { userId = userId, code = code },
            protocol: Request.Scheme);
        await _emailSender.SendEmailAsync(
            email,
            "Confirm your email",
            $"Please confirm your account by <a href='{HtmlEncoder.Default.Encode(callbackUrl)}'>clicking here</a>.");
    }

    StatusMessage = "Verification email sent. Please check your email.";
    return RedirectToPage();
}
}
```

Atualizar o *Areas/Identity/Pages/Account/Manage/Index.cshtml* com a seguinte marcação realçada:

```

@page
@model IndexModel
 @{
     ViewData["Title"] = "Profile";
 }

<h4>@ViewData["Title"]</h4>
@Html.Partial("_StatusMessage", Model.StatusMessage)
<div class="row">
    <div class="col-md-6">
        <form id="profile-form" method="post">
            <div asp-validation-summary="All" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Username"></label>
                <input asp-for="Username" class="form-control" disabled />
            </div>
            <div class="form-group">
                <label asp-for="Input.Email"></label>
                @if (Model.IsEmailConfirmed)
                {
                    <div class="input-group">
                        <input asp-for="Input.Email" class="form-control" />
                        <span class="input-group-addon" aria-hidden="true"><span class="glyphicon glyphicon-ok text-success"></span></span>
                    </div>
                }
                else
                {
                    <input asp-for="Input.Email" class="form-control" />
                    <button id="email-verification" type="submit" asp-page-handler="SendVerificationEmail" class="btn btn-link">Send verification email</button>
                }
                <span asp-validation-for="Input.Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <div class="form-group">
                    <label asp-for="Input.Name"></label>
                    <input asp-for="Input.Name" class="form-control" />
                </div>
                <div class="form-group">
                    <label asp-for="Input.DOB"></label>
                    <input asp-for="Input.DOB" class="form-control" />
                </div>
                <label asp-for="Input.PhoneNumber"></label>
                <input asp-for="Input.PhoneNumber" class="form-control" />
                <span asp-validation-for="Input.PhoneNumber" class="text-danger"></span>
            </div>
            <button type="submit" class="btn btn-default">Save</button>
        </form>
    </div>
</div>

@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}

```

## Atualizar a página Register

Atualizar o `InputModel` na `Areas/Identity/Pages/Account/Register.cshtml.cs` com o seguinte código realçado:

```

[BindProperty]
public InputModel Input { get; set; }

public string ReturnUrl { get; set; }

public class InputModel
{

```

```

{
    [Required]
    [DataType(DataType.Text)]
    [Display(Name = "Full name")]
    public string Name { get; set; }

    [Required]
    [Display(Name = "Birth Date")]
    [DataType(DataType.Date)]
    public DateTime DOB { get; set; }

    [Required]
    [EmailAddress]
    [Display(Name = "Email")]
    public string Email { get; set; }

    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at max {1} characters long.",
    MinimumLength = 6)]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    public string Password { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Confirm password")]
    [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
    public string ConfirmPassword { get; set; }
}

public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl = returnUrl ?? Url.Content("~/");
    if (ModelState.IsValid)
    {
        var user = new WebApp1User {
            UserName = Input.Email,
            Email = Input.Email,
            Name = Input.Name,
            DOB = Input.DOB
        };
        var result = await _userManager.CreateAsync(user, Input.Password);
        if (result.Succeeded)
        {
            _logger.LogInformation("User created a new account with password.");
            var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
            var callbackUrl = Url.Page(
                "/Account/ConfirmEmail",
                pageHandler: null,
                values: new { userId = user.Id, code = code },
                protocol: Request.Scheme);

            await _emailSender.SendEmailAsync(Input.Email, "Confirm your email",
                $"Please confirm your account by <a href='{HtmlEncoder.Default.Encode(callbackUrl)}'>clicking here</a>.");
            await _signInManager.SignInAsync(user, isPersistent: false);
            return LocalRedirect(returnUrl);
        }
        foreach (var error in result.Errors)
        {
            ModelState.AddModelError(string.Empty, error.Description);
        }
    }
    // If we got this far, something failed, redisplay form
    return Page();
}

```

Atualizar o `Areas/Identity/Pages/Account/Register.cshtml` com a seguinte marcação realçada:

```
@page
@model RegisterModel
 @{
     ViewData["Title"] = "Register";
 }

<h2>@ViewData["Title"]</h2>

<div class="row">
    <div class="col-md-4">
        <form asp-route-returnUrl="@Model.ReturnUrl" method="post">
            <h4>Create a new account.</h4>
            <hr />
            <div asp-validation-summary="All" class="text-danger"></div>

            <div class="form-group">
                <label asp-for="Input.Name"></label>
                <input asp-for="Input.Name" class="form-control" />
                <span asp-validation-for="Input.Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Input.DOB"></label>
                <input asp-for="Input.DOB" class="form-control" />
                <span asp-validation-for="Input.DOB" class="text-danger"></span>
            </div>

            <div class="form-group">
                <label asp-for="Input.Email"></label>
                <input asp-for="Input.Email" class="form-control" />
                <span asp-validation-for="Input.Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Input.Password"></label>
                <input asp-for="Input.Password" class="form-control" />
                <span asp-validation-for="Input.Password" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Input.ConfirmPassword"></label>
                <input asp-for="Input.ConfirmPassword" class="form-control" />
                <span asp-validation-for="Input.ConfirmPassword" class="text-danger"></span>
            </div>
            <button type="submit" class="btn btn-default">Register</button>
        </form>
    </div>
</div>

@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
```

Compile o projeto.

### Adicionar uma migração para os dados de usuário personalizada

- [Visual Studio](#)
- [CLI do .NET Core](#)

No Visual Studio **Package Manager Console**:

```
Add-Migration CustomUserData
Update-Database
```

## Teste de criar, exibir, baixar, excluir dados de usuário personalizada

Teste o aplicativo:

- Registre um novo usuário.
- Exibir os dados de usuário personalizada no `/Identity/Account/Manage` página.
- Baixar e exibir os dados pessoais de usuários da `/Identity/Account/Manage/PersonalData` página.

# Amostras de autenticação para o ASP.NET Core

30/01/2019 • 2 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

O [repositório do ASP.NET Core](#) contém os seguintes exemplos de autenticação na `AspNetCore/src/Security/samples` pasta:

- [Transformação de declarações](#)
- [Autenticação de cookie](#)
- [Provedor de política personalizada - `IAuthorizationPolicyProvider`](#)
- [Opções e esquemas de autenticação dinâmico](#)
- [Declarações externas](#)
- [Selecionando entre o cookie e o outro esquema de autenticação com base na solicitação](#)
- [Restringe o acesso a arquivos estáticos](#)

## Executar os exemplos

- Selecione uma [ramificação](#). Por exemplo, `release/2.2`
- Clone ou baixe o [repositório do ASP.NET Core](#).
- Verifique se você instalou o [SDK do .NET Core](#) versão que corresponde o clone do repositório do ASP.NET Core.
- Navegue até um exemplo na `AspNetCore/src/Security/samples` e executar o exemplo com `dotnet run`.

# Personalização de modelo de identidade no ASP.NET Core

27/09/2018 • 30 minutes to read • [Edit Online](#)

Por [Arthur Vickers](#)

Identidade do ASP.NET Core fornece uma estrutura para gerenciar e armazenar as contas de usuário em aplicativos ASP.NET Core. Identidade é adicionada ao seu projeto quando **contas de usuário individuais** é selecionado como o mecanismo de autenticação. Por padrão, a identidade faz usar de um Entity Framework (EF) modelo de dados central. Este artigo descreve como personalizar o modelo de identidade.

## Identidade e migrações do EF Core

Antes de examinar o modelo, ele é útil para entender como a identidade funciona com [migrações do EF Core](#) para criar e atualizar um banco de dados. No nível superior, o processo é:

1. Definir ou atualizar uma [modelo de dados no código](#).
2. Adicione uma migração para traduzir esse modelo para as alterações que podem ser aplicadas ao banco de dados.
3. Verifique que a migração representa corretamente suas intenções.
4. Aplique a migração para atualizar o banco de dados para ser sincronizado com o modelo.
5. Repita as etapas 1 a 4 para refinar o modelo ainda mais e manter o banco de dados em sincronia.

Use uma das abordagens a seguir para adicionar e aplicar as migrações:

- O **Package Manager Console** janela (PMC) se usando o Visual Studio. Para obter mais informações, consulte [ferramentas do EF Core PMC](#).
- A CLI do .NET Core se usando a linha de comando. Para obter mais informações, consulte [ferramentas de linha de comando do EF Core .NET](#).
- Clicar a **aplicar migrações** botão na página de erro quando o aplicativo é executado.

O ASP.NET Core tem um manipulador de página de erro de tempo de desenvolvimento. O manipulador pode aplicar migrações quando o aplicativo é executado. Para aplicativos de produção, geralmente é mais apropriado para gerar scripts SQL das migrações e implantar as alterações do banco de dados como parte de uma implantação de aplicativo e o banco de dados controlada.

Quando um novo aplicativo usando a identidade é criado, as etapas 1 e 2 acima já tem sido concluídas. Ou seja, o modelo de dados iniciais já existir e a migração inicial foi adicionada ao projeto. A migração inicial ainda precisa ser aplicada ao banco de dados. A migração inicial pode ser aplicada por meio de uma das seguintes abordagens:

- Execute `Update-Database` no PMC.
- Executar `dotnet ef database update` em um shell de comando.
- Clique o **aplicar migrações** botão na página de erro quando o aplicativo é executado.

Repita as etapas anteriores, como as alterações são feitas no modelo.

## O modelo de identidade

### Tipos de entidade

O modelo de identidade consiste dos seguintes tipos de entidade.

TIPO DE ENTIDADE	DESCRIÇÃO
User	Representa o usuário.
Role	Representa uma função.
UserClaim	Representa uma declaração que um usuário possui.
UserToken	Representa um token de autenticação para um usuário.
UserLogin	Associa um usuário com um logon.
RoleClaim	Representa uma declaração que é concedida a todos os usuários dentro de uma função.
UserRole	Uma entidade de junção que associa os usuários e funções.

## Relacionamentos de tipo de entidade

O [tipos de entidade](#) estão relacionados uns aos outros das seguintes maneiras:

- Cada `User` pode ter muitas `UserClaims`.
- Cada `User` pode ter muitas `UserLogins`.
- Cada `User` pode ter muitas `UserTokens`.
- Cada `Role` pode ter muitas associado `RoleClaims`.
- Cada `User` pode ter muitas associados `Roles` e cada `Role` pode ser associado a muitos `Users`. Essa é uma relação de muitos-para-muitos que requer uma tabela de junção no banco de dados. A tabela de junção é representada pelo `UserRole` entidade.

## Configuração de modelo padrão

Identidade define muitos *classes de contexto* que herdam de `DbContext` para configurar e usar o modelo. Essa configuração é feita usando o [EF Core Fluent API do Code First](#) no `OnModelCreating` método da classe de contexto. A configuração padrão é:

```
builder.Entity<TUser>(b =>
{
    // Primary key
    b.HasKey(u => u.Id);

    // Indexes for "normalized" username and email, to allow efficient lookups
    b.HasIndex(u => u.NormalizedUserName).HasName("UserNameIndex").IsUnique();
    b.HasIndex(u => u.NormalizedEmail).HasName("EmailIndex");

    // Maps to the AspNetUsers table
    b.ToTable("AspNetUsers");

    // A concurrency token for use with the optimistic concurrency checking
    b.Property(u => uConcurrencyStamp).IsConcurrencyToken();

    // Limit the size of columns to use efficient database types
    b.Property(u => u.UserName).HasMaxLength(256);
    b.Property(u => u.NormalizedUserName).HasMaxLength(256);
    b.Property(u => u.Email).HasMaxLength(256);
    b.Property(u => u.NormalizedEmail).HasMaxLength(256);

    // The relationships between User and other entity types
    // Note that these relationships are configured with no navigation properties
});
```

```

// Each User can have many UserClaims
b.HasMany<TUserClaim>().WithOne().HasForeignKey(uc => uc.UserId).IsRequired();

// Each User can have many UserLogins
b.HasMany<TUserLogin>().WithOne().HasForeignKey.ul => ul.UserId).IsRequired();

// Each User can have many UserTokens
b.HasMany<TUserToken>().WithOne().HasForeignKey(ut => ut.UserId).IsRequired();

// Each User can have many entries in the UserRole join table
b.HasMany<TUserRole>().WithOne().HasForeignKey(ur => ur.UserId).IsRequired();
});

builder.Entity<TUserClaim>(b =>
{
    // Primary key
    b.HasKey(uc => uc.Id);

    // Maps to the AspNetUserClaims table
    b.ToTable("AspNetUserClaims");
});

builder.Entity<TUserLogin>(b =>
{
    // Composite primary key consisting of the LoginProvider and the key to use
    // with that provider
    b.HasKey(l => new { l.LoginProvider, l.ProviderKey });

    // Limit the size of the composite key columns due to common DB restrictions
    b.Property(l => l.LoginProvider).HasMaxLength(128);
    b.Property(l => l.ProviderKey).HasMaxLength(128);

    // Maps to the AspNetUserLogins table
    b.ToTable("AspNetUserLogins");
});

builder.Entity<TUserToken>(b =>
{
    // Composite primary key consisting of the UserId, LoginProvider and Name
    b.HasKey(t => new { t.UserId, t.LoginProvider, t.Name });

    // Limit the size of the composite key columns due to common DB restrictions
    b.Property(t => t.LoginProvider).HasMaxLength(maxKeyLength);
    b.Property(t => t.Name).HasMaxLength(maxKeyLength);

    // Maps to the AspNetUserTokens table
    b.ToTable("AspNetUserTokens");
});

builder.Entity<TRole>(b =>
{
    // Primary key
    b.HasKey(r => r.Id);

    // Index for "normalized" role name to allow efficient lookups
    b.HasIndex(r => r.NormalizedName).HasName("RoleNameIndex").IsUnique();

    // Maps to the AspNetRoles table
    b.ToTable("AspNetRoles");

    // A concurrency token for use with the optimistic concurrency checking
    b.Property(r => rConcurrencyStamp).IsConcurrencyToken();

    // Limit the size of columns to use efficient database types
    b.Property(u => u.Name).HasMaxLength(256);
    b.Property(u => u.NormalizedName).HasMaxLength(256);

    // The relationships between Role and other entity types
    // Note that these relationships are configured with no navigation properties
}
);

```

```

// Each Role can have many entries in the UserRole join table
b.HasMany<TUserRole>().WithOne().HasForeignKey(ur => ur.RoleId).IsRequired();

// Each Role can have many associated RoleClaims
b.HasMany<TRoleClaim>().WithOne().HasForeignKey(rc => rc.RoleId).IsRequired();
});

builder.Entity<TRoleClaim>(b =>
{
    // Primary key
    b.HasKey(rc => rc.Id);

    // Maps to the AspNetRoleClaims table
    b.ToTable("AspNetRoleClaims");
});

builder.Entity<TUserRole>(b =>
{
    // Primary key
    b.HasKey(r => new { r.UserId, r.RoleId });

    // Maps to the AspNetUserRoles table
    b.ToTable("AspNetUserRoles");
});

```

## Tipos genéricos de modelo

Identidade define o padrão [Common Language Runtime](#) tipos (CLR) para cada um dos tipos de entidade listados acima. Esses tipos são prefixados com *identidade*:

- `IdentityUser`
- `IdentityRole`
- `IdentityUserClaim`
- `IdentityUserToken`
- `IdentityUserLogin`
- `IdentityRoleClaim`
- `IdentityUserRole`

Em vez de usar esses tipos diretamente, os tipos podem ser usados como classes base para os tipos do aplicativo. O `DbContext` classes definidas pela identidade são genéricas, de modo que diferentes tipos CLR podem ser usados para uma ou mais dos tipos de entidade no modelo. Esses tipos genéricos também permitem que o `User` tipo de dados (PK) chave primária a ser alterado.

Ao usar a identidade com suporte para funções, um `IdentityDbContext` classe deve ser usada. Por exemplo:

```

// Uses all the built-in Identity types
// Uses `string` as the key type
public class IdentityDbContext
    : IdentityDbContext<IdentityUser, IdentityRole, string>
{
}

// Uses the built-in Identity types except with a custom User type
// Uses `string` as the key type
public class IdentityDbContext<TUser>
    : IdentityDbContext<TUser, IdentityRole, string>
    where TUser : IdentityUser
{
}

// Uses the built-in Identity types except with custom User and Role types
// The key type is defined by TKey
public class IdentityDbContext<TUser, TRole, TKey> : IdentityDbContext<
    TUser, TRole, TKey, IdentityUserClaim<TKey>, IdentityUserRole<TKey>,
    IdentityUserLogin<TKey>, IdentityRoleClaim<TKey>, IdentityUserToken<TKey>>
    where TUser : IdentityUser<TKey>
    where TRole : IdentityRole<TKey>
    where TKey : IEquatable<TKey>
{
}

// No built-in Identity types are used; all are specified by generic arguments
// The key type is defined by TKey
public abstract class IdentityDbContext<
    TUser, TRole, TKey, TUserClaim, TUserRole, TUserLogin, TRoleClaim, TUserToken>
    : IdentityUserContext<TUser, TKey, TUserClaim, TUserLogin, TUserToken>
    where TUser : IdentityUser<TKey>
    where TRole : IdentityRole<TKey>
    where TKey : IEquatable<TKey>
    where TUserClaim : IdentityUserClaim<TKey>
    where TUserRole : IdentityUserRole<TKey>
    where TUserLogin : IdentityUserLogin<TKey>
    where TRoleClaim : IdentityRoleClaim<TKey>
    where TUserToken : IdentityUserToken<TKey>

```

Também é possível usar a identidade sem funções (apenas declarações), caso em que um [IdentityUserContext<TUser>](#) classe deve ser usada:

```

// Uses the built-in non-role Identity types except with a custom User type
// Uses `string` as the key type
public class IdentityUserContext<TUser>
    : IdentityUserContext<TUser, string>
    where TUser : IdentityUser
{
}

// Uses the built-in non-role Identity types except with a custom User type
// The key type is defined by TKey
public class IdentityUserContext<TUser, TKey> : IdentityUserContext<
    TUser, TKey, IdentityUserClaim<TKey>, IdentityUserLogin<TKey>,
    IdentityUserToken<TKey>>
    where TUser : IdentityUser<TKey>
    where TKey : IEquatable<TKey>
{
}

// No built-in Identity types are used; all are specified by generic arguments, with no roles
// The key type is defined by TKey
public abstract class IdentityUserContext<
    TUser, TKey, TUserClaim, TUserLogin, TUserToken> : DbContext
    where TUser : IdentityUser<TKey>
    where TKey : IEquatable<TKey>
    where TUserClaim : IdentityUserClaim<TKey>
    where TUserLogin : IdentityUserLogin<TKey>
    where TUserToken : IdentityUserToken<TKey>
{
}

```

## Personalizar o modelo

O ponto de partida para a personalização de modelo é derivado do tipo de contexto apropriado. Consulte a [modelar tipos genéricos](#) seção. Esse tipo de contexto geralmente é chamado `ApplicationDbContext` e é criada pelos modelos do ASP.NET Core.

O contexto é usado para configurar o modelo de duas maneiras:

- Fornecimento de entidade e tipos de chaves para os parâmetros de tipo genérico.
- Substituindo `OnModelCreating` para modificar o mapeamento desses tipos.

Ao substituir `OnModelCreating`, `base.OnModelCreating` deve ser chamado pela primeira vez; a configuração de substituição deve ser chamada em seguida. O EF Core geralmente tem uma política last-one-wins para a configuração. Por exemplo, se o `ToTable` método para um tipo de entidade é chamado pela primeira vez com o nome de uma tabela e, em seguida, novamente mais tarde com um nome de tabela diferente, o nome da tabela na segunda chamada é usado.

### Dados de usuário personalizada

[Dados de usuário personalizados](#) há suporte para herdar de `IdentityUser`. É comum para esse tipo de nome `ApplicationUser`:

```

public class ApplicationUser : IdentityUser
{
    public string CustomTag { get; set; }
}

```

Use o  `ApplicationUser`  tipo como um argumento genérico para o contexto:

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }
}
```

Não é necessário substituir `OnModelCreating` no `ApplicationDbContext` classe. O EF Core mapeia o `CustomTag` propriedade por convenção. No entanto, o banco de dados precisa ser atualizado para criar um novo `CustomTag` coluna. Para criar a coluna, adicione uma migração e, em seguida, atualize o banco de dados, conforme descrito em [identidade e migrações do EF Core](#).

Atualização `Startup.ConfigureServices` para usar o novo  `ApplicationUser`  classe:

```
services.AddDefaultIdentity< ApplicationUser >()
    .AddEntityFrameworkStores< ApplicationDbContext >()
    .AddDefaultUI();
```

No ASP.NET Core 2.1 ou posterior, a identidade é fornecida como uma biblioteca de classes Razor. Para obter mais informações, consulte [Identidade de Scaffold em projetos ASP.NET Core](#). Consequentemente, o código anterior requer uma chamada para `AddDefaultUI`. Se o scaffold de identidade foi usado para adicionar arquivos de identidade para o projeto, remova a chamada para `AddDefaultUI`. Para obter mais informações, consulte:

- [Identidade Scaffold](#)
- [Adicionar, baixar e excluir dados de usuário personalizada à identidade](#)

```
services.AddIdentity< ApplicationUser , IdentityRole >()
    .AddEntityFrameworkStores< ApplicationDbContext >()
    .AddDefaultTokenProviders();
```

```
services.AddIdentity< ApplicationUser , IdentityRole >()
    .AddEntityFrameworkStores< ApplicationDbContext , Guid >()
    .AddDefaultTokenProviders();
```

## Alterar o tipo de chave primária

Uma alteração para o tipo de dados da coluna PK depois que o banco de dados foi criado é um problema em muitos sistemas de banco de dados. Alterando a PK normalmente envolve descartar e recriar a tabela. Portanto, tipos de chave devem ser especificados na migração inicial quando o banco de dados é criado.

Siga estas etapas para alterar o tipo de PK:

1. Se o banco de dados foi criado antes da alteração de PK, execute `Drop-Database` (PMC) ou `dotnet ef database drop` (CLI do .NET Core) para excluí-lo.
2. Depois de confirmar a exclusão do banco de dados, remova a migração inicial com `Remove-Migration` (PMC) ou `dotnet ef migrations remove` (CLI do .NET Core).
3. Atualizar o `ApplicationDbContext` classe para derivar de `IdentityDbContext< TUser,TRole,TKey >`. Especifique o novo tipo de chave para `TKey`. Por exemplo, para usar um `Guid` tipo de chave:

```
public class ApplicationDbContext
    : IdentityDbContext<IdentityUser<Guid>, IdentityRole<Guid>, Guid>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }
}
```

No código anterior, as classes genéricas `IdentityUser<TKey>` e `IdentityRole<TKey>` deve ser especificado para usar o novo tipo de chave.

No código anterior, as classes genéricas `IdentityUser<TKey>` e `IdentityRole<TKey>` deve ser especificado para usar o novo tipo de chave.

`Startup.ConfigureServices` deve ser atualizado para usar o usuário genérico:

```
services.AddDefaultIdentity<IdentityUser<Guid>>()
    .AddEntityFrameworkStores<ApplicationContext>()
    .AddDefaultTokenProviders();
```

```
services.AddIdentity<IdentityUser<Guid>, IdentityRole>()
    .AddEntityFrameworkStores<ApplicationContext>()
    .AddDefaultTokenProviders();
```

```
services.AddIdentity<IdentityUser<Guid>, IdentityRole>()
    .AddEntityFrameworkStores<ApplicationContext, Guid>()
    .AddDefaultTokenProviders();
```

4. Se um personalizado  `ApplicationUser`  classe está sendo usado, atualize a classe para herdar de  `IdentityUser` . Por exemplo:

```
using System;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

public class ApplicationUser : IdentityUser<Guid>
{
    public string CustomTag { get; set; }
}
```

```
using System;
using Microsoft.AspNetCore.Identity;

public class ApplicationUser : IdentityUser<Guid>
{
    public string CustomTag { get; set; }
}
```

Atualização  `ApplicationContext`  para fazer referência a personalizada  `ApplicationUser`  classe:

```
public class ApplicationDbContext
    : IdentityDbContext<ApplicationUser, IdentityRole<Guid>, Guid>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }
}
```

Registrar a classe de contexto do banco de dados personalizados ao adicionar o serviço de identidade no `Startup.ConfigureServices` :

```
services.AddDefaultIdentity< ApplicationUser >()
    .AddEntityFrameworkStores< ApplicationDbContext >()
    .AddDefaultUI()
    .AddDefaultTokenProviders();
```

Tipo de dados da chave primária é inferido, analisando o `DbContext` objeto.

No ASP.NET Core 2.1 ou posterior, a identidade é fornecida como uma biblioteca de classes Razor. Para obter mais informações, consulte [Identidade de Scaffold em projetos ASP.NET Core](#). Consequentemente, o código anterior requer uma chamada para `AddDefaultUI`. Se o scaffolder de identidade foi usado para adicionar arquivos de identidade para o projeto, remova a chamada para `AddDefaultUI`.

```
services.AddIdentity< ApplicationUser, IdentityRole >()
    .AddEntityFrameworkStores< ApplicationDbContext, Guid >()
    .AddDefaultTokenProviders();
```

Tipo de dados da chave primária é inferido, analisando o `DbContext` objeto.

```
services.AddIdentity< ApplicationUser, IdentityRole >()
    .AddEntityFrameworkStores< ApplicationDbContext, Guid >()
    .AddDefaultTokenProviders();
```

O `AddEntityFrameworkStores` método aceita um `TKey` tipo que indica o tipo de dados da chave primária.

5. Se um personalizado `ApplicationRole` classe está sendo usado, atualize a classe para herdar de `IdentityRole< TKey >`. Por exemplo:

```
using System;
using Microsoft.AspNetCore.Identity;

public class ApplicationRole : IdentityRole< Guid >
{
    public string Description { get; set; }
}
```

Atualização `ApplicationDbContext` para fazer referência a personalizada `ApplicationRole` classe. Por exemplo, a classe a seguir faz referência a um personalizado  `ApplicationUser` e um personalizado  `ApplicationRole`:

```

using System;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

public class ApplicationDbContext :
    IdentityDbContext<ApplicationUser, ApplicationRole, Guid>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }
}

```

Registrar a classe de contexto do banco de dados personalizados ao adicionar o serviço de identidade no `Startup.ConfigureServices`:

```

public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<ApplicationUser, ApplicationRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultUI()
        .AddDefaultTokenProviders();

    services.AddMvc()
        .SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
}

```

Tipo de dados da chave primária é inferido, analisando o `DbContext` objeto.

No ASP.NET Core 2.1 ou posterior, a identidade é fornecida como uma biblioteca de classes Razor. Para obter mais informações, consulte [Identidade de Scaffold em projetos ASP.NET Core](#). Consequentemente, o código anterior requer uma chamada para [AddDefaultUI](#). Se o scaffold de identidade foi usado para adicionar arquivos de identidade para o projeto, remova a chamada para `AddDefaultUI`.

```

using System;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

public class ApplicationDbContext :
    IdentityDbContext<ApplicationUser, ApplicationRole, Guid>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }
}

```

Registrar a classe de contexto do banco de dados personalizados ao adicionar o serviço de identidade no `Startup.ConfigureServices`:

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<ApplicationUser, ApplicationRole>()
        .AddEntityFrameworkStores<ApplicationContext>()
        .AddDefaultTokenProviders();

    services.AddMvc()
        .AddRazorPagesOptions(options =>
    {
        options.Conventions.AuthorizeFolder("/Account/Manage");
        options.Conventions.AuthorizePage("/Account/Logout");
    });

    services.AddSingleton<IEmailSender, EmailSender>();
}

```

Tipo de dados da chave primária é inferido, analisando o [DbContext](#) objeto.

```

using System;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

public class ApplicationContext :
    IdentityDbContext<ApplicationUser, ApplicationRole, Guid>
{
    public ApplicationContext(DbContextOptions<ApplicationContext> options)
        : base(options)
    {
    }
}

```

Registrar a classe de contexto do banco de dados personalizados ao adicionar o serviço de identidade no [Startup.ConfigureServices](#) :

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<ApplicationUser, ApplicationRole>()
        .AddEntityFrameworkStores<ApplicationContext, Guid>()
        .AddDefaultTokenProviders();

    services.AddMvc();

    services.AddTransient<IEmailSender, AuthMessageSender>();
    services.AddTransient<ISmsSender, AuthMessageSender>();
}

```

O [AddEntityFrameworkStores](#) método aceita um [TKey](#) tipo que indica o tipo de dados da chave primária.

## Adicionar propriedades de navegação

Alterar a configuração de modelo para relações pode ser mais difícil do que fazendo outras alterações. Deve-se ter cuidado para substituir as relações existentes em vez de criar novas relações adicionais. Em particular, a relação alterada deve especificar o mesmo (FK) propriedade de chave estrangeira como a relação existente. Por exemplo, a relação entre [Users](#) e [UserClaims](#) é, por padrão, especificada da seguinte maneira:

```

builder.Entity<TUser>(b =>
{
    // Each User can have many UserClaims
    b.HasMany<TUserClaim>()
        .WithOne()
        .HasForeignKey(uc => uc.UserId)
        .IsRequired();
});

```

A FK para essa relação é especificada como o `UserClaim.UserId` propriedade. `HasMany` e `WithOne` são chamados sem argumentos para criar a relação sem propriedades de navegação.

Adicionar uma propriedade de navegação  `ApplicationUser` que permite que associado `UserClaims` seja referenciado do usuário:

```

public class ApplicationUser : IdentityUser
{
    public virtual ICollection<IdentityUserClaim<string>> Claims { get; set; }
}

```

O `TKey` para `IdentityUserClaim< TKey >` é o tipo especificado para o PK de usuários. Nesse caso, `TKey` é `string` porque os padrões estão sendo usados. Ele tem **não** o tipo de PK para o `UserClaim` tipo de entidade.

Agora que existe a propriedade de navegação, ele deve ser configurado no `OnModelCreating`:

```

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.Entity< ApplicationUser >(b =>
        {
            // Each User can have many UserClaims
            b.HasMany(e => e.Claims)
                .WithOne()
                .HasForeignKey(uc => uc.UserId)
                .IsRequired();
        });
    }
}

```

Observe que a relação é configurada exatamente como ele era antes, apenas com uma propriedade de navegação especificada na chamada para `HasMany`.

As propriedades de navegação só existem no modelo do EF, não o banco de dados. Como a FK para a relação não foi alterado, esse tipo de alteração de modelo não exige o banco de dados a ser atualizada. Isso pode ser verificado com a adição de uma migração após fazer a alteração. O `Up` e `Down` métodos estão vazios.

### **Adicionar todas as propriedades de navegação do usuário**

O exemplo a seguir usando a seção acima como orientação, configura as propriedades de navegação unidirecional para todas as relações em usuário:

```

public class ApplicationUser : IdentityUser
{
    public virtual ICollection<IdentityUserClaim<string>> Claims { get; set; }
    public virtual ICollection<IdentityUserLogin<string>> Logins { get; set; }
    public virtual ICollection<IdentityUserToken<string>> Tokens { get; set; }
    public virtual ICollection<IdentityUserRole<string>> UserRoles { get; set; }
}

```

```

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.Entity< ApplicationUser >(b =>
        {
            // Each User can have many UserClaims
            b.HasMany(e => e.Claims)
                .WithOne()
                .HasForeignKey(uc => uc.UserId)
                .IsRequired();

            // Each User can have many UserLogins
            b.HasMany(e => e.Logins)
                .WithOne()
                .HasForeignKey(ul => ul.UserId)
                .IsRequired();

            // Each User can have many UserTokens
            b.HasMany(e => e.Tokens)
                .WithOne()
                .HasForeignKey(ut => ut.UserId)
                .IsRequired();

            // Each User can have many entries in the UserRole join table
            b.HasMany(e => e.UserRoles)
                .WithOne()
                .HasForeignKey(ur => ur.UserId)
                .IsRequired();
        });
    }
}

```

## Adicionar propriedades de navegação do usuário e a função

Usando a seção acima como orientação, o exemplo a seguir configura as propriedades de navegação para todas as relações no usuário e a função:

```
public class ApplicationUser : IdentityUser
{
    public virtual ICollection<IdentityUserClaim<string>> Claims { get; set; }
    public virtual ICollection<IdentityUserLogin<string>> Logins { get; set; }
    public virtual ICollection<IdentityUserToken<string>> Tokens { get; set; }
    public virtual ICollection<ApplicationUserRole> UserRoles { get; set; }
}

public class ApplicationRole : IdentityRole
{
    public virtual ICollection<ApplicationUserRole> UserRoles { get; set; }
}

public class ApplicationUserRole : IdentityUserRole<string>
{
    public virtual ApplicationUser User { get; set; }
    public virtual ApplicationRole Role { get; set; }
}
```

```

public class ApplicationDbContext
    : IdentityDbContext<
        ApplicationUser, ApplicationRole, string,
        IdentityUserClaim<string>, ApplicationUserRole, IdentityUserLogin<string>,
        IdentityRoleClaim<string>, IdentityUserToken<string>>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.Entity< ApplicationUser >(b =>
        {
            // Each User can have many UserClaims
            b.HasMany(e => e.Claims)
                .WithOne()
                .HasForeignKey(uc => uc.UserId)
                .IsRequired();

            // Each User can have many UserLogins
            b.HasMany(e => e.Logins)
                .WithOne()
                .HasForeignKey(ul => ul.UserId)
                .IsRequired();

            // Each User can have many UserTokens
            b.HasMany(e => e.Tokens)
                .WithOne()
                .HasForeignKey(ut => ut.UserId)
                .IsRequired();

            // Each User can have many entries in the UserRole join table
            b.HasMany(e => e.UserRoles)
                .WithOne(e => e.User)
                .HasForeignKey(ur => ur.UserId)
                .IsRequired();
        });

        modelBuilder.Entity< ApplicationRole >(b =>
        {
            // Each Role can have many entries in the UserRole join table
            b.HasMany(e => e.UserRoles)
                .WithOne(e => e.Role)
                .HasForeignKey(ur => ur.RoleId)
                .IsRequired();
        });
    }
}

```

Notas:

- Este exemplo também inclui o `UserRole` entidade, que é necessário para navegar pela relação muitos-para-muitos dos usuários às funções de ingresso.
- Lembre-se de alterar os tipos das propriedades de navegação de refletir isso `ApplicationXxx` tipos agora estão sendo usados em vez de `IdentityXxx` tipos.
- Lembre-se de usar o `ApplicationXxx` no genérico `ApplicationContext` definição.

### Adicionar todas as propriedades de navegação

Usando a seção acima como orientação, o exemplo a seguir configura as propriedades de navegação para todas as

relações em todos os tipos de entidade:

```
public class ApplicationUser : IdentityUser
{
    public virtual ICollection<ApplicationUserClaim> Claims { get; set; }
    public virtual ICollection<ApplicationUserLogin> Logins { get; set; }
    public virtual ICollection<ApplicationUserToken> Tokens { get; set; }
    public virtual ICollection<ApplicationUserRole> UserRoles { get; set; }
}

public class ApplicationRole : IdentityRole
{
    public virtual ICollection<ApplicationUserRole> UserRoles { get; set; }
    public virtual ICollection<ApplicationRoleClaim> RoleClaims { get; set; }
}

public class ApplicationUserRole : IdentityUserRole<string>
{
    public virtual ApplicationUser User { get; set; }
    public virtual ApplicationRole Role { get; set; }
}

public class ApplicationUserClaim : IdentityUserClaim<string>
{
    public virtual ApplicationUser User { get; set; }
}

public class ApplicationUserLogin : IdentityUserLogin<string>
{
    public virtual ApplicationUser User { get; set; }
}

public class ApplicationRoleClaim : IdentityRoleClaim<string>
{
    public virtual ApplicationRole Role { get; set; }
}

public class ApplicationUserToken : IdentityUserToken<string>
{
    public virtual ApplicationUser User { get; set; }
}
```

```

public class ApplicationDbContext
    : IdentityDbContext<
        ApplicationUser, ApplicationRole, string,
        ApplicationUserClaim, ApplicationUserRole, ApplicationUserLogin,
        ApplicationRoleClaim, ApplicationUserToken>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.Entity<ApplicationUser>(b =>
        {
            // Each User can have many UserClaims
            b.HasMany(e => e.Claims)
                .WithOne(e => e.User)
                .HasForeignKey(uc => uc.UserId)
                .IsRequired();

            // Each User can have many UserLogins
            b.HasMany(e => e.Logins)
                .WithOne(e => e.User)
                .HasForeignKey(ul => ul.UserId)
                .IsRequired();

            // Each User can have many UserTokens
            b.HasMany(e => e.Tokens)
                .WithOne(e => e.User)
                .HasForeignKey(ut => ut.UserId)
                .IsRequired();

            // Each User can have many entries in the UserRole join table
            b.HasMany(e => e.UserRoles)
                .WithOne(e => e.User)
                .HasForeignKey(ur => ur.UserId)
                .IsRequired();
        });

        modelBuilder.Entity<ApplicationRole>(b =>
        {
            // Each Role can have many entries in the UserRole join table
            b.HasMany(e => e.UserRoles)
                .WithOne(e => e.Role)
                .HasForeignKey(ur => ur.RoleId)
                .IsRequired();

            // Each Role can have many associated RoleClaims
            b.HasMany(e => e.RoleClaims)
                .WithOne(e => e.Role)
                .HasForeignKey(rc => rc.RoleId)
                .IsRequired();
        });
    }
}

```

## Usar chaves compostas

As seções anteriores demonstrou como alterar o tipo da chave usada no modelo de identidade. Alterando o modelo de chave de identidade para usar chaves compostas não é suportado nem recomendado. Usar uma chave composta com identidade envolve a alteração como o código do Gerenciador de identidade interage com o modelo. Essa personalização está além do escopo deste documento.

## Alterar nomes de tabela/coluna e as facetas

Para alterar os nomes das tabelas e colunas, chame `base.OnModelCreating`. Em seguida, adicione a configuração para substituir qualquer um dos padrões. Por exemplo, para alterar o nome de todas as tabelas de identidade:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    modelBuilder.Entity<IdentityUser>(b =>
    {
        b.ToTable("MyUsers");
    });

    modelBuilder.Entity<IdentityUserClaim<string>>(b =>
    {
        b.ToTable("MyUserClaims");
    });

    modelBuilder.Entity<IdentityUserLogin<string>>(b =>
    {
        b.ToTable("MyUserLogins");
    });

    modelBuilder.Entity<IdentityUserToken<string>>(b =>
    {
        b.ToTable("MyUserTokens");
    });

    modelBuilder.Entity<IdentityRole>(b =>
    {
        b.ToTable("MyRoles");
    });

    modelBuilder.Entity<IdentityRoleClaim<string>>(b =>
    {
        b.ToTable("MyRoleClaims");
    });

    modelBuilder.Entity<IdentityUserRole<string>>(b =>
    {
        b.ToTable("MyUserRoles");
    });
}
```

Esses exemplos usam os tipos de identidade padrão. Se usar um tipo de aplicativo, como  `ApplicationUser` , configurar esse tipo em vez do tipo padrão.

O exemplo a seguir altera alguns nomes de coluna:

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    modelBuilder.Entity<IdentityUser>(b =>
    {
        b.Property(e => e.Email).HasColumnName("EMail");
    });

    modelBuilder.Entity<IdentityUserClaim<string>>(b =>
    {
        b.Property(e => e.ClaimType).HasColumnName("CType");
        b.Property(e => e.ClaimValue).HasColumnName("CValue");
    });
}

```

Alguns tipos de colunas de banco de dados podem ser configurados com determinados *facetas* (por exemplo, o máximo `string` comprimento permitido). O exemplo a seguir define os comprimentos máximos da coluna para várias `string` propriedades no modelo:

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    modelBuilder.Entity<IdentityUser>(b =>
    {
        b.Property(u => u.UserName).HasMaxLength(128);
        b.Property(u => u.NormalizedUserName).HasMaxLength(128);
        b.Property(u => u.Email).HasMaxLength(128);
        b.Property(u => u.NormalizedEmail).HasMaxLength(128);
    });

    modelBuilder.Entity<IdentityUserToken<string>>(b =>
    {
        b.Property(t => t.LoginProvider).HasMaxLength(128);
        b.Property(t => t.Name).HasMaxLength(128);
    });
}

```

## Mapear para um esquema diferente

Esquemas podem se comportar de maneira diferente em provedores de banco de dados. Para o SQL Server, o padrão é criar todas as tabelas na `dbo` esquema. As tabelas podem ser criadas em um esquema diferente. Por exemplo:

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    modelBuilder.HasDefaultSchema("notdbo");
}

```

## Carregamento lento

Nesta seção, o suporte para proxies de carregamento lento no modelo de identidade é adicionado. Carregamento lento é útil, pois ele permite que as propriedades de navegação a ser usada sem primeiro garantir que eles são carregados.

Tipos de entidade podem ser feitos adequados para carregamento lento de diversas maneiras, conforme descrito na [documentação do EF Core](#). Para simplificar, use proxies de carregamento lento, que requer:

- Instalação dos `entityframeworkcore` pacote.
- Uma chamada para `UseLazyLoadingProxies` dentro de `AddDbContext<TContext>`.
- Tipos de entidade pública com `public virtual` propriedades de navegação.

O exemplo a seguir demonstra a chamada `UseLazyLoadingProxies` em `Startup.ConfigureServices`:

```
services
    .AddDbContext<ApplicationDbContext>(
        b => b.UseSqlServer(connectionString)
            .UseLazyLoadingProxies()
    .AddDefaultIdentity< ApplicationUser >()
    .AddEntityFrameworkStores< ApplicationDbContext >();
```

Consulte os exemplos anteriores para obter orientação sobre como adicionar propriedades de navegação para os tipos de entidade.

## Recursos adicionais

- [Identidade de Scaffold em projetos ASP.NET Core](#)

# Opções de autenticação de comunidade OSS para ASP.NET Core

22/06/2018 • 2 minutes to read • [Edit Online](#)

Esta página contém as opções de autenticação fornecido pela comunidade de código aberto para o ASP.NET Core. Esta página é atualizada periodicamente como novos provedores de se tornar disponíveis.

## Provedores de autenticação de sistemas operacionais

A lista a seguir está classificada alfabeticamente.

NOME	DESCRIÇÃO
<a href="#">AspNet.Security.OpenIdConnect.Server (ASOS)</a>	ASOS é um nível baixo, o primeiro protocolo OpenID Connect server framework para ASP.NET Core e OWIN/Katana.
<a href="#">Cierge</a>	Cierge é um servidor de OpenID Connect que manipula a inscrição de usuário, logon, perfis, gerenciamento e logons sociais.
<a href="#">Servidor Gluu</a>	Enterprise estiver pronto, abra o software de origem para a identidade, gerenciamento de acesso (IAM) e logon único (SSO). Para obter mais informações, consulte o <a href="#">documento do produto Gluu</a> .
<a href="#">IdentityServer</a>	IdentityServer é uma estrutura de OpenID Connect e OAuth 2.0 para ASP.NET Core, certificada oficialmente a base de OpenID e sob controle do Foundation .NET. Para obter mais informações, consulte <a href="#">bem-vindo ao IdentityServer4 (documentação)</a> .
<a href="#">OpenIddict</a>	OpenIddict é um servidor de OpenID Connect de fácil de usar para ASP.NET Core.

Para adicionar um provedor, [editar esta página](#).

# Configurar a identidade do ASP.NET Core

10/10/2018 • 7 minutes to read • [Edit Online](#)

Identidade do ASP.NET Core usa valores padrão para configurações como a política de senha, bloqueio e configuração de cookie. Essas configurações podem ser substituídas no `Startup` classe.

## Opções de identidade

O `IdentityOptions` classe representa as opções que podem ser usadas para configurar o sistema de identidade.

`IdentityOptions` deve ser definido **após** chamando `AddIdentity` ou `AddDefaultIdentity`.

### Identidade baseada em declarações

`IdentityOptions.ClaimsIdentity` Especifica o `ClaimsIdentityOptions` com as propriedades mostradas na tabela a seguir.

PROPRIEDADE	DESCRIÇÃO	PADRÃO
<code>RoleClaimType</code>	Obtém ou define o tipo de declaração usado para uma declaração de função.	<code>ClaimTypes.Role</code>
<code>SecurityStampClaimType</code>	Obtém ou define o tipo de declaração usado para a declaração de carimbo de segurança.	<code>AspNet.Identity.SecurityStamp</code>
<code>UserIdClaimType</code>	Obtém ou define o tipo de declaração usado para a declaração do identificador de usuário.	<code>ClaimTypes.NameIdentifier</code>
<code>UserNameClaimType</code>	Obtém ou define o tipo de declaração usado para a declaração de nome de usuário.	<code>ClaimTypes.Name</code>

### Bloqueio

O bloqueio é definido na `PasswordSignInAsync` método:

```

public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl = returnUrl ?? Url.Content("~/");

    if (ModelState.IsValid)
    {
        var result = await _signInManager.PasswordSignInAsync(Input.Email,
            Input.Password, Input.RememberMe,
            lockoutOnFailure: false);
        if (result.Succeeded)
        {
            _logger.LogInformation("User logged in.");
            return LocalRedirect(returnUrl);
        }
        if (result.RequiresTwoFactor)
        {
            return RedirectToPage("./LoginWith2fa", new { ReturnUrl = returnUrl,
                Input.RememberMe });
        }
        if (result.IsLockedOut)
        {
            _logger.LogWarning("User account locked out.");
            return RedirectToPage("./Lockout");
        }
        else
        {
            ModelState.AddModelError(string.Empty, "Invalid login attempt.");
            return Page();
        }
    }

    // If we got this far, something failed, redisplay form
    return Page();
}

```

O código anterior se baseia o `Login` modelo de identidade.

Opções de bloqueio são definidas na `StartUp.ConfigureServices` :

```

services.Configure<IdentityOptions>(options =>
{
    // Default Lockout settings.
    options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);
    options.Lockout.MaxFailedAccessAttempts = 5;
    options.Lockout.AllowedForNewUsers = true;
});

```

O código anterior define o `IdentityOptions LockoutOptions` com valores padrão.

Uma autenticação bem-sucedida redefine a contagem de tentativas de acesso com falha e redefine o relógio.

`IdentityOptions.Lockout` Especifica o `LockoutOptions` com as propriedades mostradas na tabela.

PROPRIEDADE	DESCRIÇÃO	PADRÃO
<code>AllowedForNewUsers</code>	Determina se um novo usuário poderá ser bloqueado.	<code>true</code>
<code>DefaultLockoutTimeSpan</code>	A quantidade de tempo um usuário está bloqueado quando ocorre um bloqueio.	5 minutos

PROPRIEDADE	DESCRIÇÃO	PADRÃO
MaxFailedAccessAttempts	O número de tentativas de acesso com falha até que um usuário está bloqueado, se o bloqueio está habilitado.	5

## Senha

Por padrão, a identidade requer que as senhas conter um caractere maiusculo, caractere minúsculo, um dígito e um caractere não alfanumérico. As senhas devem ter pelo menos seis caracteres. [PasswordOptions](#) podem ser definidas na `Startup.ConfigureServices`.

```
services.Configure<IdentityOptions>(options =>
{
    // Default Password settings.
    options.Password.RequireDigit = true;
    options.Password.RequireLowercase = true;
    options.Password.RequireNonAlphanumeric = true;
    options.Password.RequireUppercase = true;
    options.Password.RequiredLength = 6;
    options.Password.RequiredUniqueChars = 1;
});
```

```
services.AddIdentity<ApplicationUser, IdentityRole>(options =>
{
    // Password settings
    options.Password.RequireDigit = true;
    options.Password.RequiredLength = 8;
    options.Password.RequiredUniqueChars = 2;
    options.Password.RequireLowercase = true;
    options.Password.RequireNonAlphanumeric = true;
    options.Password.RequireUppercase = true;
})
.AddEntityFrameworkStores<ApplicationContext>()
.AddDefaultTokenProviders();
```

```
services.Configure<IdentityOptions>(options =>
{
    // Password settings
    options.Password.RequireDigit = true;
    options.Password.RequiredLength = 8;
    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequireUppercase = true;
    options.Password.RequireLowercase = false;
});
```

`IdentityOptions.Password` especifica o [PasswordOptions](#) com as propriedades mostradas na tabela.

PROPRIEDADE	DESCRIÇÃO	PADRÃO
RequireDigit	Requer um número entre 0 a 9 na senha.	<code>true</code>
RequiredLength	O comprimento mínimo da senha.	6
RequireLowercase	Requer um caractere minúsculo na senha.	<code>true</code>

PROPRIEDADE	DESCRIÇÃO	PADRÃO
RequireNonAlphanumeric	Requer um caractere não alfanuméricos na senha.	true
RequiredUniqueChars	Só se aplica ao ASP.NET Core 2.0 ou posterior.  Requer o número de caracteres distintas na senha.	1
RequireUppercase	Requer um caractere maiusculo na senha.	true
PROPRIEDADE	DESCRIÇÃO	PADRÃO
RequireDigit	Requer um número entre 0 a 9 na senha.	true
RequiredLength	O comprimento mínimo da senha.	6
RequireLowercase	Requer um caractere minúsculo na senha.	true
RequireNonAlphanumeric	Requer um caractere não alfanuméricos na senha.	true
RequireUppercase	Requer um caractere maiusculo na senha.	true

## Entrar

O seguinte código define `SignIn` configurações (para valores padrão):

```
services.Configure<IdentityOptions>(options =>
{
    // Default SignIn settings.
    options.SignIn.RequireConfirmedEmail = true;
    options.SignIn.RequireConfirmedPhoneNumber = false;
});
```

```
services.AddIdentity<ApplicationUser, IdentityRole>(options =>
{
    // Signin settings
    options.SignIn.RequireConfirmedEmail = true;
    options.SignIn.RequireConfirmedPhoneNumber = false;
})
.AddEntityFrameworkStores<ApplicationContext>()
.AddDefaultTokenProviders();
```

`IdentityOptions.SignIn` Especifica o `SignInOptions` com as propriedades mostradas na tabela.

PROPRIEDADE	DESCRIÇÃO	PADRÃO
RequireConfirmedEmail	Requer um email confirmado para entrar.	false

PROPRIEDADE	DESCRIÇÃO	PADRÃO
RequireConfirmedPhoneNumber	Requer um número de telefone confirmado entrar.	false

## Tokens

[IdentityOptions.Tokens](#) Especifica o [TokenOptions](#) com as propriedades mostradas na tabela.

PROPRIEDADE	DESCRIÇÃO
AuthenticatorTokenProvider	Obtém ou define o <a href="#">AuthenticatorTokenProvider</a> usado para validar entradas de dois fatores com um autenticador.
ChangeEmailTokenProvider	Obtém ou define o <a href="#">ChangeEmailTokenProvider</a> usado para gerar tokens usados nos emails de confirmação de alteração de email.
ChangePhoneNumberTokenProvider	Obtém ou define o <a href="#">ChangePhoneNumberTokenProvider</a> usada para gerar tokens usados ao alterar os números de telefone.
EmailConfirmationTokenProvider	Obtém ou define o provedor de token usado para gerar tokens usados nos emails de confirmação de conta.
PasswordResetTokenProvider	Obtém ou define o <a href="#">IUserTwoFactorTokenProvider</a> usado para gerar tokens usados nos emails de redefinição de senha.
ProviderMap	Usado para construir um <a href="#">provedor de Token de usuário</a> com a chave usada como o nome do provedor.

## User

```
services.Configure<IdentityOptions>(options =>
{
    // Default User settings.
    options.User.AllowedUserNameCharacters =
        "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-_@+";
    options.User.RequireUniqueEmail = true;

});
```

[IdentityOptions.User](#) Especifica o [UserOptions](#) com as propriedades mostradas na tabela.

PROPRIEDADE	DESCRIÇÃO	PADRÃO
AllowedUserNameCharacters	Caracteres permitidos no nome de usuário.	abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789 -_@+
RequireUniqueEmail	Requer que cada usuário tenha um email exclusivo.	false

## Configurações de cookie

Configurar o cookie do aplicativo em [Startup.ConfigureServices](#). [ConfigureApplicationCookie](#) deve ser chamado **após** chamando [AddIdentity](#) OU [AddDefaultIdentity](#).

```
services.ConfigureApplicationCookie(options =>
{
    options.AccessDeniedPath = "/Identity/Account/AccessDenied";
    options.Cookie.Name = "YourAppCookieName";
    options.Cookie.HttpOnly = true;
    options.ExpireTimeSpan = TimeSpan.FromMinutes(60);
    options.LoginPath = "/Identity/Account/Login";
    // ReturnUrlParameter requires
    //using Microsoft.AspNetCore.Authentication.Cookies;
    options.ReturnUrlParameter = CookieAuthenticationDefaults.ReturnUrlParameter;
    options SlidingExpiration = true;
});
```

```
services.ConfigureApplicationCookie(options =>
{
    options.AccessDeniedPath = "/Account/AccessDenied";
    options.Cookie.Name = "YourAppCookieName";
    options.Cookie.HttpOnly = true;
    options.ExpireTimeSpan = TimeSpan.FromMinutes(60);
    options.LoginPath = "/Account/Login";
    // ReturnUrlParameter requires `using Microsoft.AspNetCore.Authentication.Cookies;`
    options.ReturnUrlParameter = CookieAuthenticationDefaults.ReturnUrlParameter;
    options SlidingExpiration = true;
});
```

```
services.Configure<IdentityOptions>(options =>
{
    // Cookie settings
    options.Cookies.ApplicationCookie.CookieName = "YourAppCookieName";
    options.Cookies.ApplicationCookie.ExpireTimeSpan = TimeSpan.FromDays(150);
    options.Cookies.ApplicationCookie.LoginPath = "/Account/LogIn";
    options.Cookies.ApplicationCookie.AccessDeniedPath = "/Account/AccessDenied";
    options.Cookies.ApplicationCookie.AutomaticAuthenticate = true;
    // Requires `using Microsoft.AspNetCore.Authentication.Cookies;`
    options.Cookies.ApplicationCookie.AuthenticationScheme =
        CookieAuthenticationDefaults.AuthenticationScheme;
    options.Cookies.ApplicationCookie.ReturnUrlParameter = CookieAuthenticationDefaults.ReturnUrlParameter;
});
```

Para obter mais informações, consulte [CookieAuthenticationOptions](#).

# Configurar a autenticação do Windows no ASP.NET Core

16/01/2019 • 14 minutes to read • [Edit Online](#)

Por [Scott Addie](#) e [Luke Latham](#)

Autenticação do Windows podem ser configuradas para aplicativos ASP.NET Core hospedados com [IIS](#) ou [HTTP.sys](#).

Autenticação do Windows se baseia no sistema operacional para autenticar usuários de aplicativos ASP.NET Core. Você pode usar a autenticação do Windows quando o servidor é executado em uma rede corporativa usando identidades de domínio do Active Directory ou contas do Windows para identificar os usuários. Autenticação do Windows é mais adequada para ambientes de intranet em que os usuários, aplicativos cliente e servidores web pertencem ao mesmo domínio do Windows.

## Habilitar a autenticação do Windows em um aplicativo ASP.NET Core

O **aplicativo Web** modelo disponível por meio do Visual Studio ou a CLI do .NET Core pode ser configurado para dar suporte à autenticação do Windows.

- [Visual Studio](#)
- [CLI do .NET Core](#)

### Usar o modelo de aplicativo de autenticação do Windows para um novo projeto

No Visual Studio:

1. Criar um novo **aplicativo Web ASP.NET Core**.
2. Selecione **aplicativo Web** da lista de modelos.
3. Selecione o **alterar autenticação** botão e selecione **autenticação do Windows**.

Execute o aplicativo. O nome de usuário é exibido na interface do usuário do aplicativo renderizado.

### Configuração manual para um projeto existente

As propriedades do projeto permitem que você habilitar a autenticação do Windows e desabilite a autenticação anônima:

1. Clique com botão direito no projeto no Visual Studio **Gerenciador de soluções** e selecione **propriedades**.
2. Selecione a guia **Depurar**.
3. Desmarque a caixa de seleção **habilitar a autenticação anônima**.
4. Marque a caixa de seleção **habilitar a autenticação do Windows**.

Como alternativa, as propriedades podem ser configuradas na `iisSettings` nó do `launchsettings.json` arquivo:

```
"iisSettings": {  
    "windowsAuthentication": true,  
    "anonymousAuthentication": false,  
    "iisExpress": {  
        "applicationUrl": "http://localhost:52171/",  
        "sslPort": 0  
    }  
}
```

Ao modificar um projeto existente, confirme se o arquivo de projeto inclui uma referência de pacote para o metapacote do Microsoft ou o Microsoft.AspNetCore.Authentication pacote do NuGet.

## Habilitar a autenticação do Windows com o IIS

O IIS usa a [módulo do ASP.NET Core](#) para hospedar aplicativos ASP.NET Core. Autenticação do Windows está configurada para o IIS por meio de *Web.config* arquivo. As seções a seguir mostram como:

- Forneça um local *Web.config* arquivo que ativa a autenticação do Windows no servidor quando o aplicativo é implantado.
- Use o Gerenciador do IIS para configurar o *Web.config* arquivo de um aplicativo ASP.NET Core que já tenha sido implantado no servidor.

### Configuração do IIS

Se você ainda não fez isso, habilite o IIS para hospedar aplicativos ASP.NET Core. Para obter mais informações, consulte [Hospedar o ASP.NET Core no Windows com o IIS](#).

Habilite o serviço de função do IIS para autenticação do Windows. Para obter mais informações, consulte [habilitar autenticação do Windows nos serviços de função do IIS \(consulte a etapa 2\)](#).

Middleware de integração do IIS está configurado para autenticar solicitações de automaticamente por padrão. Para obter mais informações, consulte [Host ASP.NET Core no Windows com o IIS: Opções do IIS \(AutomaticAuthentication\)](#).

O módulo do ASP.NET está configurado para encaminhar o token de autenticação do Windows para o aplicativo por padrão. Para obter mais informações, consulte [referência de configuração do módulo do ASP.NET Core: Atributos do elemento aspNetCore](#).

### Criar um novo site do IIS

Especifique um nome e uma pasta e permitir que ele crie um novo pool de aplicativos.

### Habilitar a autenticação do Windows para o aplicativo no IIS

Use **qualquer** das seguintes abordagens:

- [Configuração do lado do desenvolvimento antes de publicar o aplicativo \(recomendado\)](#)
- [Configuração do lado do servidor depois de publicar o aplicativo](#)

#### Configuração do lado do desenvolvimento com um arquivo Web.config local

Execute as seguintes etapas **antes de** você [publicar e implantar seu projeto](#).

Adicione o seguinte *Web.config* arquivo para a raiz do projeto:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <location path=". " inheritInChildApplications="false">
    <system.webServer>
      <security>
        <authentication>
          <anonymousAuthentication enabled="false" />
          <windowsAuthentication enabled="true" />
        </authentication>
      </security>
    </system.webServer>
  </location>
</configuration>
```

Quando o projeto é publicado pelo SDK (sem o `<IsTransformWebConfigDisabled>` propriedade definida como `true` no arquivo de projeto), publicado *Web.config* arquivo inclui o `<location><system.webServer><security><authentication>` seção. Para obter mais informações sobre o `<IsTransformWebConfigDisabled>` propriedade, consulte [Hospedar o ASP.NET Core no Windows com o IIS](#).

#### Configuração do lado do servidor com o Gerenciador do IIS

Execute as seguintes etapas **após** você [publicar e implantar seu projeto](#).

1. No Gerenciador do IIS, selecione o site do IIS sob o **Sites** nó do **conexões** barra lateral.
2. Clique duas vezes em **autenticação** na **IIS** área.
3. Selecione **autenticação anônima**. Selecione **desabilite** na **ações** barra lateral.
4. Selecione **autenticação do Windows**. Selecione **habilitar** na **ações** barra lateral.

Quando essas ações são executadas, o Gerenciador do IIS modifica o aplicativo *Web.config* arquivo. Um `<system.webServer><security><authentication>` nó for adicionado com configurações atualizadas para `anonymousAuthentication` e `windowsAuthentication`:

```
<system.webServer>
  <security>
    <authentication>
      <anonymousAuthentication enabled="false" />
      <windowsAuthentication enabled="true" />
    </authentication>
  </security>
</system.webServer>
```

O `<system.webServer>` seção adicionada para o *Web.config* arquivo pelo Gerenciador do IIS está fora do aplicativo `<location>` seção adicionada pelo SDK do .NET Core quando o aplicativo for publicado. Porque a seção é adicionada fora das `<location>` nó, as configurações são herdadas por qualquer **subaplicativos** ao aplicativo atual. Para impedir a herança, move a adicionada `<security>` seção dentro do `<location><system.webServer>` seção que o SDK fornecido.

Quando o Gerenciador do IIS é usado para adicionar a configuração do IIS, ele afeta somente o aplicativo *Web.config* arquivo no servidor. Uma implantação subsequente do aplicativo pode substituir as configurações no servidor se a cópia do servidor do *Web.config* é substituído do projeto *Web.config* arquivo. Use **qualquer** das abordagens a seguir para gerenciar as configurações:

- Use o Gerenciador do IIS para redefinir as configurações na *Web.config* depois que o arquivo será substituído na implantação de arquivos.
- Adicionar um *arquivo Web.config* para o aplicativo localmente com as configurações. Para obter mais informações, consulte o [configuração do lado do desenvolvimento](#) seção.

#### Publicar e implantar seu projeto para a pasta de site do IIS

Usando o Visual Studio ou a CLI do .NET Core, publicar e implantar o aplicativo para a pasta de destino.

Para obter mais informações sobre hospedagem com o IIS, publicação e implantação, consulte os tópicos a seguir:

- [dotnet publish](#)
- [Hospedar o ASP.NET Core no Windows com o IIS](#)
- [Módulo do ASP.NET Core](#)
- [Perfis de publicação do Visual Studio para a implantação do aplicativo ASP.NET Core](#)

Inicie o aplicativo para verificar se a autenticação do Windows está funcionando.

## Habilitar a autenticação do Windows com o HTTP.sys

Embora o Kestrel não dê suporte à autenticação do Windows, você pode usar [HTTP.sys](#) para dar suporte a cenários que são hospedados no Windows. O exemplo a seguir configura o host de web do aplicativo para usar o HTTP.sys com a autenticação do Windows:

```
public class Program
{
    public static void Main(string[] args) =>
        BuildWebHost(args).Run();

    public static IWebHost BuildWebHost(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>()
            .UseHttpSys(options =>
            {
                options.Authentication.Schemes =
                    AuthenticationSchemes.NTLM | AuthenticationSchemes.Negotiate;
                options.Authentication.AllowAnonymous = false;
            })
            .Build();
}
```

### NOTE

O HTTP.sys delega à autenticação de modo kernel com o protocolo de autenticação Kerberos. Não há suporte para autenticação de modo de usuário com o Kerberos e o HTTP.sys. A conta do computador precisa ser usada para descriptografar o token/tíquete do Kerberos que é obtido do Active Directory e encaminhado pelo cliente ao servidor para autenticar o usuário. Registre o SPN (nome da entidade de serviço) do host, não do usuário do aplicativo.

### NOTE

Não há suporte para http.sys no Nano Server versão 1709 ou posterior. Para usar a autenticação do Windows e o HTTP.sys com o Nano Server, use uma [contêiner de Server Core \(microsoft/windowsservercore\)](#). Para obter mais informações sobre o Server Core, consulte [qual é a opção de instalação Server Core no Windows Server?](#).

## Trabalhar com a autenticação do Windows

Estado da configuração do acesso anônimo determina o modo no qual o `[Authorize]` e `[AllowAnonymous]` atributos são usados no aplicativo. As seções a seguir explicam como lidar com os estados de configuração não permitidos e têm permissão de acesso anônimo.

### Não permitir acesso anônimo

Quando a autenticação do Windows está habilitada e o acesso anônimo é desabilitado, o `[Authorize]` e `[AllowAnonymous]` atributos não têm nenhum efeito. Se o site do IIS (ou HTTP.sys) estiver configurado para não permitir acesso anônimo, a solicitação nunca atinge seu aplicativo. Por esse motivo, o `[AllowAnonymous]` atributo não é aplicável.

## Permitir o acesso anônimo

Quando a autenticação do Windows e o acesso anônimo estão habilitados, use o `[Authorize]` e `[AllowAnonymous]` atributos. O `[Authorize]` atributo permite que você possa proteger partes do aplicativo que realmente exigem a autenticação do Windows. O `[AllowAnonymous]` substituições de atributo `[Authorize]` atributo uso dentro de aplicativos que permitem o acesso anônimo. Ver [autorização simples](#) para obter detalhes de uso do atributo.

No ASP.NET Core 2.x, o `[Authorize]` atributo requer configuração adicional no `Startup.cs` desafiar solicitações anônimas para a autenticação do Windows. A configuração recomendada varia um pouco com base no servidor web que está sendo usado.

### NOTE

Por padrão, os usuários que não têm autorização para acessar uma página são apresentados com uma resposta HTTP 403 vazia. O [StatusCodePages middleware](#) pode ser configurado para fornecer aos usuários uma melhor experiência de "Acesso negado".

## IIS

Se usar o IIS, adicione o seguinte para o `ConfigureServices` método:

```
// IISDefaults requires the following import:  
// using Microsoft.AspNetCore.Server.IISIntegration;  
services.AddAuthentication(IISDefaults.AuthenticationScheme);
```

## HTTP.sys

Se usando HTTP.sys, adicione o seguinte para o `ConfigureServices` método:

```
// HttpSysDefaults requires the following import:  
// using Microsoft.AspNetCore.Server.HttpSys;  
services.AddAuthentication(HttpSysDefaults.AuthenticationScheme);
```

## Representação

ASP.NET Core não implementar a representação. Aplicativos executados com a identidade do aplicativo para todas as solicitações, usando a identidade de pool ou processo do aplicativo. Se você precisar executar explicitamente uma ação em nome do usuário, use `WindowsIdentity.RunImpersonated` em um [middleware embutido terminal](#) em `Startup.Configure`. Executar uma única ação nesse contexto e, em seguida, feche o contexto.

```
app.Run(async (context) =>
{
    try
    {
        var user = (WindowsIdentity)context.User.Identity;

        await context.Response
            .WriteAsync($"User: {user.Name}\tState: {user.ImpersonationLevel}\n");

        WindowsIdentity.RunImpersonated(user.AccessToken, () =>
        {
            var impersonatedUser = WindowsIdentity.GetCurrent();
            var message =
                $"User: {impersonatedUser.Name}\t" +
                $"State: {impersonatedUser.ImpersonationLevel}";

            var bytes = Encoding.UTF8.GetBytes(message);
            context.Response.Body.Write(bytes, 0, bytes.Length);
        });
    }
    catch (Exception e)
    {
        await context.Response.WriteAsync(e.ToString());
    }
});
```

`RunImpersonated` não oferece suporte a operações assíncronas e não deve ser usado para cenários complexos.

Por exemplo, solicitações de todas ou cadeias de middleware de encapsulamento não é tem suporte ou recomendado.

# Provedores de armazenamento personalizados para ASP.NET Core Identity

26/10/2018 • 20 minutes to read • [Edit Online](#)

Por [Steve Smith](#)

O ASP.NET Core Identity é um sistema extensível que permite que você crie um provedor de armazenamento personalizado e conectá-lo ao seu aplicativo. Este tópico descreve como criar um provedor de armazenamento personalizado para o ASP.NET Core Identity. Ele aborda os conceitos importantes para a criação de seu próprio provedor de armazenamento, mas não é um passo a passo.

[Exibir ou baixar amostra do GitHub.](#)

## Introdução

Por padrão, o sistema de identidade do ASP.NET Core armazena informações de usuário em um banco de dados do SQL Server usando o Entity Framework Core. Para muitos aplicativos, essa abordagem funciona bem. No entanto, você pode preferir usar um mecanismo de persistência diferente ou esquema de dados. Por exemplo:

- Você usa [Azure Table Storage](#) ou outro armazenamento de dados.
- As tabelas de banco de dados têm uma estrutura diferente.
- Talvez você queira usar uma abordagem de acesso de dados diferentes, como [Dapper](#).

Em cada um desses casos, você pode escrever um provedor personalizado para seu mecanismo de armazenamento e conecte esse provedor ao seu aplicativo.

O ASP.NET Core Identity é incluído nos modelos de projeto no Visual Studio com a opção "Contas de usuário individuais".

Ao usar a CLI do .NET Core, adicione `-au Individual`:

```
dotnet new mvc -au Individual  
dotnet new webapi -au Individual
```

## A arquitetura do ASP.NET Core Identity

Identidade do ASP.NET Core consiste em classes chamado gerentes e repositórios. *Gerenciadores* são classes de alto nível que um desenvolvedor de aplicativo usa para executar operações, como a criação de um usuário de identidade. *Repositórios* são classes de nível inferior que especificam como entidades, como usuários e funções, são persistentes. Re却itórios seguem o padrão de repositório e estão intimamente acoplados com o mecanismo de persistência. Os gerentes são separados dos armazenamentos, que significa que você pode substituir o mecanismo de persistência sem alterar o código do aplicativo (com exceção de configuração).

O diagrama a seguir mostra como um aplicativo web interage com os gerentes, enquanto os armazenamentos de interagem com a camada de acesso a dados.



Identity Manager

Example: UserManager, RoleManager

Identity Store

Example: UserStore, RoleStore

Data Access  
Layer

Data Source

Example: SQL Server Database, Azure Table Storage

Para criar um provedor de armazenamento personalizado, crie a fonte de dados, a camada de acesso a dados e as classes de armazenamento que interagem com esta camada de acesso de dados (as caixas verdes e cinza no diagrama acima). Você não precisa personalizar os gerentes ou seu código de aplicativo que interage com eles (as caixas azuis acima).

Ao criar uma nova instância da `UserManager` ou `RoleManager` você fornecer o tipo da classe de usuário e passar uma instância da classe de armazenamento como um argumento. Essa abordagem permite que você conecte suas classes personalizadas no ASP.NET Core.

[Reconfigurar o aplicativo para usar o novo provedor de armazenamento](#) mostra como instanciar `UserManager` e `RoleManager` com um repositório personalizado.

## Tipos de dados de armazenamentos de identidade do ASP.NET Core

[ASP.NET Core Identity](#) tipos de dados são detalhados nas seções a seguir:

### Usuários

Usuários registrados do seu site da web. O `IdentityUser` tipo pode ser estendido ou usado como um exemplo para seu próprio tipo personalizado. Você não precisa herdar de um tipo específico para implementar sua própria solução de armazenamento de identidade personalizada.

### Declarações de usuário

Um conjunto de instruções (ou [declarações](#)) sobre o usuário que representam a identidade do usuário. Pode permitir que uma expressão maior da identidade do usuário que pode ser obtido por meio de funções.

### Logons de usuário

Informações sobre o provedor de autenticação externa (como Facebook ou uma conta da Microsoft) para usar ao fazer logon em um usuário. [Exemplo](#)

## Funções

Grupos de autorização para seu site. Inclui o nome da função Id e a função (como "Admin" ou "Employee").

### Exemplo

## A camada de acesso a dados

Este tópico pressupõe que você esteja familiarizado com o mecanismo de persistência que você pretende usar e como criar entidades para esse mecanismo. Este tópico não fornece detalhes sobre como criar os repositórios ou classes de acesso a dados; Ele fornece algumas sugestões sobre decisões de design ao trabalhar com o ASP.NET Core Identity.

Você tem liberdade ao projetar a camada de acesso a dados para um provedor de repositório personalizado. Você só precisará criar mecanismos de persistência para os recursos que você pretende usar em seu aplicativo. Por exemplo, se você não estiver usando as funções em seu aplicativo, você não precisa criar armazenamento para funções ou associações de função de usuário. Sua tecnologia e a infraestrutura existente podem exigir uma estrutura que é muito diferente da implementação padrão do ASP.NET Core Identity. Na camada de acesso a dados, você deve fornecer a lógica para trabalhar com a estrutura da sua implementação de armazenamento.

A camada de acesso a dados fornece a lógica para salvar os dados de identidade do ASP.NET Core em uma fonte de dados. Camada de acesso a dados para o seu provedor de armazenamento personalizado pode incluir as seguintes classes para armazenar informações de usuário e a função.

### Classe de contexto

Encapsula as informações para conectar-se ao mecanismo de persistência e executar consultas. Várias classes de dados requerem uma instância dessa classe, normalmente é fornecido por meio da injeção de dependência.

### Exemplo

### Armazenamento de usuário

Armazena e recupera informações de usuário (por exemplo, o hash de nome e a senha do usuário). [Exemplo](#)

### Armazenamento de função

Armazena e recupera informações de função (como o nome da função). [Exemplo](#)

### Armazenamento de UserClaims

Armazena e recupera informações de declaração de usuário (como o tipo de declaração e o valor). [Exemplo](#)

### Armazenamento de UserLogins

Armazena e recupera informações de logon do usuário (como um provedor de autenticação externa). [Exemplo](#)

### Armazenamento de UserRole

Armazena e recupera a quais funções são atribuídas a quais usuários. [Exemplo](#)

**Dica:** implementar apenas as classes que você pretende usar em seu aplicativo.

Em classes de acesso a dados, forneça o código para executar operações de dados para o mecanismo de persistência. Por exemplo, dentro de um provedor personalizado, você pode ter o código a seguir para criar um novo usuário na `armazenar` classe:

```
public async Task<IdentityResult> CreateAsync(ApplicationUser user,
    CancellationToken cancellationToken = default(CancellationToken))
{
    cancellationToken.ThrowIfCancellationRequested();
    if (user == null) throw new ArgumentNullException(nameof(user));

    return await _usersTable.CreateAsync(user);
}
```

A lógica de implementação para criar o usuário está no `_usersTable.CreateAsync` método, mostrado abaixo.

## Personalizar a classe de usuário

Ao implementar um provedor de armazenamento, crie uma classe de usuário que é equivalente à classe `IdentityUser`.

No mínimo, sua classe de usuário deve incluir um `Id` e um `UserName` propriedade.

O `IdentityUser` classe define as propriedades que o `UserManager` chamadas ao executar operações de solicitadas. O tipo de padrão de `Id` propriedade é uma cadeia de caracteres, mas você pode herdar de `IdentityUser< TKey, TUserClaim, TUserRole, TUserLogin, TUserToken >` e especifique um tipo diferente. O framework espera a implementação de armazenamento para lidar com conversões de tipo de dados.

## Personalizar o repositório do usuário

Criar um `UserStore` classe que fornece os métodos para todas as operações de dados no usuário. Essa classe é equivalente a `UserStore< TUser >` classe. No seu `UserStore` classe, implemente `IUserStore< TUser >` e as interfaces opcionais necessárias. Você selecione quais interfaces opcionais para implementar com base na funcionalidade fornecida no seu aplicativo.

### Interfaces opcionais

- `IUserRoleStore`
- `IUserClaimStore`
- `IUserPasswordStore`
- `IUserSecurityStampStore`
- `IUserEmailStore`
- `IPhoneNumberStore`
- `IQueryableUserStore`
- `IUserLoginStore`
- `IUserTwoFactorStore`
- `IUserLockoutStore`

As interfaces opcionais herdam `IUserStore< TUser >`. Você pode ver que um usuário de exemplo parcialmente implementada armazenar na [aplicativo de exemplo](#).

Dentro de `UserStore` classe, que você usar as classes de acesso de dados que você criou para executar operações. Eles são passados usando a injeção de dependência. Por exemplo, no SQL Server com a implementação do Dapper, o `UserStore` classe tem o `CreateAsync` método que usa uma instância de `DapperUsersTable` para inserir um novo registro:

```
public async Task<IdentityResult> CreateAsync(ApplicationUser user)
{
    string sql = "INSERT INTO dbo.CustomUser " +
        "VALUES (@id, @Email, @EmailConfirmed, @PasswordHash, @UserName)";

    int rows = await _connection.ExecuteAsync(sql, new { user.Id, user.Email, user.EmailConfirmed,
        user.PasswordHash, user.UserName });

    if(rows > 0)
    {
        return IdentityResult.Success;
    }
    return IdentityResult.Failed(new IdentityError { Description = $"Could not insert user {user.Email}." });
}
```

## Interfaces a serem implementadas durante a personalização armazenamento de usuário

- **IUserStore**

O [IUserStore<TUser>](#) interface é a única interface, você deve implementar no repositório do usuário. Define métodos para criar, atualizar, excluir e recuperar os usuários.

- **IUserClaimStore**

O [IUserClaimStore<TUser>](#) interface define os métodos a implementar para habilitar declarações do usuário. Ela contém métodos para adicionar, remover e recuperando as declarações de usuário.

- **IUserLoginStore**

O [IUserLoginStore<TUser>](#) define os métodos a implementar para habilitar provedores de autenticação externa. Ela contém métodos para adicionar, remover e recuperar os logons de usuário e um método para recuperar um usuário com base nas informações de logon.

- **IUserRoleStore**

O [IUserRoleStore<TUser>](#) interface define os métodos a implementar para mapear um usuário a uma função. Ela contém métodos para adicionar, remover e recuperar funções de usuário e um método para verificar se um usuário está atribuído a uma função.

- **IUserPasswordStore**

O [IUserPasswordStore<TUser>](#) interface define os métodos a implementar para persistir as senhas hash. Ela contém métodos para obter e definir a senha de hash e um método que indica se o usuário tiver definido uma senha.

- **IUserSecurityStampStore**

O [IUserSecurityStampStore<TUser>](#) interface define os métodos a implementar para usar um carimbo de segurança para que indica se as informações da conta do usuário foi alterado. Esse carimbo é atualizado quando um usuário altera a senha ou adiciona ou remove logons. Ela contém métodos para obter e definir o carimbo de segurança.

- **IUserTwoFactorStore**

O [IUserTwoFactorStore<TUser>](#) interface define os métodos a implementar para dar suporte à autenticação de dois fatores. Ela contém métodos para obter e definir se a autenticação de dois fatores é ativada para um usuário.

- **IUserPhoneNumberStore**

O [IUserPhoneNumberStore<TUser>](#) interface define os métodos a implementar para armazenar números de telefone do usuário. Ela contém métodos para obter e definir o número de telefone e se o número de telefone foi confirmado.

- **IUserEmailStore**

O [IUserEmailStore<TUser>](#) interface define os métodos a implementar para armazenar endereços de email do usuário. Ela contém métodos para obter e definir o endereço de email e se o email for confirmado.

- **IUserLockoutStore**

O [IUserLockoutStore<TUser>](#) interface define os métodos a implementar para armazenar informações sobre como bloquear uma conta. Ela contém métodos para controlar as tentativas de acesso com falha e bloqueios.

- **IQueryableUserStore**

O [IQueryableUserStore<TUser>](#) interface define os membros que você pode implementar para fornecer um repositório de usuários que podem ser consultados.

Você pode implementar apenas as interfaces que são necessários em seu aplicativo. Por exemplo:

```

public class UserStore : IUserStore<IdentityUser>,
    IUserClaimStore<IdentityUser>,
    IUserLoginStore<IdentityUser>,
    IUserRoleStore<IdentityUser>,
    IUserIdentityStore<IdentityUser>,
    IUserSecurityStampStore<IdentityUser>
{
    // interface implementations not shown
}

```

## IdentityUserClaim, IdentityUserLogin e IdentityUserRole

O `Microsoft.AspNet.Identity.EntityFramework` namespace contém implementações do `IdentityUserClaim`, `IdentityUserLogin`, e `IdentityUserRole` classes. Se você estiver usando esses recursos, você talvez queira criar suas próprias versões dessas classes e definir as propriedades para seu aplicativo. No entanto, às vezes, é mais eficiente para não carregar essas entidades na memória ao executar operações básicas (como adicionar ou remover a declaração do usuário). Em vez disso, as classes de armazenamento de back-end podem executar essas operações diretamente na fonte de dados. Por exemplo, o `UserStore.GetClaimsAsync` método pode chamar o `userClaimTable.FindByUserId(user.Id)` método para executar uma consulta em que diretamente da tabela e retorna uma lista de declarações.

## Personalizar a classe de função

Ao implementar um provedor de armazenamento de função, você pode criar um tipo de função personalizada. Ele não precisa implementar uma interface específica, mas ele deve ter uma `Id` e, normalmente terá uma `Name` propriedade.

A seguir está um exemplo de classe de função:

```

using System;

namespace CustomIdentityProviderSample.CustomProvider
{
    public class ApplicationRole
    {
        public Guid Id { get; set; } = Guid.NewGuid();
        public string Name { get; set; }
    }
}

```

## Personalizar o repositório da função

Você pode criar um `RoleStore` classe que fornece os métodos para todas as operações de dados em funções. Essa classe é equivalente a `alteração do RoleStore<TRole>` classe. No `RoleStore` classe, você implementa o `IRoleStore<TRole>` e, opcionalmente, o `IQueryableRoleStore<TRole>` interface.

- **`IRoleStore<TRole>`**

O `IRoleStore<TRole>` interface define os métodos a implementar na classe de repositório de função. Ela contém métodos para criar, atualizar, excluir e recuperar funções.

- **`RoleStore<TRole>`**

Para personalizar `RoleStore`, crie uma classe que implementa o `IRoleStore<TRole>` interface.

## Reconfigurar o aplicativo para usar um novo provedor de armazenamento

Depois de implementar um provedor de armazenamento, você pode configurar seu aplicativo para usá-lo. Se seu

aplicativo usado o provedor padrão, substitua-o com o provedor personalizado.

1. Remover o `Microsoft.AspNetCore.Identity` pacote do NuGet.
2. Se o provedor de armazenamento reside em um projeto separado ou um pacote, adicione uma referência a ele.
3. Substitua todas as referências a `Microsoft.AspNetCore.Identity` com o uso de uma instrução para o namespace do seu provedor de armazenamento.
4. No `ConfigureServices` método, altere o `AddIdentity` método usar seus tipos personalizados. Você pode criar seus próprios métodos de extensão para essa finalidade. Ver [IdentityServiceCollectionExtensions](#) para obter um exemplo.
5. Se você estiver usando as funções, atualize o `RoleManager` para usar seu `RoleStore` classe.
6. Atualize a cadeia de caracteres de conexão e as credenciais para a configuração de seu aplicativo.

Exemplo:

```
public void ConfigureServices(IServiceCollection services)
{
    // Add identity types
    services.AddIdentity<ApplicationUser, ApplicationRole>()
        .AddDefaultTokenProviders();

    // Identity Services
    services.AddTransient<IUserStore<ApplicationUser>, CustomUserStore>();
    services.AddTransient<IRoleStore<ApplicationRole>, CustomRoleStore>();
    string connectionString = Configuration.GetConnectionString("DefaultConnection");
    services.AddTransient<SqlConnection>(e => new SqlConnection(connectionString));
    services.AddTransient<DapperUsersTable>();

    // additional configuration
}
```

## Referências

- [Provedores de armazenamento personalizados para a identidade do ASP.NET 4.x](#)
- [ASP.NET Core Identity](#) – neste repositório inclui links para a comunidade mantida provedores de armazenamento.

# Autenticação de Facebook, Google e de provedor externo no ASP.NET Core

21/01/2019 • 6 minutes to read • [Edit Online](#)

Por [Valeriy Novytskyy](#) e [Rick Anderson](#)

Este tutorial demonstra como criar um aplicativo ASP.NET Core 2.2, que permite aos usuários fazer logon usando o OAuth 2.0 com as credenciais de provedores de autenticação externa.

Os provedores [Facebook](#), [Twitter](#), [Google](#) e [Microsoft](#) são abordados nas seções a seguir. Outros provedores estão disponíveis em pacotes de terceiros, como [AspNet.Security.OAuth.Providers](#) e [AspNet.Security.OpenId.Providers](#).



Permitir que os usuários entrem com suas credenciais existentes é conveniente para os usuários e transfere muitas das complexidades de gerenciar o processo de entrada para um terceiro. Para obter exemplos de como os logons sociais podem impulsionar o tráfego e as conversões de clientes, consulte os estudos de caso do [Facebook](#) e do [Twitter](#).

## Criar um novo projeto ASP.NET Core

- No Visual Studio 2017, crie um projeto na Página Inicial ou por meio de **Arquivo > Novo > Projeto**.
- Selecione o modelo **Aplicativo Web ASP.NET Core**, disponível na categoria **Visual C# > .NET Core**:
- Selecione **Alterar Autenticação** e defina a autenticação para **Contas de Usuário Individuais**.

## Aplicar migrações

- Execute o aplicativo e selecione o link **Registrar**.
- Insira o email e a senha para a nova conta e, em seguida, selecione **Registrar**.
- Siga as instruções para aplicar as migrações.

## Solicitar informações com um proxy para frente ou balanceador de carga

Se o aplicativo for implantado atrás de um servidor proxy ou um balanceador de carga, algumas das informações de solicitação original podem ser encaminhadas para o aplicativo nos cabeçalhos de solicitação. Essas informações geralmente incluem o esquema de solicitação de seguro (`https`), host e endereço IP do cliente. Aplicativos não lidas automaticamente esses cabeçalhos de solicitação para descobrir e usar as informações da solicitação original.

O esquema é usado na geração de link que afeta o fluxo de autenticação com provedores externos. Perder o esquema de seguro (`https`) resulta no aplicativo de geração de URLs de redirecionamento inseguro incorreto.

Use o Middleware de cabeçalhos encaminhados para disponibilizar as informações da solicitação original para o aplicativo para o processamento da solicitação.

Para obter mais informações, consulte [Configure o ASP.NET Core para trabalhar com servidores proxy](#)

balanceadores de carga.

## Usar o SecretManager para armazenar os tokens atribuídos por provedores de logon

Os provedores de logon social atribuem tokens de **ID do Aplicativo** e **Segredo do Aplicativo** durante o processo de registro. Os nomes de token exatos variam de acordo com o provedor. Esses tokens representam as credenciais que seu aplicativo usa para acessar a API. Os tokens constituem os "segredos" que podem ser vinculados à configuração de aplicativo com a ajuda do [Secret Manager](#). O Secret Manager é uma alternativa mais segura para armazenar os tokens em um arquivo de configuração, como `appsettings.json`.

### IMPORTANT

O Secret Manager destina-se apenas para fins de desenvolvimento. Você pode armazenar e proteger os segredos de teste e produção do Azure com o [provedor de configuração do Azure Key Vault](#).

Siga as etapas no tópico [Armazenamento seguro dos segredos do aplicativo em desenvolvimento no ASP.NET Core](#) para armazenar os tokens atribuídos por cada provedor de logon abaixo.

## Configurar os provedores de logon necessários para o aplicativo

Use os seguintes tópicos para configurar seu aplicativo para usar os respectivos provedores:

- Instruções do [Facebook](#)
- Instruções do [Twitter](#)
- Instruções do [Google](#)
- Instruções da [Microsoft](#)
- [Outras](#) instruções do provedor

## Vários provedores de autenticação

Caso o aplicativo exija vários provedores, encadeie os métodos de extensão do provedor em `AddAuthentication`:

```
services.AddAuthentication()
    .AddMicrosoftAccount(microsoftOptions => { ... })
    .AddGoogle(googleOptions => { ... })
    .AddTwitter(twitterOptions => { ... })
    .AddFacebook(facebookOptions => { ... });
```

## Definir a senha opcionalmente

Ao registrar um provedor de logon externo, você não precisa ter uma senha registrada no aplicativo. Isso o alivia da tarefa de criar e lembrar de uma senha para o site, mas também o torna dependente do provedor de logon externo. Se o provedor de logon externo não estiver disponível, você não poderá fazer logon no site.

Para criar uma senha e entrar usando seu email definido durante o processo de entrada com provedores externos:

- Selecione o link **Olá,< >alias de email** na parte superior direita para navegar até a exibição **Gerenciar**.

## Manage your account.

Change your account settings

Password:	<a href="#">[ Create ]</a>
External Logins:	1 <a href="#">[Manage]</a>
Phone Number:	Phone Numbers can be used as a second factor of verification in two-factor authentication. See <a href="#">this article</a> for details on setting up this ASP.NET application to support two-factor authentication using SMS.
Two-Factor Authentic...	There are no two-factor authentication providers configured. See <a href="#">this article</a> for setting up this application to support two-factor authentication.

- Selecione **Criar**

You do not have a local username/password for this site. Add a local account so you can log in without an external login.

### Set your password

New password

••••••

Confirm new password

••••••

[Set password](#)

© 2015 - WebApplication224

- Defina uma senha válida e use-a para entrar com seu email.

## Próximas etapas

- Este artigo apresentou a autenticação externa e explicou os pré-requisitos necessários para adicionar logons externos ao aplicativo ASP.NET Core.
- Páginas de referência específicas ao provedor para configurar logons para os provedores necessários para o aplicativo.
- Você talvez queira manter os dados adicionais sobre o usuário e seus tokens de atualização e acesso. Para obter mais informações, consulte [Manter declarações adicionais e os tokens de provedores externos no ASP.NET Core](#).

# Configuração de logon externo do Google no ASP.NET Core

18/01/2019 • 7 minutes to read • [Edit Online](#)

Por [Valeriy Novytskyy](#) e [Rick Anderson](#)

Em janeiro de 2019 Google começou a [desligar](#) Google + entrar e os desenvolvedores devem mover para um novo logon do Google no sistema, março. O ASP.NET Core 2.1 e 2.2 pacotes para a autenticação do Google serão atualizados em fevereiro para acomodar as alterações. Para obter mais informações e atenuações temporárias para o ASP.NET Core, consulte [esse problema de GitHub](#). Este tutorial foi atualizado com o novo processo de instalação.

Este tutorial mostra como habilitar usuários entrar com sua conta do Google usando um projeto ASP.NET Core 2.2 criado na [página anterior](#).

## Criar uma ID de cliente e o projeto de Console de API do Google

- Navegue até a [integração do Google Sign-In em seu aplicativo web](#) e selecione **configurar um projeto**.
- No **configurar o cliente do OAuth** caixa de diálogo, selecione **servidor Web**.
- No **URIs de redirecionamento autorizados** caixa de entrada de texto, defina o URI de redirecionamento. Por exemplo, `https://localhost:5001/signin-google`
- Salvar a **ID do cliente e segredo do cliente**.
- Ao implantar o site, registrar a nova url pública do **Console do Google**.

## Store Google ClientID e ClientSecret

Store as configurações confidenciais, como o Google `client ID` e `Client Secret` com o [Secret Manager](#). Para os fins deste tutorial, nomeie os tokens `Authentication:Google:ClientId` e `Authentication:Google:ClientSecret`:

```
dotnet user-secrets set "Authentication:Google:ClientId" "X.apps.googleusercontent.com"
dotnet user-secrets set "Authentication:Google:ClientSecret" "<client secret>"
```

Você pode gerenciar suas credenciais de API e o uso na [Console de API](#).

## Configurar a autenticação do Google

Adicionar o serviço do Google para `Startup.ConfigureServices`.

A chamada para [AddIdentity](#) define as configurações de esquema padrão. O [AddAuthentication\(String\)](#) conjuntos de sobrecarregar os [DefaultScheme](#) propriedade. O [AddAuthentication\(ação<AuthenticationOptions>\)](#) sobrecarga permite configurar opções de autenticação, que podem ser usadas para definir os esquemas de autenticação padrão para finalidades diferentes. As chamadas subsequentes para [AddAuthentication](#) substituição configurada anteriormente [AuthenticationOptions](#) propriedades.

[AuthenticationBuilder](#) métodos de extensão que registra um manipulador de autenticação podem ser chamados apenas uma vez por esquema de autenticação. Há sobrecargas que permitem configurar as propriedades de esquema, o nome de esquema e nome de exibição.

## Entrar com o Google

- Execute o aplicativo e clique em **faça logon no**. Aparece uma opção para entrar com o Google.
- Clique o **Google** botão, que redireciona para o Google para autenticação.
- Depois de inserir suas credenciais do Google, você será redirecionado para o site da web.

## Solicitar informações com um proxy para frente ou balanceador de carga

Se o aplicativo for implantado atrás de um servidor proxy ou um balanceador de carga, algumas das informações de solicitação original podem ser encaminhadas para o aplicativo nos cabeçalhos de solicitação. Essas informações geralmente incluem o esquema de solicitação de seguro (`https`), host e endereço IP do cliente. Aplicativos não lidas automaticamente esses cabeçalhos de solicitação para descobrir e usar as informações da solicitação original.

O esquema é usado na geração de link que afeta o fluxo de autenticação com provedores externos. Perder o esquema de seguro (`https`) resulta no aplicativo de geração de URLs de redirecionamento inseguro incorreto.

Use o Middleware de cabeçalhos encaminhados para disponibilizar as informações da solicitação original para o aplicativo para o processamento da solicitação.

Para obter mais informações, consulte [Configure o ASP.NET Core para trabalhar com servidores proxy e平衡adores de carga](#).

## Vários provedores de autenticação

Caso o aplicativo exija vários provedores, encadeie os métodos de extensão do provedor em [AddAuthentication](#):

```
services.AddAuthentication()
    .AddMicrosoftAccount(microsoftOptions => { ... })
    .AddGoogle(googleOptions => { ... })
    .AddTwitter(twitterOptions => { ... })
    .AddFacebook(facebookOptions => { ... });
```

Consulte a [GoogleOptions](#) referência da API para obter mais informações sobre opções de configuração com suporte pela autenticação do Google. Isso pode ser usado para solicitar informações diferentes sobre o usuário.

## Alterar o retorno de chamada padrão URI

O segmento URI `/signin-google` é definido como o retorno de chamada padrão do provedor de autenticação do Google. Você pode alterar o retorno de chamada padrão URI ao configurar o middleware de autenticação do Google via o herdadas [RemoteAuthenticationOptions.CallbackPath](#) propriedade da [GoogleOptions](#) classe.

## Solução de problemas

- Se a entrada não funciona e você não estiver recebendo erros, alterne para modo de desenvolvimento para que o problema seja mais fácil de depurar.
- Se a identidade não estiver configurada, chamando `services.AddIdentity` na `ConfigureServices`, tentar autenticar resulta em *ArgumentException: A opção 'SignInScheme' deve ser fornecida*. O modelo de projeto usado neste tutorial garante que isso é feito.
- Se o banco de dados do site não foi criado por meio da aplicação a migração inicial, você obterá *uma operação de banco de dados falhou ao processar a solicitação* erro. Toque **aplicar migrações** para criar o banco de dados e atualizar para continuar após o erro.

## Próximas etapas

- Este artigo mostrou como você pode autenticar com o Google. Você pode seguir uma abordagem semelhante para autenticar com outros provedores listados na [página anterior](#).
- Depois que você publica o aplicativo no Azure, redefinir o `ClientSecret` no Console de API do Google.
- Defina as `Authentication:Google:ClientId` e `Authentication:Google:ClientSecret` como configurações de aplicativo no portal do Azure. Configurar o sistema de configuração para ler as chaves de variáveis de ambiente.

# Configuração de logon externo do Facebook no ASP.NET Core

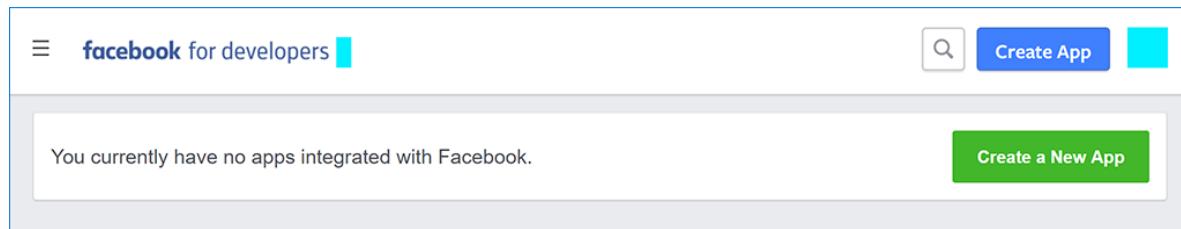
27/12/2018 • 9 minutes to read • [Edit Online](#)

Por [Valeriy Novytskyy](#) e [Rick Anderson](#)

Este tutorial mostra como permitir que os usuários entrem com a conta do Facebook deles usando um projeto de amostra do ASP.NET Core 2.0 criado na [página anterior](#). Vamos começar criando um Facebook App ID seguindo as [etapas oficiais](#).

## Criar o aplicativo no Facebook

- Navegue até a página [aplicativos no site para desenvolvedores do Facebook](#). Se você ainda não tiver uma conta do Facebook, use o link **Sign up** para se registrar no Facebook na página de logon para criar uma.
- Clique no botão **adicionar um novo aplicativo** no canto superior direito para criar uma nova ID de aplicativo.



- Preencha o formulário e clique no botão **criar ID do aplicativo**.

A screenshot of the 'Create a New App ID' form. The title is 'Create a New App ID'. Below it, a sub-instruction says 'Get started integrating Facebook into your app or website'. There are two input fields: 'Display Name' (with placeholder 'The name you want to associate with this App ID') and 'Contact Email' (with placeholder 'Used for important communication about your app'). At the bottom, there's a link 'By proceeding, you agree to the Facebook Platform Policies' and a row of buttons: 'Cancel' (gray), 'Create App ID' (blue), and another 'Create App ID' button (green).

- Na página **selecionar um produto**, clique em **Set Up** no painel **Facebook Login**.

The screenshot shows the SocialLogins dashboard. At the top, there's a navigation bar with the app name, APP ID, View Analytics, Tools & Support, and Docs. On the left, a sidebar titled 'PRODUCTS' includes links for Dashboard, Settings, Roles, Alerts, App Review, and '+ Add Product'. The '+ Add Product' link is highlighted. The main area is titled 'Select a product' and shows two options: 'Account Kit' and 'Facebook Login'. Each option has a circular icon (User plus sign for Account Kit, lock for Facebook Login), a title, a short description, and two buttons at the bottom: 'Read Docs' and 'Set Up'.

- O assistente **Quickstart** iniciará com **escolher uma plataforma** como a primeira página. Ignore o assistente clicando no link **configurações** no menu à esquerda:

This screenshot shows the 'Facebook Login' settings page. The left sidebar, titled 'PRODUCTS', lists 'Facebook Login' which is currently selected. Below it are 'Settings' (highlighted in blue) and 'Quickstart'. At the bottom of the sidebar are '+ Add Product' and other navigation links.

- Você verá a **configurações do cliente OAuth** página:

## Client OAuth Settings

Yes

### Client OAuth Login

Enables the standard OAuth client token flow. Secure your application and prevent abuse by locking down which token redirect URIs are allowed with the options below. Disable globally if not used. [?]

Yes

No

### Web OAuth Login

Enables web based OAuth client login for building custom login flows. [?]

### Force Web OAuth Reauthentication

When on, prompts people to enter their Facebook password in order to log in on the web. [?]

No

### Embedded Browser OAuth Login

Enables browser control redirect uri for OAuth client login. [?]

### Valid OAuth redirect URIs

`https://localhost:44320/signin-facebook`

No

### Login from Devices

Enables the OAuth client login flow for devices like a smart TV [?]

[Deauthorize](#)

[Discard](#)

[Save Changes](#)

- Insira o URI de desenvolvimento com `/signin-facebook` acrescentado para o **URIs de redirecionamento de OAuth válido** campo (por exemplo: `https://localhost:44320/signin-facebook`). A autenticação do Facebook configurada mais tarde neste tutorial automaticamente manipulará as solicitações no `/signin-facebook` rota para implementar o fluxo de OAuth.

#### NOTE

O URI `/signin-facebook` é definido como o retorno de chamada padrão do provedor de autenticação do Facebook. Você pode alterar o retorno de chamada padrão URI ao configurar o middleware de autenticação do Facebook por meio de herdadas `RemoteAuthenticationOptions.CallbackPath` propriedade da `FacebookOptions` classe.

- Clique em **salvar alterações**.
- Clique em **as configurações > básica** link no painel de navegação à esquerda.

Nessa página, anote seu `App ID` e seu `App Secret`. Você adicionará os dois em seu aplicativo ASP.NET Core na próxima seção:

- Ao implantar o site, você precisará voltar para a página de configurações **Logon do Facebook** e registrar um novo URI público.

## ID do Facebook App Store e o segredo do aplicativo

Vincular as configurações confidenciais, como o Facebook `App ID` e `App Secret` para sua configuração de

aplicativo usando o [Secret Manager](#). Para os fins deste tutorial, nomeie os tokens `Authentication:Facebook:AppId` e `Authentication:Facebook:AppSecret`.

Execute os seguintes comandos para armazenar com segurança `App ID` e `App Secret` usando o Secret Manager:

```
dotnet user-secrets set Authentication:Facebook:AppId <app-id>
dotnet user-secrets set Authentication:Facebook:AppSecret <app-secret>
```

## Configurar a autenticação do Facebook

Adicione o serviço do Facebook no método `ConfigureServices` do arquivo `Startup.cs`:

```
services.AddDefaultIdentity<IdentityUser>()
    .AddDefaultUI(UIFramework.Bootstrap4)
    .AddEntityFrameworkStores<ApplicationDbContext>();

services.AddAuthentication().AddFacebook(facebookOptions =>
{
    facebookOptions.AppId = Configuration["Authentication:Facebook:AppId"];
    facebookOptions.AppSecret = Configuration["Authentication:Facebook:AppSecret"];
});
```

A chamada para `AddIdentity` define as configurações de esquema padrão. O `AddAuthentication(String)` conjuntos de sobrecarregar os `DefaultScheme` propriedade. O `AddAuthentication(ação<AuthenticationOptions>)` sobrecarga permite configurar opções de autenticação, que podem ser usadas para definir os esquemas de autenticação padrão para finalidades diferentes. As chamadas subsequentes para `AddAuthentication` substituição configurada anteriormente `AuthenticationOptions` propriedades.

`AuthenticationBuilder` métodos de extensão que registra um manipulador de autenticação podem ser chamados apenas uma vez por esquema de autenticação. Há sobrecargas que permitem configurar as propriedades de esquema, o nome de esquema e nome de exibição.

## Vários provedores de autenticação

Caso o aplicativo exija vários provedores, encadeie os métodos de extensão do provedor em `AddAuthentication`:

```
services.AddAuthentication()
    .AddMicrosoftAccount(microsoftOptions => { ... })
    .AddGoogle(googleOptions => { ... })
    .AddTwitter(twitterOptions => { ... })
    .AddFacebook(facebookOptions => { ... });
```

Instalar o pacote [Microsoft.AspNetCore.Authentication.Facebook](#).

- Para instalar este pacote com o Visual Studio 2017, clique com botão direito no projeto e selecione **gerenciar pacotes NuGet**.
- Para instalar o .NET Core CLI, execute o seguinte comando no diretório do projeto:

```
dotnet add package Microsoft.AspNetCore.Authentication.Facebook
```

Adicione o middleware do Facebook no método `Configure` do arquivo `Startup.cs`:

```
app.UseFacebookAuthentication(new FacebookOptions()
{
    AppId = Configuration["Authentication:Facebook:AppId"],
    AppSecret = Configuration["Authentication:Facebook:AppSecret"]
});
```

Consulte as referências de API do [FacebookOptions](#) para obter mais informações sobre as opções de configuração compatíveis com a autenticação do Facebook. As opções de configuração podem ser usadas para:

- Solicitar informações diferentes sobre o usuário.
- Adicionar argumentos de cadeia de caracteres de consulta para personalizar a experiência de logon.

## Entrar com o Facebook

Executar o aplicativo e clique em **faça logon no**. Você verá uma opção para entrar com o Facebook.

The screenshot shows a login page for 'WebApplication3'. At the top, there's a navigation bar with links for 'Home', 'About', 'Contact', 'Register', and 'Log in'. Below the navigation bar, the main content area has a heading 'Log in.' and two sections: 'Use a local account to log in.' and 'Use another service to log in.'. The 'Local Account' section contains fields for 'Email' and 'Password', a 'Remember me?' checkbox, and a 'Log in' button. The 'Social Logins' section contains a 'Facebook' button. At the bottom of the page, there are links for 'Register as a new user?' and 'Forgot your password?'. A copyright notice at the very bottom reads '© 2016 - WebApplication3'.

Quando você clica em **Facebook**, você será redirecionado ao Facebook para autenticação:

# facebook

[Sign Up](#)

## Log into Facebook

---

 or 

---

[Create New Account](#)[Forgot account?](#)[Not now](#)

Autenticação do Facebook solicita o endereço de email e o perfil público por padrão:



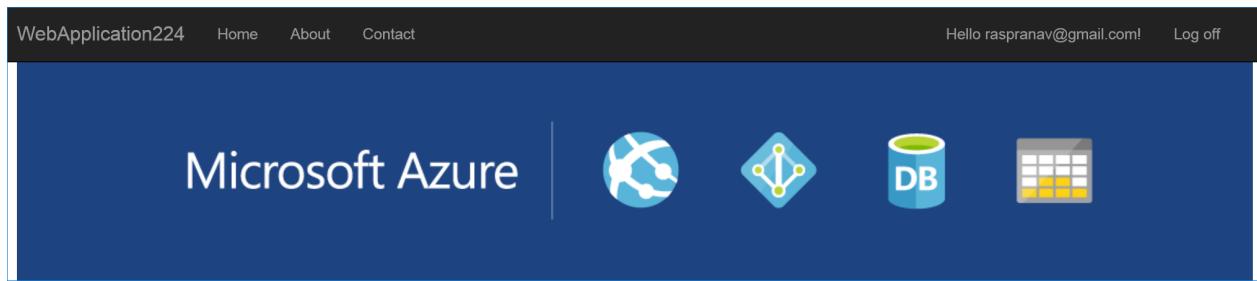
**SocialLogins** will receive:  
your public profile and email address. ⓘ

 [Edit This](#)[Cancel](#)

This doesn't let the app post to Facebook

Depois que você insira suas credenciais do Facebook, você será redirecionado para seu site em que você pode definir seu email.

Agora você está conectado usando suas credenciais do Facebook:



## Solicitar informações com um proxy para frente ou平衡ador de carga

Se o aplicativo for implantado atrás de um servidor proxy ou um balanceador de carga, algumas das informações de solicitação original podem ser encaminhadas para o aplicativo nos cabeçalhos de solicitação. Essas informações geralmente incluem o esquema de solicitação de seguro (`https`), host e endereço IP do cliente. Aplicativos não lidas automaticamente esses cabeçalhos de solicitação para descobrir e usar as informações da solicitação original.

O esquema é usado na geração de link que afeta o fluxo de autenticação com provedores externos. Perder o esquema de seguro (`https`) resulta no aplicativo de geração de URLs de redirecionamento inseguro incorreto.

Use o Middleware de cabeçalhos encaminhados para disponibilizar as informações da solicitação original para o aplicativo para o processamento da solicitação.

Para obter mais informações, consulte [Configure o ASP.NET Core para trabalhar com servidores proxy e平衡adores de carga](#).

## Solução de problemas

- **ASP.NET Core 2.x somente:** Se a identidade não está configurada por meio da chamada `services.AddIdentity` na `ConfigureServices`, a tentativa de autenticar resultará em `ArgumentException`: A opção 'SignInScheme' deve ser fornecida. O modelo de projeto usado neste tutorial garante que isso é feito.
- Se o banco de dados do site não foi criado por meio da aplicação a migração inicial, você obterá uma operação de banco de dados falhou ao processar a solicitação erro. Toque **aplicar migrações** para criar o banco de dados e atualizar para continuar após o erro.

## Próximas etapas

- Adicione a [Microsoft.AspNetCore.Authentication.Facebook](#) pacote NuGet ao seu projeto para cenários avançados de autenticação de Facebook. Este pacote não é necessário para integrar funcionalidade de logon externo do Facebook com seu aplicativo.
- Este artigo mostrou como você pode autenticar com o Facebook. Você pode seguir uma abordagem semelhante para autenticar com outros provedores listados na [página anterior](#).
- Depois de publicar seu site da web para aplicativo web do Azure, você deve redefinir o `AppSecret` no portal do desenvolvedor do Facebook.
- Defina as `Authentication:Facebook:AppId` e `Authentication:Facebook:AppSecret` como configurações de aplicativo no portal do Azure. Configurar o sistema de configuração para ler as chaves de variáveis de ambiente.

# Configuração de logon externo Account da Microsoft com o ASP.NET Core

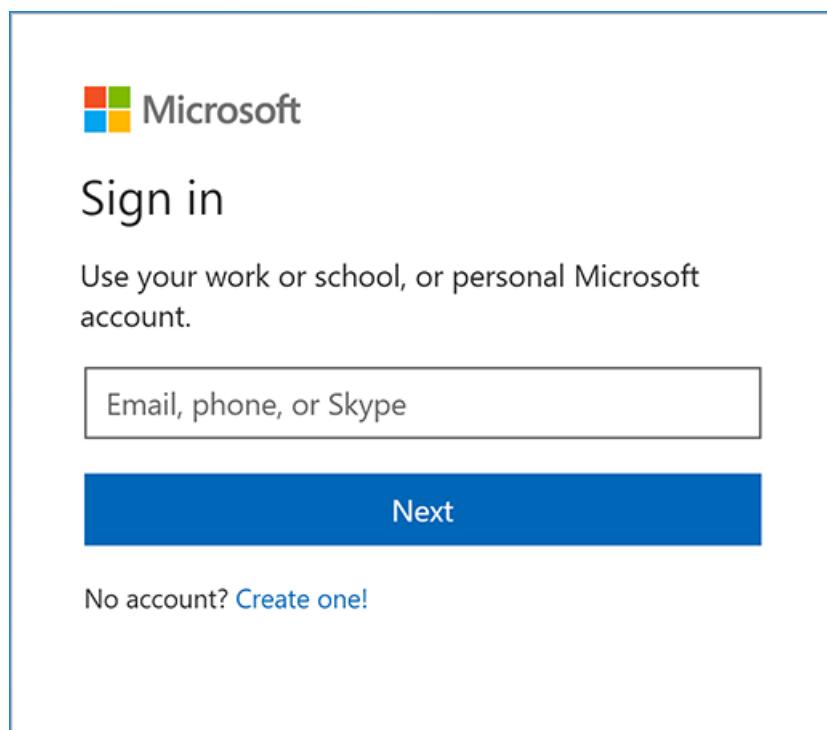
27/12/2018 • 10 minutes to read • [Edit Online](#)

Por [Valeriy Novytskyy](#) e [Rick Anderson](#)

Este tutorial mostra como permitir que seus usuários entrar com sua conta da Microsoft usando um projeto do ASP.NET Core 2.0 de exemplo criado na [página anterior](#).

## Criar o aplicativo no Portal do desenvolvedor da Microsoft

- Navegue até <https://apps.dev.microsoft.com> e criar ou entrar em uma conta da Microsoft:



Se você ainda não tiver uma conta da Microsoft, toque em **crie uma!** Depois de entrar, você será redirecionado para **meus aplicativos** página:

A screenshot of the Application Registration Portal. The top navigation bar includes the Microsoft logo, the text "Application Registration Portal", "Docs", "Feedback", and a user profile icon. The main area is titled "My applications" with a "Learn More" link and a "Add an app" button. Below this, there is a table with two columns: "Name" and "App ID / Client Id". One row shows "TestApp" and "b7e887c6-23e7-48fa-802a-c528e47cf02a", with a "Delete" button next to it.

- Toque **adicionar um aplicativo** no canto superior direito de canto e insira seu **nome do aplicativo** e

Contact Email:

The screenshot shows the Microsoft Application Registration Portal. At the top, there's a dark header bar with the Microsoft logo, the text "Application Registration Portal", "Docs", "Feedback", and a user icon. Below the header, the main title "Register your application" is displayed in a large, light gray font. Underneath it, there are two input fields: one for "Application Name" containing "SocialLogins" and another for "Contact Email" containing "aspdemo@outlook.com". There's also a section for "Guided Setup" with a checkbox labeled "Let us help you get started". A horizontal line separates this from the "By proceeding, you agree to the [Microsoft Platform Policies](#)" text. At the bottom left is a blue "Create" button.

Application Name

SocialLogins

Contact Email

Used for important communications about your application

aspdemo@outlook.com

Guided Setup

Let us help you get started

By proceeding, you agree to the [Microsoft Platform Policies](#)

Create

- Para os fins deste tutorial, desmarque a **instalação guiada** caixa de seleção.
- Toque **Create** para continuar para o **registro** página. Fornecer um **nome** e observe o valor da **Id do aplicativo**, que você usar como `ClientId` posteriormente no tutorial:

[My applications](#) / SocialLogins

# SocialLogins Registration

[Click here for help integrating your application with Microsoft.](#)

## Properties

Name

SocialLogins

Application Id

4d519acd-6a8c-4df4-a246-fe5ea741c9db

## Application Secrets

[Generate New Password](#)[Generate New Key Pair](#)[Upload Public Key](#)

## Platforms

[Add Platform](#)

- Toque **Adicionar plataforma** na **plataformas** seção e selecione o **Web** plataforma:

### Add Platform

[Cancel](#)

- No novo **Web** plataforma, digite sua URL de desenvolvimento com `/signin-microsoft` acrescentado para o **URLs de redirecionamento** campo (por exemplo: `https://localhost:44320/signin-microsoft`). O esquema de autenticação da Microsoft configurado mais tarde neste tutorial automaticamente manipulará as solicitações em `/signin-microsoft` rota para implementar o fluxo de OAuth:

The screenshot shows the 'Platforms' section of a configuration interface. A single platform entry named 'Web' is listed. The 'Allow Implicit Flow' checkbox is checked. Under the 'Redirect URLs' section, the URL 'https://localhost:44320/signin-microsoft' is entered. There is also a 'Logout URL' field with the placeholder 'e.g. https://myapp.com/end-session'.

#### NOTE

O segmento URI `/signin-microsoft` é definido como o retorno de chamada padrão do provedor de autenticação do Microsoft. Você pode alterar o retorno de chamada padrão URI ao configurar o middleware de autenticação da Microsoft por meio de herdadas `RemoteAuthenticationOptions.CallbackPath` propriedade do `MicrosoftAccountOptions` classe.

- Toque **Adicionar URL** para garantir que a URL foi adicionada.
- Preencha quaisquer outras configurações de aplicativo, se necessário e toque em **salvar** na parte inferior da página para salvar as alterações à configuração de aplicativo.
- Ao implantar o site será necessário rever a **registro** página e defina uma nova URL pública.

## Id do aplicativo da Microsoft e a senha de Store

- Observe a `Application Id` exibido na **registro** página.
- Toque **gerar nova senha** na **segredos do aplicativo** seção. Isso exibe uma caixa em que você pode copiar a senha de aplicativo:

# New password generated

This is the only time when it will be displayed. Please store it securely.

bae

Ok

Vincular as configurações confidenciais, como a Microsoft `Application ID` e `Password` para sua configuração de aplicativo usando o [Secret Manager](#). Para os fins deste tutorial, nomeie os tokens `Authentication:Microsoft:ApplicationId` e `Authentication:Microsoft:Password`.

## Configurar a autenticação de conta da Microsoft

O modelo de projeto usado neste tutorial garante que `Microsoft.AspNetCore.Authentication.MicrosoftAccount` pacote já está instalado.

- Para instalar este pacote com o Visual Studio 2017, clique com botão direito no projeto e selecione **gerenciar pacotes NuGet**.
- Para instalar o .NET Core CLI, execute o seguinte comando no diretório do projeto:

```
dotnet add package Microsoft.AspNetCore.Authentication.MicrosoftAccount
```

Adicione o serviço Microsoft Account na `ConfigureServices` método no `Startup.cs` arquivo:

```
services.AddDefaultIdentity<IdentityUser>()
    .AddDefaultUI(UIFramework.Bootstrap4)
    .AddEntityFrameworkStores<ApplicationContext>();

services.AddAuthentication().AddMicrosoftAccount(microsoftOptions =>
{
    microsoftOptions.ClientId = Configuration["Authentication:Microsoft:ApplicationId"];
    microsoftOptions.ClientSecret = Configuration["Authentication:Microsoft:Password"];
});
```

A chamada para `AddIdentity` define as configurações de esquema padrão. O `AddAuthentication(String)` conjuntos de sobrecarregar os `DefaultScheme` propriedade. O `AddAuthentication(ação<AuthenticationOptions>)` sobrecarga permite configurar opções de autenticação, que podem ser usadas para definir os esquemas de autenticação padrão para finalidades diferentes. As chamadas subsequentes para `AddAuthentication` substituição configurada anteriormente `AuthenticationOptions` propriedades.

`AuthenticationBuilder` métodos de extensão que registra um manipulador de autenticação podem ser chamados apenas uma vez por esquema de autenticação. Há sobrecargas que permitem configurar as propriedades de esquema, o nome de esquema e nome de exibição.

## Vários provedores de autenticação

Caso o aplicativo exija vários provedores, encadeie os métodos de extensão do provedor em `AddAuthentication`:

```
services.AddAuthentication()
    .AddMicrosoftAccount(microsoftOptions => { ... })
    .AddGoogle(googleOptions => { ... })
    .AddTwitter(twitterOptions => { ... })
    .AddFacebook(facebookOptions => { ... });
```

Adicione o middleware Account da Microsoft na `Configure` método no `Startup.cs` arquivo:

```
app.UseMicrosoftAccountAuthentication(new MicrosoftAccountOptions()
{
    ClientId = Configuration["Authentication:Microsoft:ApplicationId"],
    ClientSecret = Configuration["Authentication:Microsoft:Password"]
});
```

Embora a terminologia usada no Portal do desenvolvedor Microsoft nomeie esses tokens `ApplicationId` e `Password`, elas são expostas como `ClientId` e `ClientSecret` para a API de configuração.

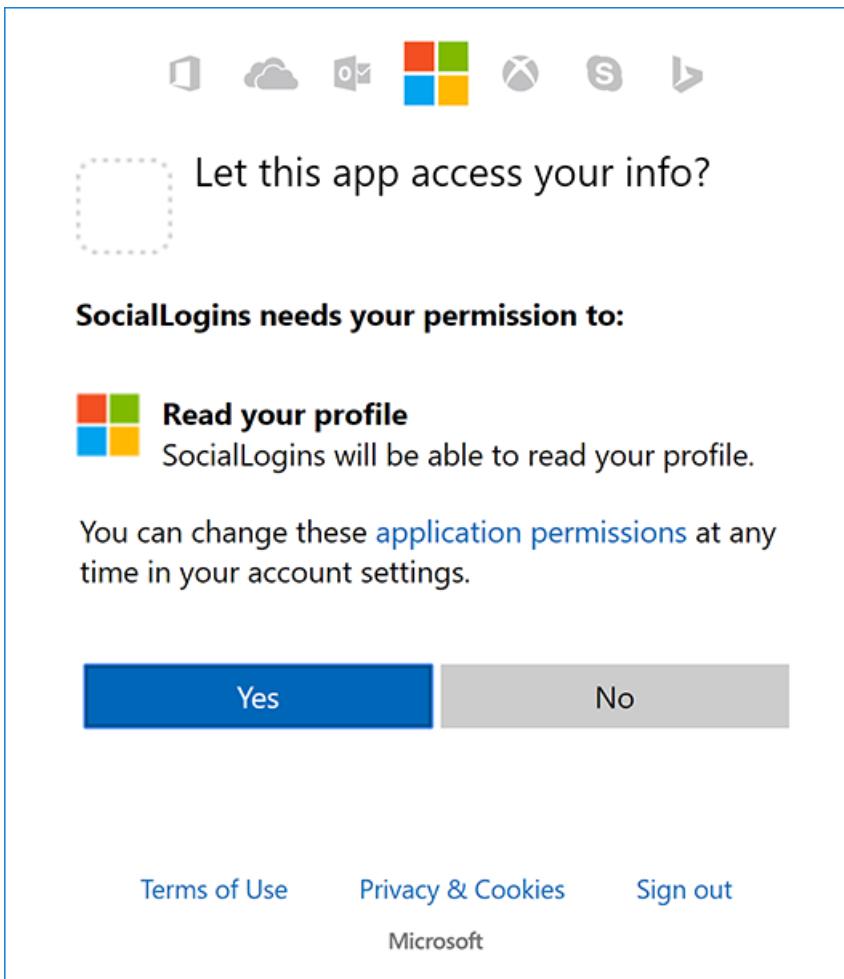
Consulte a [MicrosoftAccountOptions](#) referência da API para obter mais informações sobre opções de configuração com suporte pela autenticação Account da Microsoft. Isso pode ser usado para solicitar informações diferentes sobre o usuário.

## Entrar com conta da Microsoft

Executar o aplicativo e clique em **faça logon no**. Uma opção para entrar com a Microsoft será exibida:

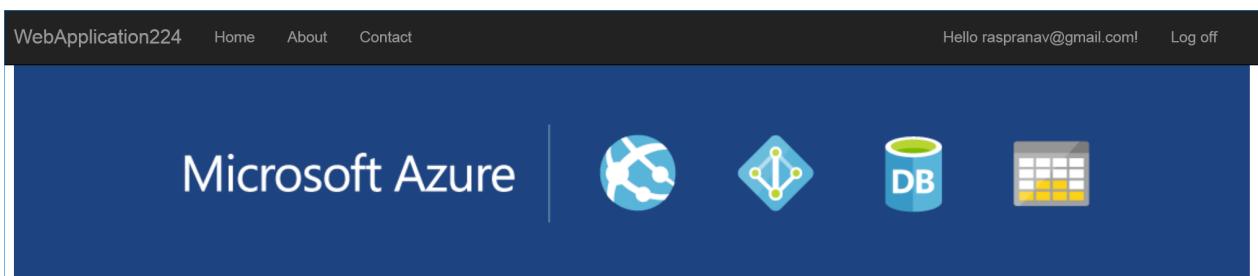
The screenshot shows a web application's login page. At the top, there is a navigation bar with links for Home, About, Contact, Register, and Log in. Below the navigation bar, the page title is "Log in." There are two main sections for logging in: "Use a local account to log in." and "Use another service to log in." The "Local account" section contains fields for Email and Password, and a "Remember me?" checkbox. The "Microsoft" section contains a single button labeled "Microsoft". Below the "Local account" section, there are links for "Register as a new user?" and "Forgot your password?". At the bottom of the page, there is a copyright notice: "© 2016 - WebApplication3".

Quando você clica na Microsoft, você será redirecionado para a Microsoft para autenticação. Após entrar com sua Account da Microsoft (se ainda não estiver conectado), você será solicitado para permitir que o aplicativo acessar suas informações:



Toque **Sim** e você será redirecionado para o site da web onde você pode definir seu email.

Agora você está conectado usando suas credenciais da Microsoft:



## Solicitar informações com um proxy para frente ou balanceador de carga

Se o aplicativo for implantado atrás de um servidor proxy ou um平衡ador de carga, algumas das informações de solicitação original podem ser encaminhadas para o aplicativo nos cabeçalhos de solicitação. Essas informações geralmente incluem o esquema de solicitação de seguro (`https`), host e endereço IP do cliente. Aplicativos não lidos automaticamente esses cabeçalhos de solicitação para descobrir e usar as informações da solicitação original.

O esquema é usado na geração de link que afeta o fluxo de autenticação com provedores externos. Perder o esquema de seguro (`https`) resulta no aplicativo de geração de URLs de redirecionamento inseguro incorreto.

Use o Middleware de cabeçalhos encaminhados para disponibilizar as informações da solicitação original para o aplicativo para o processamento da solicitação.

Para obter mais informações, consulte [Configure o ASP.NET Core para trabalhar com servidores proxy e balanceadores de carga](#).

## Solução de problemas

- Se o provedor Microsoft Account redireciona você para uma página de erro de entrada, observe os erro título e descrição de cadeia de caracteres parâmetros de consulta diretamente após o `#` (hashtag) no Uri.

Embora pareça a mensagem de erro indicar um problema com a autenticação da Microsoft, a causa mais comum é seu Uri não corresponda a um aplicativo de **URIs de redirecionamento** especificado para o **Web** plataforma .

- **ASP.NET Core 2.x somente:** Se a identidade não está configurada por meio da chamada `services.AddIdentity` na `ConfigureServices`, a tentativa de autenticar resultará em *ArgumentException: A opção 'SignInScheme' deve ser fornecida*. O modelo de projeto usado neste tutorial garante que isso é feito.
- Se o banco de dados do site não tiver sido criado aplicando-se a migração inicial, você obterá *uma operação de banco de dados falhou ao processar a solicitação* erro. Toque **aplicar migrações** para criar o banco de dados e atualizar para continuar após o erro.

## Próximas etapas

- Este artigo mostrou como você pode autenticar com a Microsoft. Você pode seguir uma abordagem semelhante para autenticar com outros provedores listados na [página anterior](#).
- Depois de publicar seu site da web para aplicativo web do Azure, você deve criar um novo `Password` no Portal do desenvolvedor Microsoft.
- Defina as `Authentication:Microsoft:ApplicationId` e `Authentication:Microsoft:Password` como configurações de aplicativo no portal do Azure. Configurar o sistema de configuração para ler as chaves de variáveis de ambiente.

# Configuração de logon externo do Twitter com o ASP.NET Core

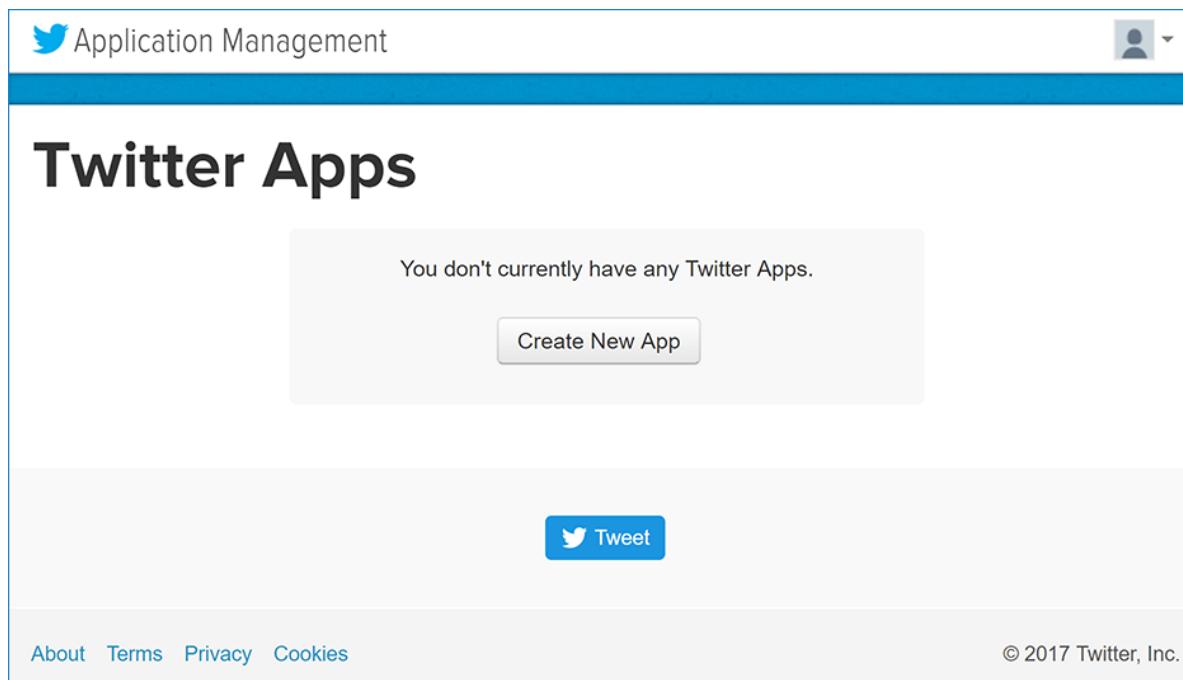
27/12/2018 • 8 minutes to read • [Edit Online](#)

Por [Valeriy Novytskyy](#) e [Rick Anderson](#)

Este tutorial mostra como permitir que os usuários [entrar com sua conta do Twitter](#) usando um projeto do ASP.NET Core 2.0 de exemplo criado sobre o [página anterior](#).

## Criar o aplicativo no Twitter

- Navegue até <https://apps.twitter.com/> e entre. Se você ainda não tiver uma conta do Twitter, use o [Inscreve-se agora](#) link para criar um. Depois de entrar, a **gerenciamento de aplicativos** página é mostrada:



- Toque **criar novo aplicativo** e preencha o requerimento **nome, descrição** e pública **site** URI (Isso pode ser temporário até que você Registre o nome de domínio):

## Create an application

### Application Details

**Name \****Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.***Description \****Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.***Website \****Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.**(If you don't have a URL yet, just put a placeholder here but remember to change it later.)***Callback URL***Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth\_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.*

### Developer Agreement

 Yes, I have read and agree to the [Twitter Developer Agreement](#).[Create your Twitter application](#)

- Insira o URI de desenvolvimento com `/signin-twitter` acrescentados em de **URIs de redirecionamento de OAuth válido** campo (por exemplo: `https://localhost:44320/signin-twitter`). O esquema de autenticação do Twitter configurado mais tarde neste tutorial automaticamente manipulará as solicitações em `/signin-twitter` rota para implementar o fluxo de OAuth.

**NOTE**

O segmento URI `/signin-twitter` é definido como o retorno de chamada padrão do provedor de autenticação do Twitter. Você pode alterar o retorno de chamada padrão URI ao configurar o middleware de autenticação do Twitter por meio de herdadas `RemoteAuthenticationOptions.CallbackPath` propriedade da `TwitterOptions` classe.

- Preencha o restante do formulário e toque **criar seu aplicativo Twitter**. Novos detalhes do aplicativo são exibidos:



Your application has been created. Please take a moment to review and adjust your application's settings.

# SocialLogins

[Test OAuth](#)[Details](#)[Settings](#)[Keys and Access Tokens](#)[Permissions](#)

Social login demo

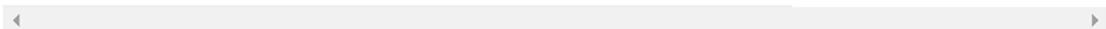
<https://www.mysite.com/>

## Organization

Information about the organization or company associated with your application. This information is optional.

Organization None

Organization website None



## Application Settings

Your application's Consumer Key and Secret are used to [authenticate](#) requests to the Twitter Platform.

Access level Read and write ([modify app permissions](#))[Consumer Key \(API Key\)](#)[\(manage keys and access\)](#)

- Ao implantar o site será necessário rever a **gerenciamento de aplicativos** página e registrar um novo URI público.

## Armazenar Twitter ConsumerKey e ConsumerSecret

Vincular as configurações confidenciais, como o Twitter `Consumer Key` e `Consumer Secret` para sua configuração de aplicativo usando o [Secret Manager](#). Para os fins deste tutorial, nomeie os tokens

`Authentication:Twitter:ConsumerKey` e `Authentication:Twitter:ConsumerSecret`.

Esses tokens podem ser encontrados na **chaves e Tokens de acesso** guia depois de criar seu novo aplicativo do Twitter:



# SocialLogins

[Test OAuth](#)[Details](#)[Settings](#)[Keys and Access Tokens](#)[Permissions](#)

## Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)

Consumer Secret (API Secret)

Access Level      Read and write ([modify app permissions](#))

Owner

Owner ID      890753103795859456

## Application Actions

[Regenerate Consumer Key and Secret](#)[Change App Permissions](#)

## Configurar a autenticação do Twitter

O modelo de projeto usado neste tutorial garante que [Microsoft.AspNetCore.Authentication.Twitter](#) pacote já está instalado.

- Para instalar este pacote com o Visual Studio 2017, clique com botão direito no projeto e selecione **gerenciar pacotes NuGet**.
- Para instalar o .NET Core CLI, execute o seguinte comando no diretório do projeto:

```
dotnet add package Microsoft.AspNetCore.Authentication.Twitter
```

Adicione o serviço do Twitter na `ConfigureServices` método no `Startup.cs` arquivo:

```
services.AddDefaultIdentity<IdentityUser>()
    .AddDefaultUI(UIFramework.Bootstrap4)
    .AddEntityFrameworkStores<ApplicationContext>();

services.AddAuthentication().AddTwitter(twitterOptions =>
{
    twitterOptions.ConsumerKey = Configuration["Authentication:Twitter:ConsumerKey"];
    twitterOptions.ConsumerSecret = Configuration["Authentication:Twitter:ConsumerSecret"];
});
```

A chamada para `AddIdentity` define as configurações de esquema padrão. O `AddAuthentication(String)` conjuntos de sobrecarregar os `DefaultScheme` propriedade. O `AddAuthentication(ação<AuthenticationOptions>)` sobrecarga permite configurar opções de autenticação, que podem ser usadas para definir os esquemas de autenticação padrão para finalidades diferentes. As chamadas subsequentes para `AddAuthentication` substituição

configurada anteriormente [AuthenticationOptions](#) propriedades.

[AuthenticationBuilder](#) métodos de extensão que registra um manipulador de autenticação podem ser chamados apenas uma vez por esquema de autenticação. Há sobrecargas que permitem configurar as propriedades de esquema, o nome de esquema e nome de exibição.

## Vários provedores de autenticação

Caso o aplicativo exija vários provedores, encadeie os métodos de extensão do provedor em [AddAuthentication](#):

```
services.AddAuthentication()
    .AddMicrosoftAccount(microsoftOptions => { ... })
    .AddGoogle(googleOptions => { ... })
    .AddTwitter(twitterOptions => { ... })
    .AddFacebook(facebookOptions => { ... });
```

Adicione o middleware do Twitter na [Configure](#) método no *Startup.cs* arquivo:

```
app.UseTwitterAuthentication(new TwitterOptions()
{
    ConsumerKey = Configuration["Authentication:Twitter:ConsumerKey"],
    ConsumerSecret = Configuration["Authentication:Twitter:ConsumerSecret"]
});
```

Consulte a [TwitterOptions](#) referência da API para obter mais informações sobre opções de configuração com suporte pela autenticação do Twitter. Isso pode ser usado para solicitar informações diferentes sobre o usuário.

## Entrar com o Twitter

Executar o aplicativo e clique em **faça logon no**. Será exibida uma opção para entrar com o Twitter:

The screenshot shows a login interface for a web application named "WebApplication3". The top navigation bar has links for "Home", "About", "Contact", "Register", and "Log in". The main content area is divided into two sections. The left section is titled "Log in." and says "Use a local account to log in." It features input fields for "Email" and "Password", and a "Remember me?" checkbox followed by a "Log in" button. The right section is titled "Use another service to log in." and features a "Twitter" button. At the bottom of the page, there are links for "Register as a new user?" and "Forgot your password?", and a copyright notice at the very bottom: "© 2016 - WebApplication3".

Clicar em **Twitter** redireciona para o Twitter para autenticação:


[Sign up for Twitter >](#)

## Authorize SocialLogins to use your account?



**SocialLogins**  
www.mysite.com  
Social login demo

Remember me · [Forgot password?](#)

[Sign In](#)
[Cancel](#)

**This application will be able to:**

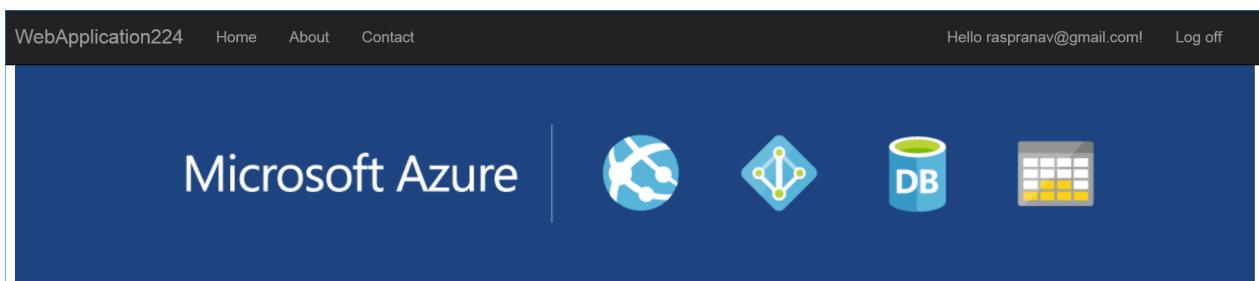
- Read Tweets from your timeline.
- See who you follow.

**Will not be able to:**

- Follow new people.
- Update your profile.
- Post Tweets for you.
- Access your direct messages.
- See your email address.
- See your Twitter password.

Depois de inserir suas credenciais do Twitter, você será redirecionado para o site da web onde você pode definir seu email.

Agora você está conectado usando suas credenciais do Twitter:



## Solicitar informações com um proxy para frente ou平衡ador de carga

Se o aplicativo for implantado atrás de um servidor proxy ou um balanceador de carga, algumas das informações de solicitação original podem ser encaminhadas para o aplicativo nos cabeçalhos de solicitação. Essas informações geralmente incluem o esquema de solicitação de seguro (`https`), host e endereço IP do cliente. Aplicativos não lidos automaticamente esses cabeçalhos de solicitação para descobrir e usar as informações da solicitação original.

O esquema é usado na geração de link que afeta o fluxo de autenticação com provedores externos. Perder o esquema de seguro (`https`) resulta no aplicativo de geração de URLs de redirecionamento inseguro incorreto.

Use o Middleware de cabeçalhos encaminhados para disponibilizar as informações da solicitação original para o aplicativo para o processamento da solicitação.

Para obter mais informações, consulte [Configure o ASP.NET Core para trabalhar com servidores proxy e平衡adores de carga](#).

## Solução de problemas

- **ASP.NET Core 2.x somente:** Se a identidade não está configurada por meio da chamada `services.AddIdentity` na `ConfigureServices`, a tentativa de autenticar resultará em *ArgumentException: A opção 'SignInScheme' deve ser fornecida*. O modelo de projeto usado neste tutorial garante que isso é feito.
- Se o banco de dados do site não tiver sido criado aplicando-se a migração inicial, você obterá *uma operação de banco de dados falhou ao processar a solicitação* erro. Toque **aplicar migrações** para criar o banco de dados e atualizar para continuar após o erro.

## Próximas etapas

- Este artigo mostrou como você pode autenticar com o Twitter. Você pode seguir uma abordagem semelhante para autenticar com outros provedores listados na [página anterior](#).
- Depois de publicar seu site da web para aplicativo web do Azure, você deve redefinir o `ConsumerSecret` no portal do desenvolvedor do Twitter.
- Defina as `Authentication:Twitter:ConsumerKey` e `Authentication:Twitter:ConsumerSecret` como configurações de aplicativo no portal do Azure. Configurar o sistema de configuração para ler as chaves de variáveis de ambiente.

# Provedores de autenticação OAuth externos

16/11/2018 • 2 minutes to read • [Edit Online](#)

Por [Rick Anderson](#), [Pranav Rastogi](#), e [Valeriy Novytskyy](#)

A lista a seguir inclui comuns OAuth provedores de autenticação externos que trabalham com aplicativos ASP.NET Core. Pacotes do NuGet de terceiros, como aqueles mantidos pelo [aspnet-contrib](#), pode ser usado para complementar os provedores de autenticação implementados pela equipe do ASP.NET Core.

- [LinkedIn \(instruções\)](#)
- [Instagram \(instruções\)](#)
- [Reddit \(instruções\)](#)
- [GitHub \(instruções\)](#)
- [Yahoo \(instruções\)](#)
- [Tumblr \(instruções\)](#)
- [Pinterest \(instruções\)](#)
- [Pocket \(instruções\)](#)
- [Flickr \(instruções\)](#)
- [Dribble \(instruções\)](#)
- [Vimeo \(instruções\)](#)
- [SoundCloud \(instruções\)](#)
- [VK \(instruções\)](#)

## Vários provedores de autenticação

Caso o aplicativo exija vários provedores, encadeie os métodos de extensão do provedor em [AddAuthentication](#):

```
services.AddAuthentication()
    .AddMicrosoftAccount(microsoftOptions => { ... })
    .AddGoogle(googleOptions => { ... })
    .AddTwitter(twitterOptions => { ... })
    .AddFacebook(facebookOptions => { ... });
```

## Solicitar informações com um proxy para frente ou balanceador de carga

Se o aplicativo for implantado atrás de um servidor proxy ou um balanceador de carga, algumas das informações de solicitação original podem ser encaminhadas para o aplicativo nos cabeçalhos de solicitação. Essas informações geralmente incluem o esquema de solicitação de seguro (`https`), host e endereço IP do cliente. Aplicativos não lidas automaticamente esses cabeçalhos de solicitação para descobrir e usar as informações da solicitação original.

O esquema é usado na geração de link que afeta o fluxo de autenticação com provedores externos. Perder o esquema de seguro (`https`) resulta no aplicativo de geração de URLs de redirecionamento inseguro incorreto.

Use o Middleware de cabeçalhos encaminhados para disponibilizar as informações da solicitação original para o aplicativo para o processamento da solicitação.

Para obter mais informações, consulte [Configure o ASP.NET Core para trabalhar com servidores proxy e平衡adores de carga](#).

# Manter declarações adicionais e os tokens de provedores externos no ASP.NET Core

16/11/2018 • 11 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

Um aplicativo ASP.NET Core pode estabelecer declarações adicionais e tokens de provedores de autenticação externa, como Facebook, Google, Microsoft e Twitter. Cada provedor revela informações diferentes sobre os usuários em sua plataforma, mas o padrão para receber e transformar dados de usuário em declarações adicionais é o mesmo.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Pré-requisitos

Decida quais provedores de autenticação externa para dar suporte a no aplicativo. Para cada provedor, registrar o aplicativo e obter o ID do cliente e segredo do cliente. Para obter mais informações, consulte [Autenticação de Facebook, Google e de provedor externo no ASP.NET Core](#). O [aplicativo de exemplo](#) usa o [provedor de autenticação do Google](#).

## Defina a ID do cliente e segredo do cliente

O provedor de autenticação OAuth estabelece uma relação de confiança com um aplicativo usando uma ID de cliente e o segredo do cliente. ID do cliente e valores de segredo do cliente são criados para o aplicativo pelo provedor de autenticação externa quando o aplicativo é registrado com o provedor. Cada provedor externo que usa o aplicativo deve ser configurado independentemente com a ID do cliente e o segredo do cliente do provedor. Para obter mais informações, consulte os tópicos de provedor de autenticação externa que se aplicam ao seu cenário:

- [Autenticação do Facebook](#)
- [Autenticação do Google](#)
- [Autenticação da Microsoft](#)
- [Autenticação do Twitter](#)
- [Outros provedores de autenticação](#)
- [OpenIdConnect](#)

O aplicativo de exemplo configura o provedor de autenticação do Google com uma ID do cliente e segredo do cliente fornecido pelo Google:

```

services.AddAuthentication().AddGoogle(options =>
{
    // Provide the Google Client ID
    options.ClientId = "XXXXXXXXXX.apps.googleusercontent.com";
    // Provide the Google Secret
    options.ClientSecret = "g4GZ2#...GD5Gg1x";
    options.Scope.Add("https://www.googleapis.com/auth/plus.login");
    options.ClaimActions.MapJsonKey(ClaimTypes.Gender, "gender");
    options.SaveTokens = true;
    options.Events.OnCreatingTicket = ctx =>
    {
        List<AuthenticationToken> tokens = ctx.Properties.GetTokens()
            as List<AuthenticationToken>;
        tokens.Add(new AuthenticationToken()
        {
            Name = "TicketCreated",
            Value = DateTime.UtcNow.ToString()
        });
        ctx.Properties.StoreTokens(tokens);
        return Task.CompletedTask;
    };
});

```

## Estabelecer o escopo de autenticação

Especifique a lista de permissões para recuperar do provedor, especificando o [Scope](#). Escopos de autenticação para provedores externos comuns aparecem na tabela a seguir.

PROVIDER	ESCOPO
Facebook	<a href="https://www.facebook.com/dialog/oauth">https://www.facebook.com/dialog/oauth</a>
Google	<a href="https://www.googleapis.com/auth/plus.login">https://www.googleapis.com/auth/plus.login</a>
Microsoft	<a href="https://login.microsoftonline.com/common/oauth2/v2.0/authorize">https://login.microsoftonline.com/common/oauth2/v2.0/authorize</a>
Twitter	<a href="https://api.twitter.com/oauth/authenticate">https://api.twitter.com/oauth/authenticate</a>

O aplicativo de exemplo adiciona o Google `plus.login` escopo para solicitar a entrada do Google + nas permissões:

```

services.AddAuthentication().AddGoogle(options =>
{
    // Provide the Google Client ID
    options.ClientId = "XXXXXXXXXXXX.apps.googleusercontent.com";
    // Provide the Google Secret
    options.ClientSecret = "g4GZ2#...GD5Gg1x";
    options.Scope.Add("https://www.googleapis.com/auth/plus.login");
    options.ClaimActions.MapJsonKey(ClaimTypes.Gender, "gender");
    options.SaveTokens = true;
    options.Events.OnCreatingTicket = ctx =>
    {
        List<AuthenticationToken> tokens = ctx.Properties.GetTokens()
            as List<AuthenticationToken>;
        tokens.Add(new AuthenticationToken()
        {
            Name = "TicketCreated",
            Value = DateTime.UtcNow.ToString()
        });
        ctx.Properties.StoreTokens(tokens);
        return Task.CompletedTask;
    };
});

```

## Mapear chaves de dados de usuário e criar declarações

Nas opções do provedor, especifique um [MapJsonKey](#) para cada chave nos dados de usuário do provedor externo JSON para a identidade do aplicativo ler em entrar. Para obter mais informações sobre tipos de declaração, consulte [ClaimTypes](#).

O aplicativo de exemplo cria um [Gender](#) reivindicar do `gender` chave nos dados de usuário do Google:

```

services.AddAuthentication().AddGoogle(options =>
{
    // Provide the Google Client ID
    options.ClientId = "XXXXXXXXXXXX.apps.googleusercontent.com";
    // Provide the Google Secret
    options.ClientSecret = "g4GZ2#...GD5Gg1x";
    options.Scope.Add("https://www.googleapis.com/auth/plus.login");
    options.ClaimActions.MapJsonKey(ClaimTypes.Gender, "gender");
    options.SaveTokens = true;
    options.Events.OnCreatingTicket = ctx =>
    {
        List<AuthenticationToken> tokens = ctx.Properties.GetTokens()
            as List<AuthenticationToken>;
        tokens.Add(new AuthenticationToken()
        {
            Name = "TicketCreated",
            Value = DateTime.UtcNow.ToString()
        });
        ctx.Properties.StoreTokens(tokens);
        return Task.CompletedTask;
    };
});

```

Na [OnPostConfirmationAsync](#), uma [IdentityUser](#) ( `ApplicationUser` ) é conectado ao aplicativo com [SignInAsync](#).

Durante o logon no processo, o [UserManager<TUser>](#) pode armazenar um  `ApplicationUser`  de declaração para os dados de usuário disponíveis no [Principal](#).

No aplicativo de exemplo,  `OnPostConfirmationAsync`  ( `Account/ExternalLogin.cshtml.cs` ) estabelece uma [Gender](#) de declaração para assinado no  `ApplicationUser` :

```

public async Task<IActionResult> OnPostConfirmationAsync(
    string returnUrl = null)
{
    if (ModelState.IsValid)
    {
        // Get the information about the user from the external login
        // provider
        var info = await _signInManager.GetExternalLoginInfoAsync();

        if (info == null)
        {
            throw new ApplicationException(
                "Error loading external login data during confirmation.");
        }

        var user = new ApplicationUser
        {
            UserName = Input.Email,
            Email = Input.Email
        };
        var result = await _userManager.CreateAsync(user);

        if (result.Succeeded)
        {
            result = await _userManager.AddLoginAsync(user, info);

            if (result.Succeeded)
            {
                // Copy over the gender claim
                await _userManager.AddClaimAsync(user,
                    info.Principal.FindFirst(ClaimTypes.Gender));

                // Include the access token in the properties
                var props = new AuthenticationProperties();
                props.StoreTokens(info.AuthenticationTokens);

                await _signInManager.SignInAsync(user, props,
                    authenticationMethod: info.LoginProvider);
                _logger.LogInformation(
                    "User created an account using {Name} provider.",
                    info.LoginProvider);
            }

            return LocalRedirect(Url.GetLocalUrl(returnUrl));
        }
    }

    foreach (var error in result.Errors)
    {
        ModelState.AddModelError(string.Empty, error.Description);
    }
}

ReturnUrl = returnUrl;

return Page();
}

```

## Salve o token de acesso

[SaveTokens](#) Define se os tokens de acesso e de atualização devem ser armazenadas do [AuthenticationProperties](#) após uma autorização bem-sucedida. `SaveTokens` é definido como `false` por padrão para reduzir o tamanho do cookie de autenticação final.

O aplicativo de exemplo define o valor de `SaveTokens` à `true` em [GoogleOptions](#):

```

services.AddAuthentication().AddGoogle(options =>
{
    // Provide the Google Client ID
    options.ClientId = "XXXXXXXXXXXX.apps.googleusercontent.com";
    // Provide the Google Secret
    options.ClientSecret = "g4GZ2#...GD5Gg1x";
    options.Scope.Add("https://www.googleapis.com/auth/plus.login");
    options.ClaimActions.MapJsonKey(ClaimTypes.Gender, "gender");
    options.SaveTokens = true;
    options.Events.OnCreatingTicket = ctx =>
    {
        List<AuthenticationToken> tokens = ctx.Properties.GetTokens()
            as List<AuthenticationToken>;
        tokens.Add(new AuthenticationToken()
        {
            Name = "TicketCreated",
            Value = DateTime.UtcNow.ToString()
        });
        ctx.Properties.StoreTokens(tokens);
        return Task.CompletedTask;
    };
});

```

Quando `OnPostConfirmationAsync` é executado, armazenar o token de acesso (`ExternalLoginInfo.AuthenticationTokens`) do provedor externo no  `ApplicationUser` do  `AuthenticationProperties`.

O aplicativo de exemplo salva o token de acesso em:

- `OnPostConfirmationAsync` – É executado para o novo registro do usuário.
- `OnGetCallbackAsync` – Executa quando um usuário registrado anteriormente entra no aplicativo.

*Account/ExternalLogin.cshtml.cs:*

```

public async Task<IActionResult> OnPostConfirmationAsync(
    string returnUrl = null)
{
    if (ModelState.IsValid)
    {
        // Get the information about the user from the external login
        // provider
        var info = await _signInManager.GetExternalLoginInfoAsync();

        if (info == null)
        {
            throw new ApplicationException(
                "Error loading external login data during confirmation.");
        }

        var user = new ApplicationUser
        {
            UserName = Input.Email,
            Email = Input.Email
        };
        var result = await _userManager.CreateAsync(user);

        if (result.Succeeded)
        {
            result = await _userManager.AddLoginAsync(user, info);

            if (result.Succeeded)
            {
                // Copy over the gender claim
                await _userManager.AddClaimAsync(user,
                    info.Principal.FindFirst(ClaimTypes.Gender));

                // Include the access token in the properties
                var props = new AuthenticationProperties();
                props.StoreTokens(info.AuthenticationTokens);

                await _signInManager.SignInAsync(user, props,
                    authenticationMethod: info.LoginProvider);
                _logger.LogInformation(
                    "User created an account using {Name} provider.",
                    info.LoginProvider);
            }

            return LocalRedirect(Url.GetLocalUrl(returnUrl));
        }
    }

    foreach (var error in result.Errors)
    {
        ModelState.AddModelError(string.Empty, error.Description);
    }
}

ReturnUrl = returnUrl;

return Page();
}

```

```

public async Task<IActionResult> OnGetCallbackAsync(
    string returnUrl = null, string remoteError = null)
{
    if (remoteError != null)
    {
        ErrorMessage = $"Error from external provider: {remoteError}";

        return RedirectToPage("./Login");
    }

    var info = await _signInManager.GetExternalLoginInfoAsync();

    if (info == null)
    {
        return RedirectToPage("./Login");
    }

    // Sign in the user with this external login provider if the user
    // already has a login
    var result = await _signInManager.ExternalLoginSignInAsync(
        info.LoginProvider, info.ProviderKey, isPersistent: false,
        bypassTwoFactor : true);

    if (result.Succeeded)
    {
        // Store the access token and resign in so the token is included in
        // in the cookie
        var user = await _userManager.FindByLoginAsync(info.LoginProvider,
            info.ProviderKey);

        var props = new AuthenticationProperties();
        props.StoreTokens(info.AuthenticationTokens);

        await _signInManager.SignInAsync(user, props, info.LoginProvider);

        _logger.LogInformation(
            "{Name} logged in with {LoginProvider} provider.",
            info.Principal.Identity.Name, info.LoginProvider);
    }

    return LocalRedirect(Url.GetLocalUrl(returnUrl));
}

if (result.IsLockedOut)
{
    return RedirectToPage("./Lockout");
}
else
{
    // If the user does not have an account, then ask the user to
    // create an account
    ReturnUrl = returnUrl;
    LoginProvider = info.LoginProvider;

    if (info.Principal.HasClaim(c => c.Type == ClaimTypes.Email))
    {
        Input = new InputModel
        {
            Email = info.Principal.FindFirstValue(ClaimTypes.Email)
        };
    }

    return Page();
}
}

```

## Como adicionar tokens personalizados adicionais

Para demonstrar como adicionar um token personalizado, que é armazenado como parte da `SaveTokens`, o aplicativo de exemplo adiciona um `AuthenticationToken` com o atual `DateTime` para um `AuthenticationToken.Name` de `TicketCreated`:

```
services.AddAuthentication().AddGoogle(options =>
{
    // Provide the Google Client ID
    options.ClientId = "XXXXXXXXXX.apps.googleusercontent.com";
    // Provide the Google Secret
    options.ClientSecret = "g4GZ2#...GD5Gg1x";
    options.Scope.Add("https://www.googleapis.com/auth/plus.login");
    options.ClaimActions.MapJsonKey(ClaimTypes.Gender, "gender");
    options.SaveTokens = true;
    options.Events.OnCreatingTicket = ctx =>
    {
        List<AuthenticationToken> tokens = ctx.Properties.GetTokens()
            as List<AuthenticationToken>;
        tokens.Add(new AuthenticationToken()
        {
            Name = "TicketCreated",
            Value = DateTime.UtcNow.ToString()
        });
        ctx.Properties.StoreTokens(tokens);
        return Task.CompletedTask;
    };
});
```

## Instruções do aplicativo de exemplo

O aplicativo de exemplo demonstra como:

- Obtenha o sexo do usuário do Google e armazene uma declaração de sexo com o valor.
- Store o token de acesso do Google, o usuário `AuthenticationProperties`.

Para usar o aplicativo de exemplo:

1. Registrar o aplicativo e obter uma ID de cliente válido e o segredo do cliente para autenticação do Google. Para obter mais informações, consulte [Configuração de logon externo do Google no ASP.NET Core](#).
2. Forneça a ID de cliente e o segredo do cliente para o aplicativo na `GoogleOptions` de `Startup.ConfigureServices`.
3. Execute o aplicativo e solicite a página Minhas declarações. Quando o usuário não estiver conectado, o aplicativo redireciona para o Google. Entrar com o Google. Google redireciona o usuário retorno ao aplicativo (`/Home/MyClaims`). O usuário é autenticado e a página Minhas declarações é carregada. A declaração de gênero esteja presente em **declarações de usuário** com o valor obtido do Google. O token de acesso será exibido na **as propriedades de autenticação**.

#### User Claims

```
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier
    b36a7b09-9135-4810-b7a5-78697ff23e99
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name
    username@gmail.com
AspNet.Identity.SecurityStamp
    2962TB881ATCUQFJSRFG1S0QJ000AWVT
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/gender
    female
http://schemas.microsoft.com/ws/2008/06/identity/claims/authenticationmethod
    Google
```

#### Authentication Properties

```
.Token.access_token
    bv42.Dgw...GQMv9ArLPs
.Token.token_type
    Bearer
.Token.expires_at
    2018-08-27T19:08:00.0000000+00:00
.Token.TicketCreated
    8/27/2018 6:08:00 PM
.TokenNames
    access_token;token_type;expires_at;TicketCreated
.issued
    Mon, 27 Aug 2018 18:08:05 GMT
.expires
    Mon, 10 Sep 2018 18:08:05 GMT
```

## Solicitar informações com um proxy para frente ou平衡ador de carga

Se o aplicativo for implantado atrás de um servidor proxy ou um balanceador de carga, algumas das informações de solicitação original podem ser encaminhadas para o aplicativo nos cabeçalhos de solicitação. Essas informações geralmente incluem o esquema de solicitação de seguro (<https>), host e endereço IP do cliente. Aplicativos não lidas automaticamente esses cabeçalhos de solicitação para descobrir e usar as informações da solicitação original.

O esquema é usado na geração de link que afeta o fluxo de autenticação com provedores externos. Perder o esquema de seguro (<https>) resulta no aplicativo de geração de URLs de redirecionamento inseguro incorreto.

Use o Middleware de cabeçalhos encaminhados para disponibilizar as informações da solicitação original para o aplicativo para o processamento da solicitação.

Para obter mais informações, consulte [Configure o ASP.NET Core para trabalhar com servidores proxy e平衡adores de carga](#).

# Autenticar usuários com o WS-Federation no ASP.NET Core

18/01/2019 • 7 minutes to read • [Edit Online](#)

Este tutorial demonstra como habilitar usuários entrar com um provedor de autenticação do WS-Federation, como o Active Directory Federation Services (ADFS) ou [Azure Active Directory](#) (AAD). Ele usa o aplicativo de exemplo do ASP.NET Core 2.0 descrito em [Facebook, Google e a autenticação de provedor externo](#).

Para aplicativos ASP.NET Core 2.0, o suporte do WS-Federation é fornecido pela [Microsoft.AspNetCore.Authentication.WsFederation](#). Esse componente foi transferido da [Microsoft.Owin.Security.WsFederation](#) e compartilha muitas das mecânicas do componente. No entanto, os componentes são diferentes de duas maneiras importantes.

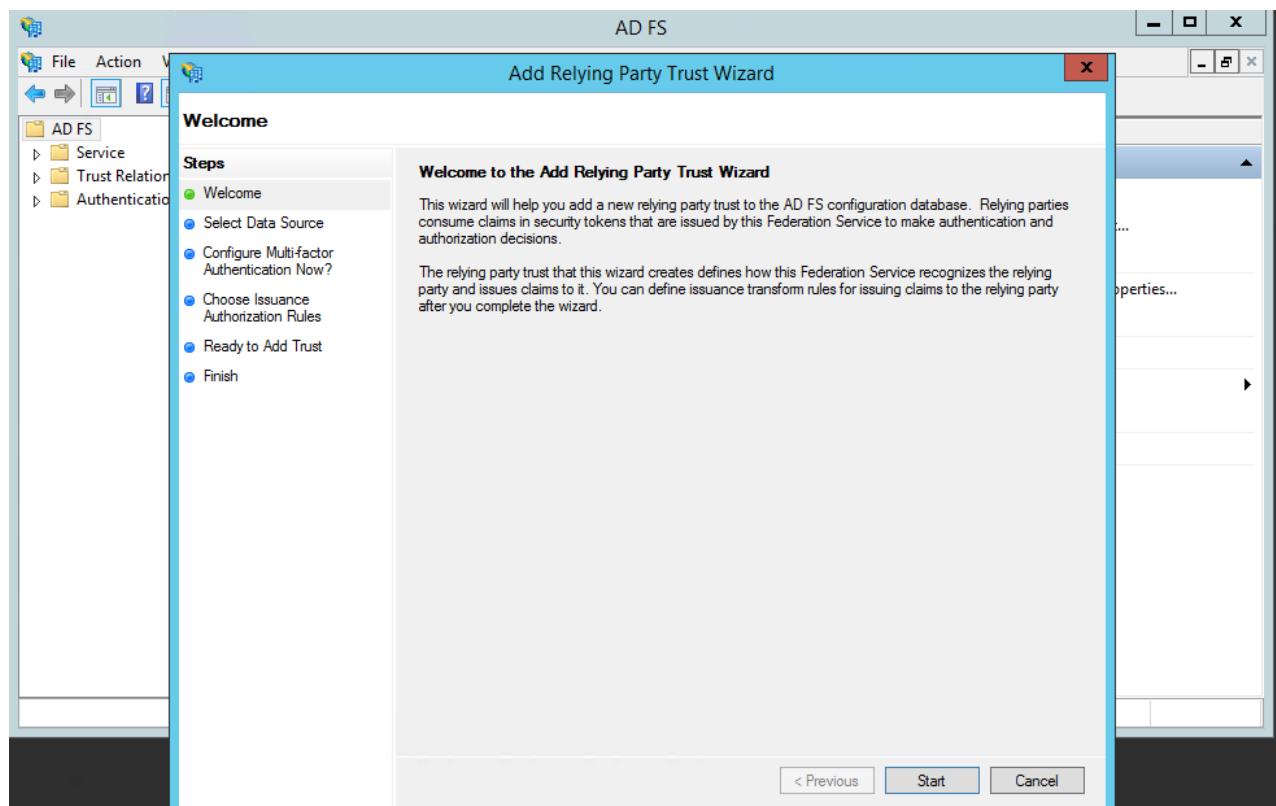
Por padrão, o middleware novo:

- Não permitir logons não solicitados. Esse recurso do protocolo WS-Federation é vulnerável a ataques de XSRF. No entanto, ele pode ser habilitado com o `AllowUnsolicitedLogins` opção.
- Não verifica cada postagem de formulário para mensagens de entrada. Somente as solicitações para o `CallbackPath` são verificados quanto à entrada-ins. `CallbackPath` assume como padrão `/signin-wsfed` mas pode ser alterado por meio de herdado `RemoteAuthenticationOptions.CallbackPath` propriedade do `WsFederationOptions` classe. Esse caminho pode ser compartilhado com outros provedores de autenticação, permitindo que o `SkipUnrecognizedRequests` opção.

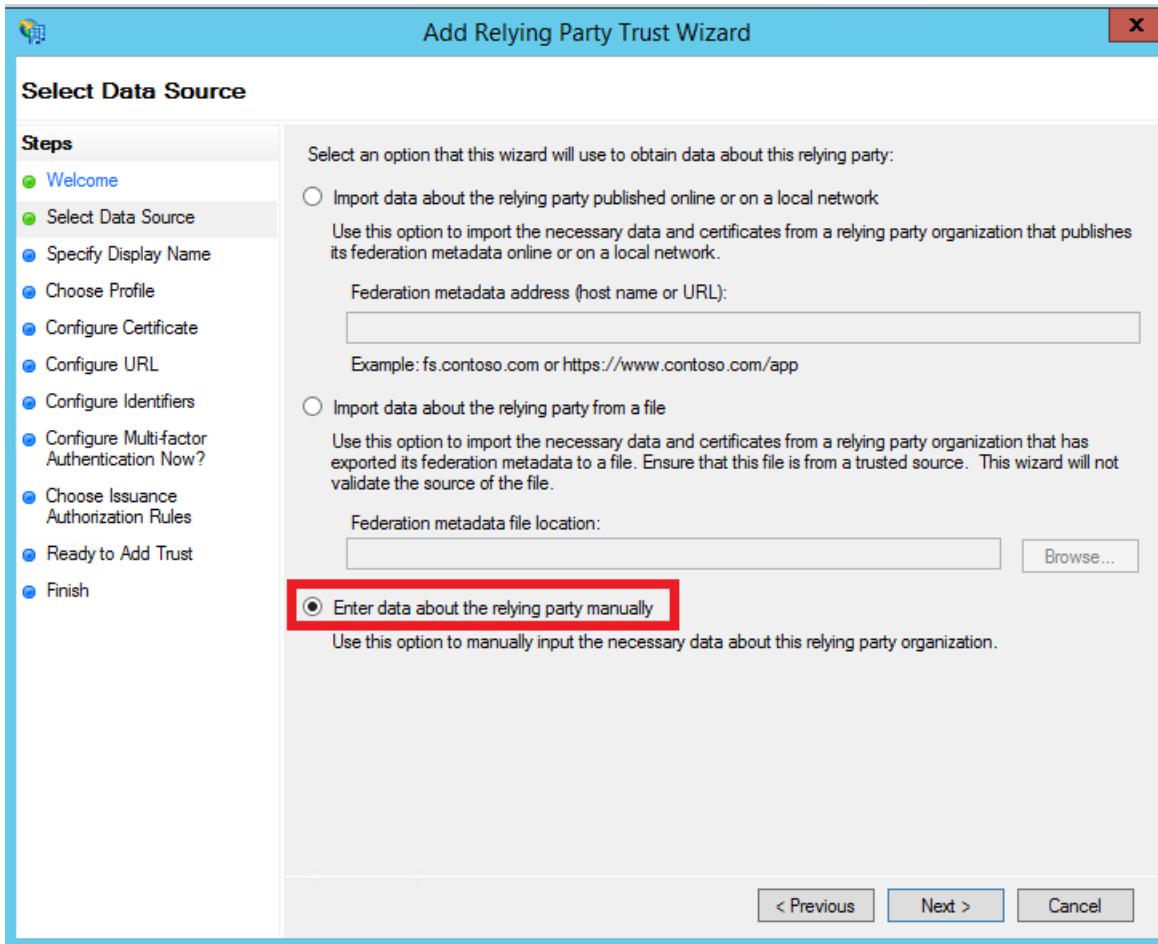
## Registrar o aplicativo com o Active Directory

### Serviços de Federação do Active Directory

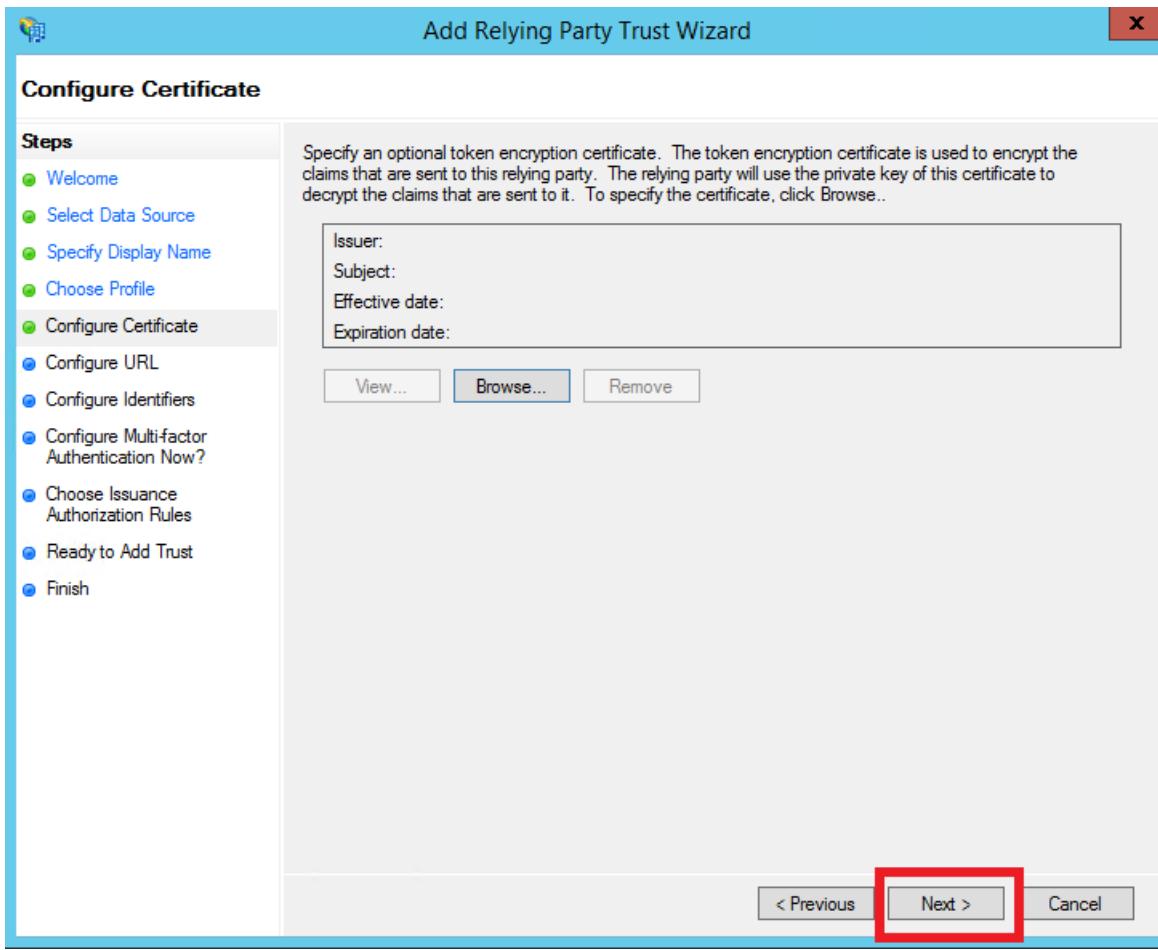
- Abra o servidor **adicionar da terceira parte confiável Assistente confiança** no console de gerenciamento do AD FS:



- Escolha esta opção para inserir os dados manualmente:



- Insira um nome de exibição para a terceira parte confiável. O nome não é importante para o aplicativo ASP.NET Core.
- [Microsoft.AspNetCore.Authentication.WsFederation](#) não tem suporte para criptografia de token, portanto, não configure um certificado de criptografia do token:



- Habilite o suporte para o protocolo WS-Federation Passive, usando a URL do aplicativo. Verifique se que a porta está correta para o aplicativo:

 Add Relying Party Trust Wizard X

### Configure URL

**Steps**

- Welcome
- Select Data Source
- Specify Display Name
- Choose Profile
- Configure Certificate
- **Configure URL**
- Configure Identifiers
- Configure Multi-factor Authentication Now?
- Choose Issuance Authorization Rules
- Ready to Add Trust
- Finish

AD FS supports the WS-Trust, WS-Federation and SAML 2.0 WebSSO protocols for relying parties. If WS-Federation, SAML, or both are used by the relying party, select the check boxes for them and specify the URLs to use. Support for the WS-Trust protocol is always enabled for a relying party.

Enable support for the WS-Federation Passive protocol

The WS-Federation Passive protocol URL supports Web-browser-based claims providers using the WS-Federation Passive protocol.

Relying party WS-Federation Passive protocol URL:

Example: https://fs.contoso.com/adfs/ls/

Enable support for the SAML 2.0 WebSSO protocol

The SAML 2.0 single-sign-on (SSO) service URL supports Web-browser-based claims providers using the SAML 2.0 WebSSO protocol.

Relying party SAML 2.0 SSO service URL:

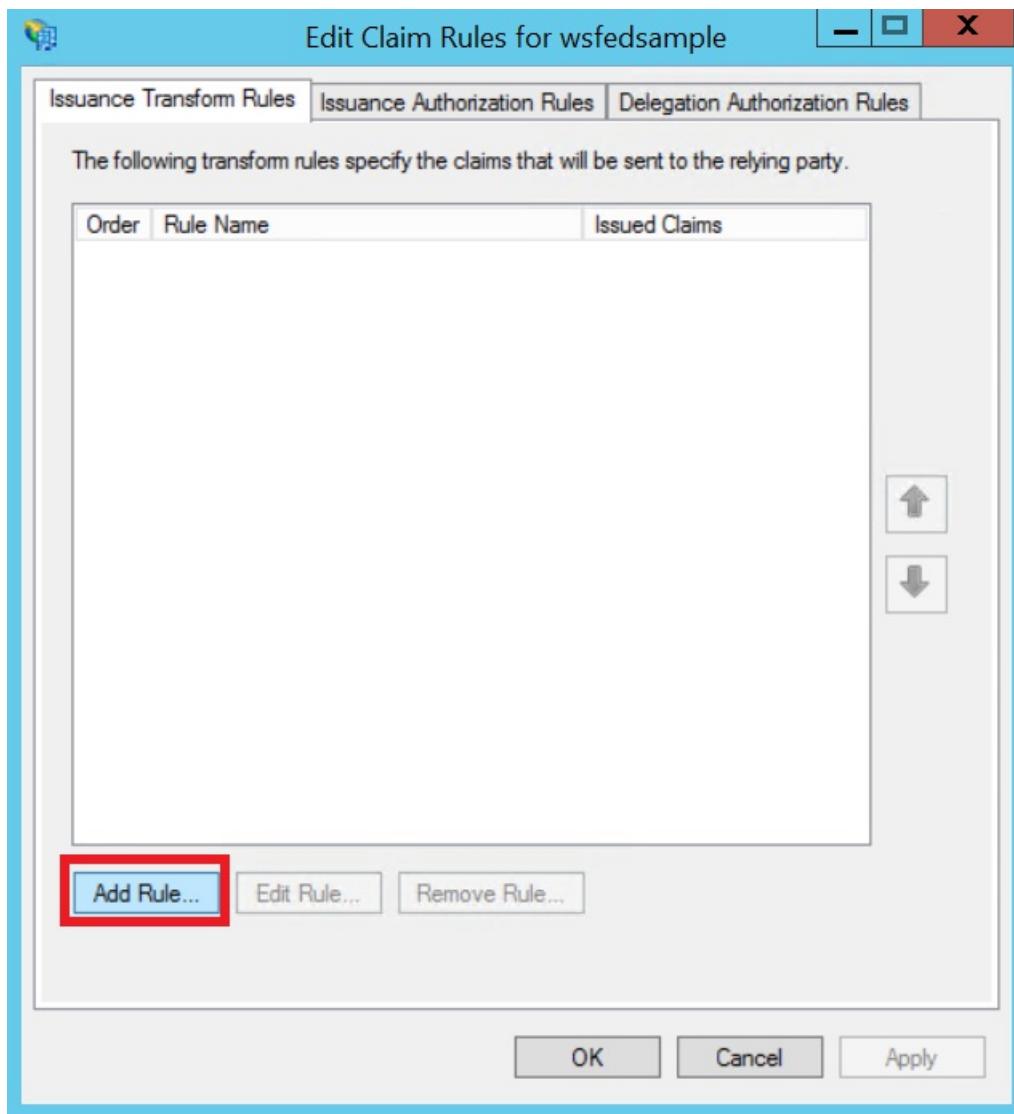
Example: https://www.contoso.com/adfs/ls/

< Previous Next > Cancel

#### NOTE

Isso deve ser uma URL HTTPS. O IIS Express pode fornecer um certificado autoassinado, ao hospedar o aplicativo durante o desenvolvimento. O kestrel requer configuração manual do certificado. Consulte a [documentação do Kestrel](#) para obter mais detalhes.

- Clique em **próxima** através do restante do assistente e **fechar** no final.
- ASP.NET Core Identity exige que um **ID de nome** de declaração. Adicione um dos **editar regras de declaração** caixa de diálogo:



- No **Adicionar Assistente de regra de declaração de transformação**, deixe o padrão **enviar atributos LDAP como declarações** modelo selecionado e clique em **próxima**. Adicionar um mapeamento de regra a **SAM-Account-Name** atributo LDAP para o **ID de nome** declaração de saída:

Add Transform Claim Rule Wizard

### Configure Rule

**Steps**

- Choose Rule Type
- Configure Claim Rule

You can configure this rule to send the values of LDAP attributes as claims. Select an attribute store from which to extract LDAP attributes. Specify how the attributes will map to the outgoing claim types that will be issued from the rule.

Claim rule name:

Rule template: Send LDAP Attributes as Claims

Attribute store:

Mapping of LDAP attributes to outgoing claim types:

LDAP Attribute (Select or type to add more)	Outgoing Claim Type (Select or type to add more)
SAM-Account-Name	Name ID
*	*

< Previous            Cancel

- Clique em **terminar** > **Okey** no **editar regras de declaração** janela.

### Azure Active Directory

- Navegue até a folha de registros de aplicativo do locatário do AAD. Clique em **novo registro de aplicativo**:

Home > wsfedsample - App registrations

wsfedsample - App registrations

Azure Active Directory

**New application registration**

To view and manage your registrations for converged applications, please visit the [Microsoft Application Console](#).

Search by name or AppID    My apps

DISPLAY NAME	APPLICATION TYPE	APPLICATION ID
No results.		

Overview    Quick start    Endpoints    Troubleshoot

**MANAGE**

- Users
- Groups
- Enterprise applications
- Devices
- App registrations**
- Application proxy

- Insira um nome para o registro do aplicativo. Isso não é importante para o aplicativo ASP.NET Core.
- Insira a URL que o aplicativo escuta como o **URL de logon**:

Create

\* Name i

wsfedsample

Application type i

Web app / API

\* Sign-on URL i

https://localhost:44307

**Create**

- Clique em **pontos de extremidade** e observe o **documento de metadados de Federação URL**. Esse é o middleware de Web Services Federation MetadataAddress :

New application registration Troubleshoot

To view and manage your registrations for converged applications, please visit the Microsoft Application Console.

Search by name or Appld All apps

DISPLAY NAME	APPLICATION TYPE	APPLICATION ID
ws fedsample	Web app / API	adc50fba-9ac5-43cd-9d40-bb7...

FEDERATION METADATA DOCUMENT  
<https://login.microsoftonline.com/39afe...>

WS-FEDERATION SIGN-ON ENDPOINT  
<https://login.microsoftonline.com/39af...>

SAML-P SIGN-ON ENDPOINT

- Navegue até o novo registro do aplicativo. Clique em **as configurações > as propriedades** e anote o **URI da ID do aplicativo**. Esse é o middleware de Web Services Federation Wtrealm :

## Adicionar o WS-Federation como um provedor de logon externo para ASP.NET Core Identity

- Adicionar uma dependência no [Microsoft.AspNetCore.Authentication.WsFederation](#) ao projeto.
- Adicionar o WS-Federation para `Startup.ConfigureServices` :

```

services.AddIdentity<ApplicationUser, IdentityRole>()
    .AddEntityFrameworkStores<ApplicationContext>()
    .AddDefaultTokenProviders();

services.AddAuthentication()
    .AddWsFederation(options =>
{
    // MetadataAddress represents the Active Directory instance used to authenticate users.
    options.MetadataAddress = "https://<ADFS FQDN or AAD tenant>/FederationMetadata/2007-
    06/FederationMetadata.xml";

    // Wtrealm is the app's identifier in the Active Directory instance.
    // For ADFS, use the relying party's identifier, its WS-Federation Passive protocol URL:
    options.Wtrealm = "https://localhost:44307/";

    // For AAD, use the App ID URI from the app registration's Properties blade:
    options.Wtrealm = "https://wsfedsample.onmicrosoft.com/bf0e7e6d-056e-4e37-b9a6-2c36797b9f01";
});

services.AddMvc()
// ...

```

A chamada para `AddIdentity` define as configurações de esquema padrão. O `AddAuthentication(String)` conjuntos de sobrepor os `DefaultScheme` propriedade. O `AddAuthentication(ação<AuthenticationOptions>)` sobrecarga permite configurar opções de autenticação, que podem ser usadas para definir os esquemas de autenticação padrão para finalidades diferentes. As chamadas subsequentes para `AddAuthentication` substituição configurada anteriormente `AuthenticationOptions` propriedades.

`AuthenticationBuilder` métodos de extensão que registra um manipulador de autenticação podem ser chamados apenas uma vez por esquema de autenticação. Há sobrecargas que permitem configurar as propriedades de esquema, o nome de esquema e nome de exibição.

### Faça logon com o WS-Federation

Navegue até o aplicativo e clique o **faça logon no** link no cabeçalho da barra de navegação. Há uma opção para fazer logon com WsFederation:



## Log in

Use a local account to log in.

Email

Password

Remember me?

WsFederation

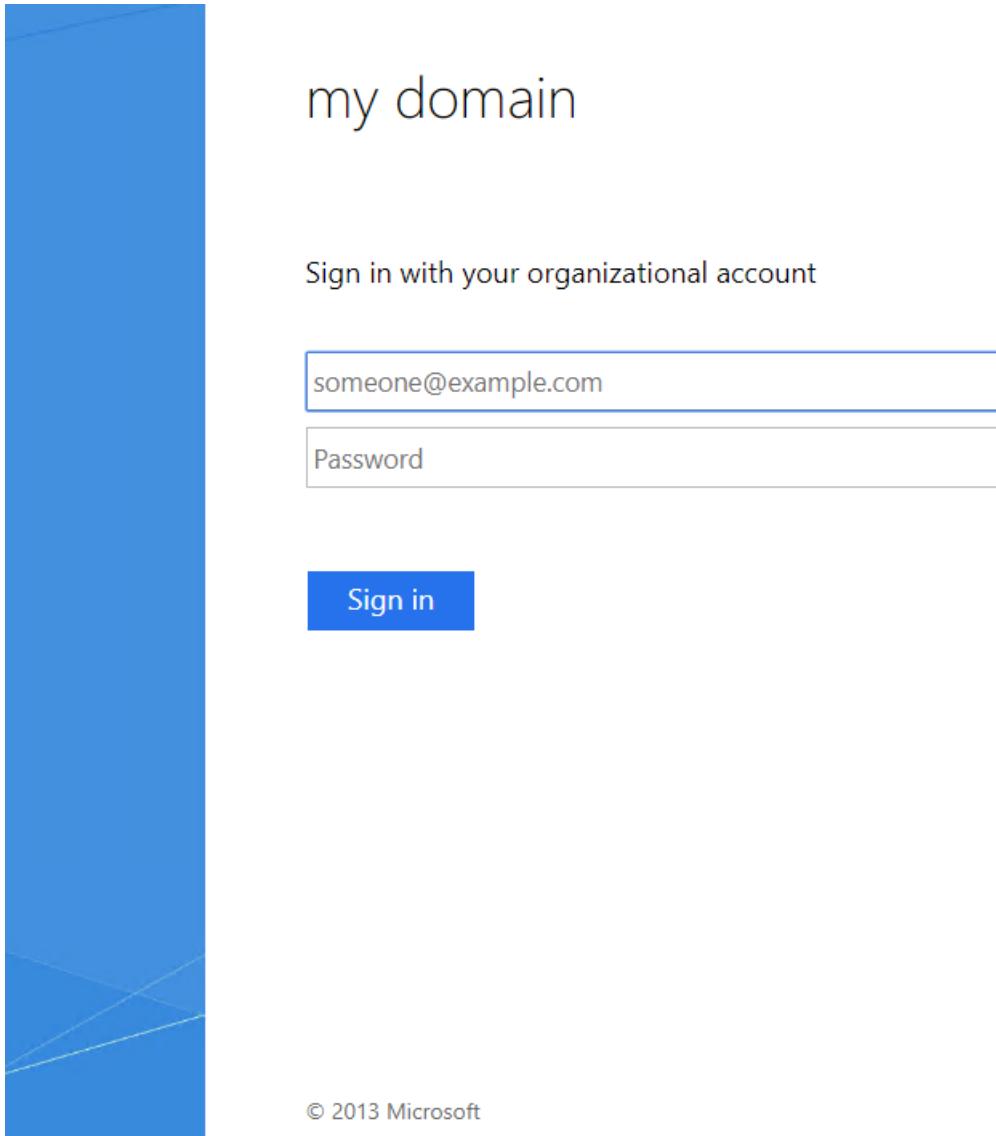
[Log in using your WsFederation account](#)

[Log in](#)

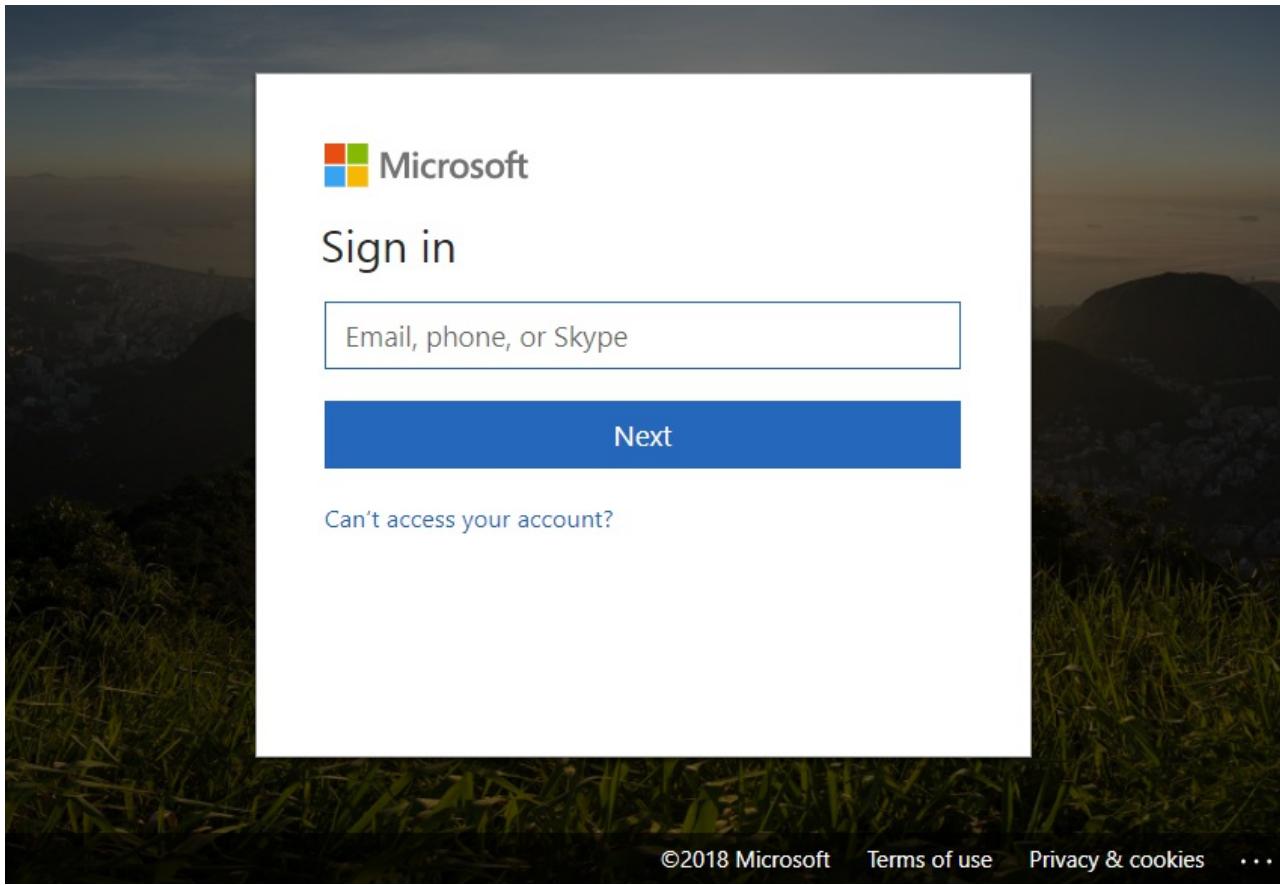
[Forgot your password?](#)

[Register as a new user](#)

Com o ADFS como o provedor, o botão redireciona para uma página de entrada do AD FS:



Com o Azure Active Directory como o provedor, o botão redireciona para uma página de logon do AAD:



Um entrar com êxito para um novo usuário redireciona para a página de registro de usuário do aplicativo:



## Register

Associate your WsFederation account.

You've successfully authenticated with **WsFederation**. Please enter an email address for this site below and click the Register button to finish logging in.

Email

## Usar o WS-Federation, sem o ASP.NET Core Identity

O middleware de WS-Federation pode ser usado sem o Identity. Por exemplo:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddAuthentication(sharedOptions =>
    {
        sharedOptions.DefaultScheme = CookieAuthenticationDefaults.AuthenticationScheme;
        sharedOptions.DefaultSignInScheme = CookieAuthenticationDefaults.AuthenticationScheme;
        sharedOptions.DefaultChallengeScheme = WsFederationDefaults.AuthenticationScheme;
    })
    .AddWsFederation(options =>
    {
        options.Wtrealm = Configuration["wsfed:realm"];
        options.MetadataAddress = Configuration["wsfed:metadata"];
    })
    .AddCookie();
}

public void Configure(IApplicationBuilder app)
{
    app.UseAuthentication();
    // ...
}
```

# Confirmação de conta e de recuperação de senha no ASP.NET Core

13/02/2019 • 15 minutes to read • [Edit Online](#)

Ver [esse arquivo PDF](#) para o ASP.NET Core 1.1 e a versão 2.1.

Por [Rick Anderson](#) e [Joe Audette](#)

Este tutorial mostra como criar um aplicativo ASP.NET Core com a redefinição de senha e de confirmação de email. Este tutorial é **não** um tópico de início. Você deve estar familiarizado com:

- [ASP.NET Core](#)
- [Autenticação](#)
- [Entity Framework Core](#)

## Pré-requisitos

[SDK do .NET Core 2.1 ou posteriores](#)

## Criar um aplicativo web e o scaffolding de identidade

- [Visual Studio](#)
- [CLI do .NET Core](#)
- No Visual Studio, crie um novo **aplicativo Web** projeto chamado **WebPWrecover**.
- Selecione **ASP.NET Core 2.1**.
- Mantenha o padrão **autenticação** definido como **sem autenticação**. Autenticação é adicionada na próxima etapa.

Na próxima etapa:

- Definir a página de layout `~/Pages/Shared/_Layout.cshtml`
- Selecionar *conta/registo*
- Criar um novo **classe de contexto de dados**

Siga as instruções em [habilitar a autenticação](#):

- Adicionar `app.UseAuthentication();` para `Startup.Configure`
- Adicionar `<partial name=" _LoginPartial" />` ao arquivo de layout.

## Novo registro de usuário de teste

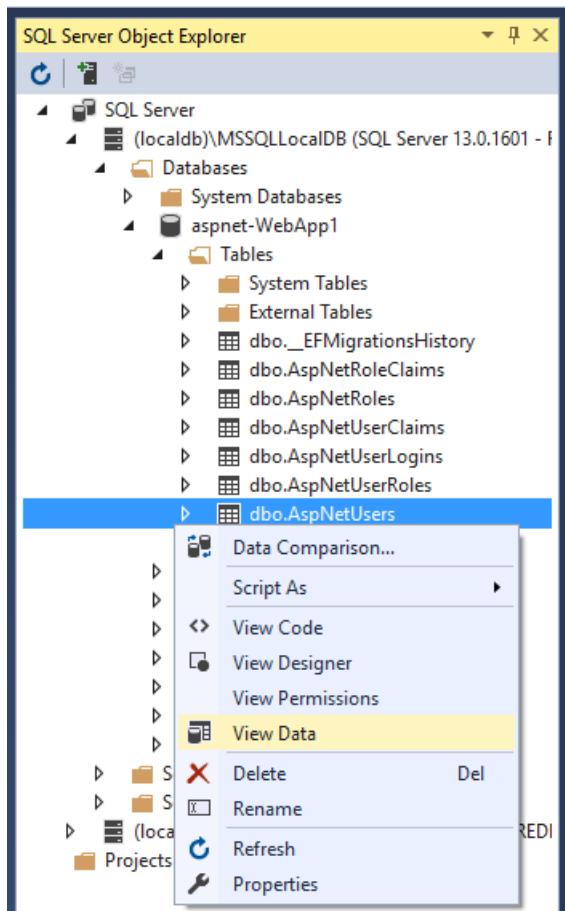
Execute o aplicativo, selecione a **registrar** vincular e registrar um usuário. Neste ponto, a validação apenas no email é com o **EmailAddress** atributo. Depois de enviar o registro, você está conectado ao aplicativo.

Posteriormente no tutorial, o código é atualizado para que novos usuários poderão entrar depois que o email é validado.

### Exibir o banco de dados de identidade

- [Visual Studio](#)
- [CLI do .NET Core](#)

- Dos **modo de exibição** menu, selecione **Pesquisador de objetos do SQL Server (SSOX)**.
- Navegue até **(localdb) MSSQLLocalDB (SQL Server 13)**. Clique duas vezes em **dbo. AspNetUsers > exibir dados**:



Observe a tabela `EmailConfirmed` campo é `False`.

Você talvez queira usar este email novamente na próxima etapa quando o aplicativo envia um email de confirmação. Clique com botão direito na linha e selecione **excluir**. Excluir o alias de email torna mais fácil nas etapas a seguir.

## Exigir email de confirmação

É uma prática recomendada para confirmar o email de um novo registro de usuário. Ajuda a confirmação para verificar se eles não estiver representando alguém de email (ou seja, eles ainda não registrados com o email de outra pessoa). Suponha que você tinha um fórum de discussão e quisesse impedir "yli@example.com" de registrar-se como "nolivetto@contoso.com". Sem email de confirmação "nolivetto@contoso.com" pode receber emails indesejados de seu aplicativo. Suponha que o usuário registrado accidentalmente como "ylo@example.com" e ainda não tenha notado o erro de ortografia de "yli". Eles não seria capazes de usar a recuperação de senha porque o aplicativo não tiver seu email correto. Email de confirmação oferece proteção limitada contra bots. Email de confirmação não fornece proteção contra usuários mal-intencionados com muitas contas de email.

Geralmente você deseja impedir que novos usuários incluem dados em seu site até que eles tenham um email confirmado.

Atualização `Areas/Identity/IdentityHostingStartup.cs` para exigir um email confirmado:

```

public class IdentityHostingStartup : IHostingStartup
{
    public void Configure(IWebHostBuilder builder)
    {
        builder.ConfigureServices((context, services) => {
            services.AddDbContext<WebPWrecoverContext>(options =>
                options.UseSqlServer(
                    context.Configuration.GetConnectionString("WebPWrecoverContextConnection")));
            services.AddDefaultIdentity<IdentityUser>(config =>
            {
                config.SignIn.RequireConfirmedEmail = true;
            })
            .AddEntityFrameworkStores<WebPWrecoverContext>();
        });
    }
}

```

`config.SignIn.RequireConfirmedEmail = true;` impede que os usuários registrados entrar até que o email seja confirmado.

### Configurar o provedor de email

Neste tutorial [SendGrid](#) é usado para enviar email. Você precisa de uma conta do SendGrid e uma chave para enviar email. Você pode usar outros provedores de email. ASP.NET Core 2.x inclui `System.Net.Mail`, que permite que você enviar um email de seu aplicativo. É recomendável que usar o SendGrid ou outro serviço de email para enviar email. SMTP é difícil proteger e configurado corretamente.

Crie uma classe para buscar a chave de email seguro. Para este exemplo, crie `Services/AuthMessageSenderOptions.cs`:

```

public class AuthMessageSenderOptions
{
    public string SendGridUser { get; set; }
    public string SendGridKey { get; set; }
}

```

### Configurar segredos de usuário do SendGrid

Adicionar um único `<UserSecretsId>` de valor para o `<PropertyGroup>` elemento do arquivo de projeto:

```

<Project Sdk="Microsoft.NET.Sdk.Web">

    <PropertyGroup>
        <TargetFramework>netcoreapp2.1</TargetFramework>
        <UserSecretsId>aspnet-WebPWrecover-1234</UserSecretsId>
    </PropertyGroup>

    <ItemGroup>
        <PackageReference Include="Microsoft.AspNetCore.App" />
        <PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Design" Version="2.1.1" />
        <PackageReference Include="SendGrid" Version="9.9.0" />
    </ItemGroup>

</Project>

```

Defina as `SendGridUser` e `SendGridKey` com o [ferramenta secret manager](#). Por exemplo:

```

C:/WebApp>dotnet user-secrets set SendGridUser RickAndMSFT
info: Successfully saved SendGridUser = RickAndMSFT to the secret store.

```

No Windows, o Secret Manager armazena os pares de chaves/valor em uma *Secrets* arquivo no `%APPDATA%/Microsoft/UserSecrets/<WebAppName-userSecretsId>` directory.

O conteúdo a *Secrets* arquivo não são criptografadas. O *Secrets* arquivo é mostrado abaixo (o `SendGridKey` valor tiver sido removido.)

```
{  
    "SendGridUser": "RickAndMSFT",  
    "SendGridKey": "<key removed>"  
}
```

Para obter mais informações, consulte o [padrão de opções e configuração](#).

## Instalar o SendGrid

Este tutorial mostra como adicionar notificações de email por meio [SendGrid](#), mas você pode enviar emails usando SMTP e outros mecanismos.

Instalar o `SendGrid` pacote do NuGet:

- [Visual Studio](#)
- [CLI do .NET Core](#)

No Console do Gerenciador de pacotes, digite o seguinte comando:

```
Install-Package SendGrid
```

Ver [inicie gratuitamente com o SendGrid](#) para se registrar para uma conta gratuita do SendGrid.

## Implementar IEmailSender

Para implementar `IEmailSender`, crie *Services/EmailSender.cs* com um código semelhante ao seguinte:

```

using Microsoft.AspNetCore.Identity.UI.Services;
using Microsoft.Extensions.Options;
using SendGrid;
using SendGrid.Helpers.Mail;
using System.Threading.Tasks;

namespace WebPWrecover.Services
{
    public class EmailSender : IEmailSender
    {
        public EmailSender(IOptions<AuthMessageSenderOptions> optionsAccessor)
        {
            Options = optionsAccessor.Value;
        }

        public AuthMessageSenderOptions Options { get; } //set only via Secret Manager

        public Task SendEmailAsync(string email, string subject, string message)
        {
            return Execute(Options.SendGridKey, subject, message, email);
        }

        public Task Execute(string apiKey, string subject, string message, string email)
        {
            var client = new SendGridClient(apiKey);
            var msg = new SendGridMessage()
            {
                From = new EmailAddress("Joe@contoso.com", "Joe Smith"),
                Subject = subject,
                PlainTextContent = message,
                HtmlContent = message
            };
            msg.AddTo(new EmailAddress(email));

            // Disable click tracking.
            // See https://sendgrid.com/docs/User_Guide/Settings/tracking.html
            msg.SetClickTracking(false, false);

            return client.SendEmailAsync(msg);
        }
    }
}

```

## Configurar a inicialização para dar suporte a email

Adicione o seguinte código para o `ConfigureServices` método na `Startup.cs` arquivo:

- Adicionar `EmailSender` como um serviço transitório.
- Registrar o `AuthMessageSenderOptions` instância de configuração.

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    // requires
    // using Microsoft.AspNetCore.Identity.UI.Services;
    // using WebPWrecover.Services;
    services.AddTransient<IEmailSender, EmailSender>();
    services.Configure<AuthMessageSenderOptions>(Configuration);
}
```

## Habilitar a recuperação de confirmação e a senha da conta

O modelo tem o código para recuperação de confirmação e a senha da conta. Localizar o `OnPostAsync` método no `Areas/Identity/Pages/Account/Register.cshtml.cs`.

Evitar que usuários recém-registrados seja registrada automaticamente comentando a linha a seguir:

```
await _signInManager.SignInAsync(user, isPersistent: false);
```

O método completo é mostrado com a linha alterada realçada:

```

public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl = returnUrl ?? Url.Content("~/");
    if (ModelState.IsValid)
    {
        var user = new IdentityUser { UserName = Input.Email, Email = Input.Email };
        var result = await _userManager.CreateAsync(user, Input.Password);
        if (result.Succeeded)
        {
            _logger.LogInformation("User created a new account with password.");
            var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
            var callbackUrl = Url.Page(
                "/Account/ConfirmEmail",
                pageHandler: null,
                values: new { userId = user.Id, code = code },
                protocol: Request.Scheme);

            await _emailSender.SendEmailAsync(Input.Email, "Confirm your email",
                $"Please confirm your account by <a href='{HtmlEncoder.Default.Encode(callbackUrl)}'>clicking here</a>.");
        }
        // await _signInManager.SignInAsync(user, isPersistent: false);
        return LocalRedirect(returnUrl);
    }
    foreach (var error in result.Errors)
    {
        ModelState.AddModelError(string.Empty, error.Description);
    }
}

// If we got this far, something failed, redisplay form
return Page();
}

```

## Registrar, confirme se o email e redefinição de senha

Executar o aplicativo web e testar o fluxo de recuperação de senha e confirmação de conta.

- Execute o aplicativo e registrar um novo usuário

Register - WebApplication1

localhost:10801/Account/Register

WebApplication1 Home About Contact Register Log in

# Register.

Create a new account.

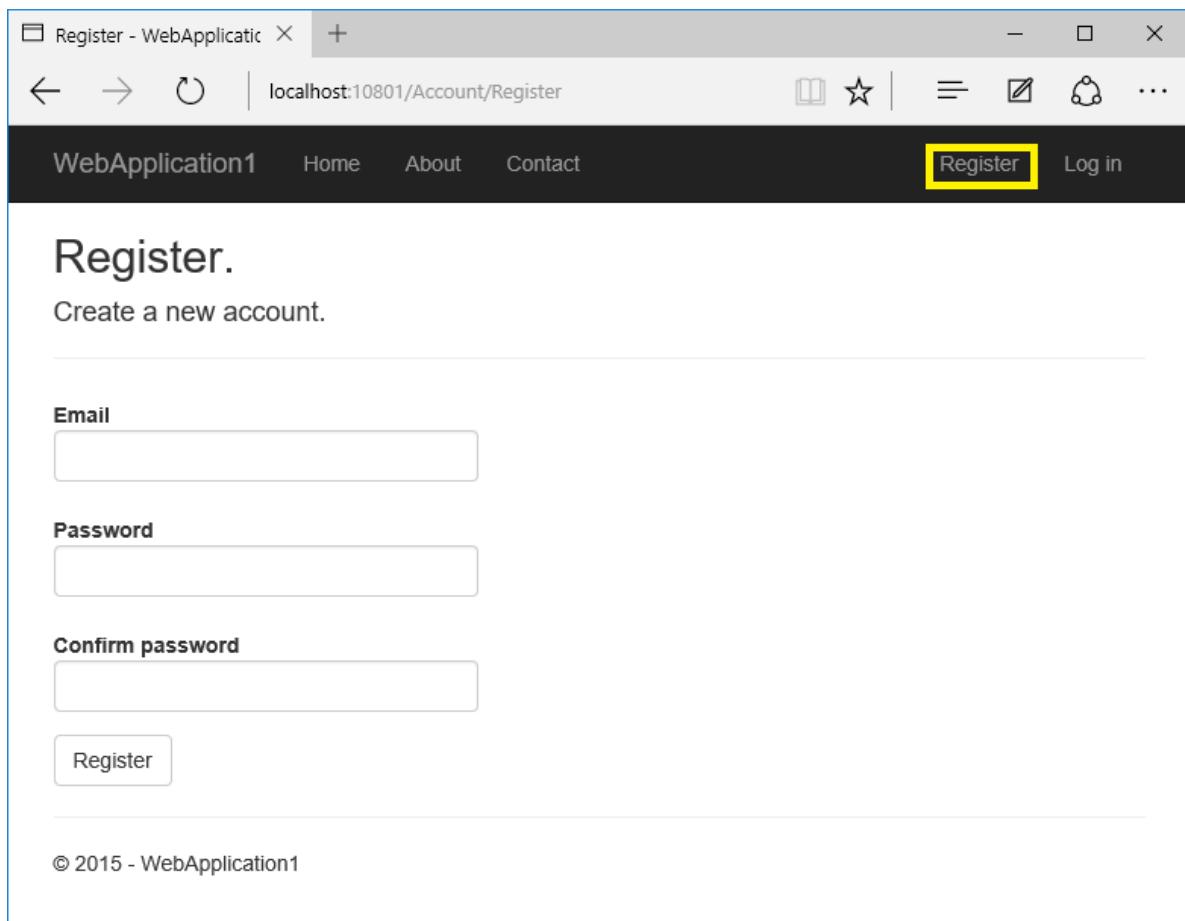
Email

Password

Confirm password

Register

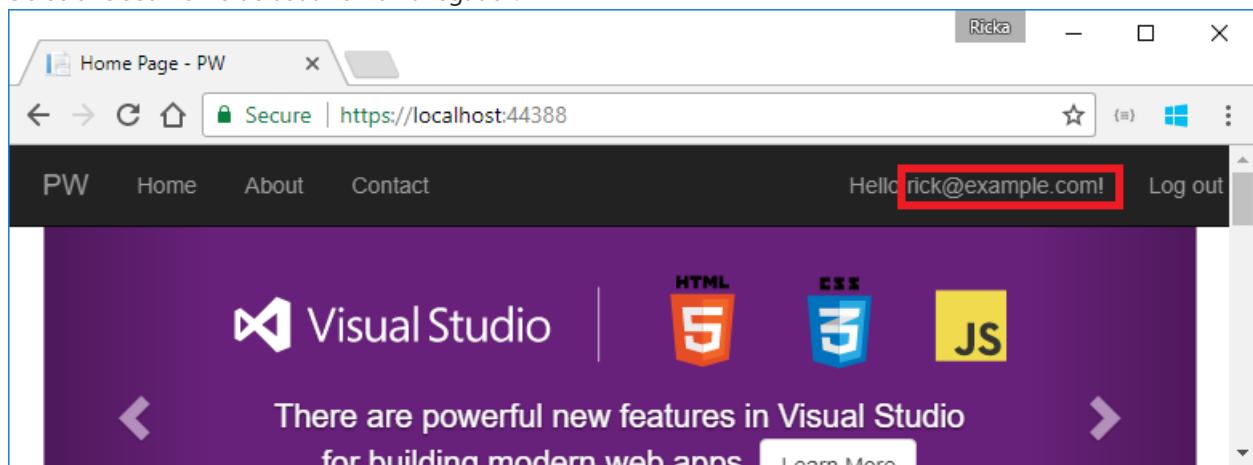
© 2015 - WebApplication1



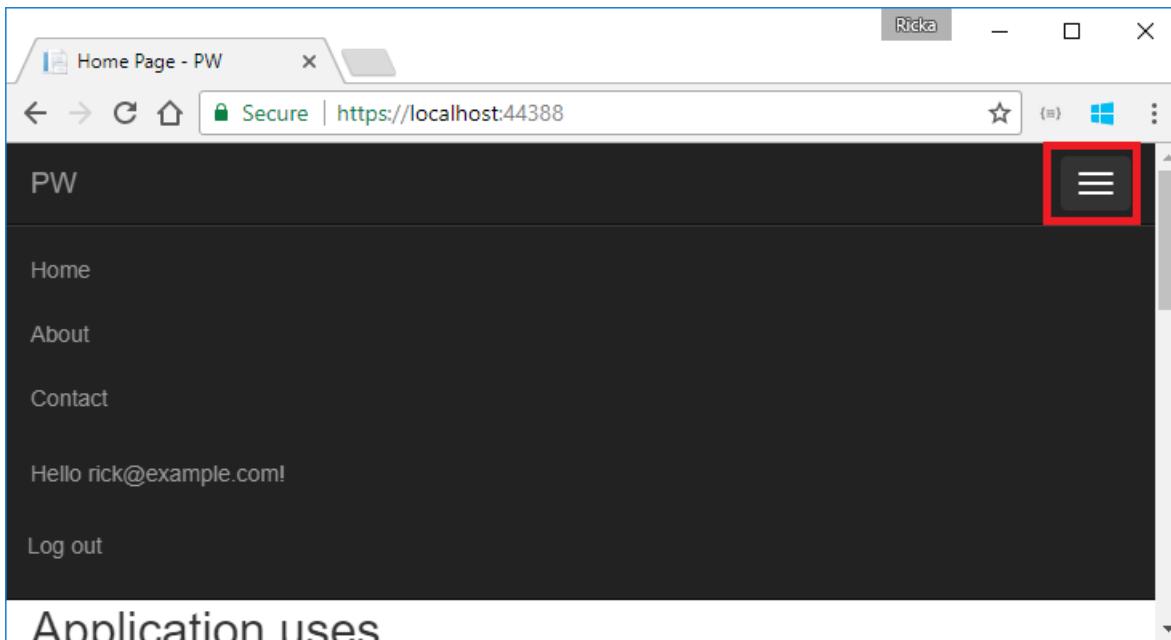
- Verifique seu email para o link de confirmação de conta. Ver [depurar email](#) se você não receber o email.
- Clique no link para confirmar seu email.
- Entrar com seu email e senha.
- Saia do serviço.

### Exibir a página Gerenciar

Selecione seu nome de usuário no navegador:



Talvez seja necessário expandir a barra de navegação para ver o nome de usuário.



A página Gerenciar é exibida com o **perfil** guia selecionada. O **Email** mostra uma caixa de seleção que indica o email foi confirmada.

#### Teste a redefinição de senha

- Se você estiver conectado, selecione **Logout**.
- Selecione o **login** link e selecione o **esqueceu sua senha?** link.
- Insira o email usado para registrar a conta.
- Um email com um link para redefinir sua senha é enviado. Verifique seu email e clique no link para redefinir sua senha. Depois que sua senha foi redefinida com êxito, você pode entrar com seu email e a nova senha.

#### Depurar o email

Se você não é possível obter o trabalho de email:

- Defina um ponto de interrupção no `EmailSender.Execute` para verificar `SendGridClient.SendEmailAsync` é chamado.
- Criar uma [aplicativo de console para enviar email](#) usando um código semelhante ao `EmailSender.Execute`.
- Examine os [atividade de Email](#) página.
- Verifique sua pasta de spam.
- Tente outro alias de email em outro provedor de email (Microsoft, Yahoo, Gmail, etc.)
- Tente enviar para contas de email diferente.

**Uma prática recomendada de segurança** é **não** use segredos de produção em desenvolvimento e teste. Se você publicar o aplicativo no Azure, você pode definir os segredos do SendGrid como configurações de aplicativo no portal do aplicativo Web do Azure. Configurar o sistema de configuração para ler as chaves de variáveis de ambiente.

## Combine as contas de logon social e local

Para concluir esta seção, você deve primeiro habilitar um provedor de autenticação externa. Ver [Facebook](#), [Google](#) e a [autenticação de provedor externo](#).

Você pode combinar as contas locais e sociais clicando no link seu email. Na sequência a seguir, "RickAndMSFT@gmail.com" é criado como um logon local; no entanto, você pode criar a conta como um logon social primeiro e adicionar um logon local.

The screenshot shows a web browser window with the title "Home Page - Web1". The address bar displays "localhost:1234". The top navigation bar includes links for "Web1", "Home", "About", "Contact", and a user account section with "Hello rickandmsft@gmail.com!" and "Log off". Below the navigation, there's a banner for "ASP.NET 5" with text about running on "Windows", "Linux", and "OSX", and a "Learn More" button. A section titled "Application uses" lists items like "Sample pages using ASP.NET 5 (MVC 6)", "Gulp and Bower for managing client-side resources", and "Theming using Bootstrap". Another section titled "New concepts" has a link to "Conceptual overview of ASP.NET 5".

Clique no **gerenciar** link. Observe externo 0 (logons sociais) associado a essa conta.

The screenshot shows a "Manage your account" page from the application. The top navigation bar is identical to the previous one. The main content area is titled "Manage your account." and contains a "Change your account settings" section. It shows fields for "Password" (with a "[Change]" link), "External Logins" (with a "0 [Manage]" link highlighted with a red box), "Phone Number" (with a descriptive text about two-factor authentication), and "Two-Factor Authentic..." (with a note about no providers configured). At the bottom, there's a copyright notice: "© 2015 - Web1".

Clique no link para outro serviço de logon e aceitar as solicitações do aplicativo. Na imagem a seguir, o Facebook é o provedor de autenticação externa:

The screenshot shows the "Manage your external logins" page. The top navigation bar is the same. The main content area is titled "Manage your external logins." and shows a "Registered Logins" section with a single entry for "Facebook".

As duas contas foram combinadas. É possível entrar com qualquer uma das contas. Convém que os usuários adicionem contas locais no caso de seu serviço de autenticação de logon social está inoperante ou, mais provavelmente eles tiver perdido o acesso à sua conta social.

## Habilitar confirmação de conta depois que um site tem usuários

Habilitando a confirmação de conta em um site com usuários bloqueia todos os usuários existentes. Os usuários existentes estão bloqueados porque suas contas não são confirmadas. Para contornar o bloqueio de usuário existente, use uma das seguintes abordagens:

- Atualize o banco de dados para marcar todos os usuários existentes como sendo confirmada.
- Confirme se os usuários existentes. Por exemplo, envio em lote-emails com links de confirmação.

# Habilitar a geração de código QR TOTP para aplicativos de autenticador no ASP.NET Core

26/01/2019 • 5 minutes to read • [Edit Online](#)

Códigos QR exigem o ASP.NET Core 2.0 ou posterior.

ASP.NET Core é fornecido com suporte para aplicativos de autenticador para autenticação individual. Dois aplicativos de autenticador 2FA (autenticação) fator, usando uma baseada em tempo avulso senha algoritmo TOTP (), são o setor de abordagem 2fa recomendado. 2FA usar TOTP é preferencial para SMS 2FA. Um aplicativo autenticador fornece um código de 6 a 8 dígitos que os usuários devem inserir depois de confirmar seu nome de usuário e senha. Normalmente, um aplicativo autenticador é instalado em um Smartphone.

Os modelos de aplicativo web ASP.NET Core autenticadores de suporte, mas não fornecem suporte para a geração de QRCode. Geradores de QRCode facilitam a configuração do 2FA. Este documento o orientará durante a adição [código QR](#) geração para a página de configuração 2FA.

Autenticação de dois fatores não acontece usando um provedor de autenticação externa, como [Google](#) ou [Facebook](#). Logons externos são protegidos por qualquer mecanismo que fornece o provedor de logon externo. Considere, por exemplo, o [Microsoft](#) provedor de autenticação requer uma chave de hardware ou outra abordagem 2FA. Se os modelos padrão impõem 2FA "local", em seguida, os usuários seriam necessários para atender às duas abordagens 2FA, que não é um cenário de uso geral.

## Adicionar códigos QR para a página de configuração 2FA

Usam estas instruções `qrcode.js` do <https://davidshimjs.github.io/qrcodejs/> repositório.

- Baixe o [biblioteca de javascript qrcode.js](#) para o `wwwroot\lib` pasta em seu projeto.
- Siga as instruções em [identidade de Scaffold](#) para gerar `/Areas/Identity/Pages/Account/Manage/EnableAuthenticator.cshtml`.
- Na `/Areas/Identity/Pages/Account/Manage/EnableAuthenticator.cshtml`, localize o `Scripts` seção no final do arquivo:
- Na `Pages/Account/Manage/EnableAuthenticator.cshtml` (páginas do Razor) ou `Views/Manage/EnableAuthenticator.cshtml` (MVC), localize o `Scripts` seção no final do arquivo:

```
@section Scripts {
    @await Html.PartialAsync("_ValidationScriptsPartial")
}
```

- Atualizar o `Scripts` seção para adicionar uma referência para o `qrcodejs` biblioteca que você adicionou e uma chamada para gerar o código QR. Ele deve ser da seguinte maneira:

```

@section Scripts {
    @await Html.PartialAsync("_ValidationScriptsPartial")

    <script type="text/javascript" src="~/lib/qrcode.js"></script>
    <script type="text/javascript">
        new QRCode(document.getElementById("qrCode"),
        {
            text: "@Html.Raw(Model.AuthenticatorUri)",
            width: 150,
            height: 150
        });
    </script>
}

```

- Exclua o parágrafo que links para essas instruções.

Executar seu aplicativo e certifique-se de que você pode digitalizar o código QR e validar o código que comprova o autenticador.

## Alterar o nome do site em que o código QR

O nome do site em que o código QR é obtido do nome do projeto que você escolher ao criar inicialmente o seu projeto. Você pode alterá-lo procurando os `GenerateQrCodeUri(string email, string unformattedKey)` método na `/Areas/Identity/Pages/Account/Manage/EnableAuthenticator.cshtml`.

O nome do site em que o código QR é obtido do nome do projeto que você escolher ao criar inicialmente o seu projeto. Você pode alterá-lo procurando a `GenerateQrCodeUri(string email, string unformattedKey)` método na `Pages/Account/Manage/EnableAuthenticator.cshtml.cs` arquivo (páginas do Razor) ou o `Controllers/ManageController.cs` arquivo (MVC).

O código padrão do modelo será semelhante ao seguinte:

```

private string GenerateQrCodeUri(string email, string unformattedKey)
{
    return string.Format(
        AuthenticatorUriFormat,
        _urlEncoder.Encode("Razor Pages"),
        _urlEncoder.Encode(email),
        unformattedKey);
}

```

O segundo parâmetro na chamada para `string.Format` é o nome do site, obtido do nome da solução. Ele pode ser alterado para qualquer valor, mas ele sempre deve ser codificado por URL.

## Usando uma biblioteca diferente do código QR

Você pode substituir a biblioteca de código QR com sua biblioteca preferencial. O HTML não contém um `qrCode` fornece de sua biblioteca de elemento no qual você pode colocar um código QR por qualquer mecanismo.

A URL formatada corretamente para o código QR está disponível na:

- `AuthenticatorUri` propriedade do modelo.
- `data-url` propriedade no `qrCodeData` elemento.

## TOTP cliente e servidor distorção de tempo

Autenticação de TOTP (senha de uso único baseados em tempo) depende do dispositivo o servidor e o autenticador, tendo a hora correta. Tokens duram por 30 segundos. Se os logons de 2FA TOTP estiverem

falhando, verifique se a hora do servidor é precisos e preferencialmente sincronizado a um serviço NTP preciso.

# Autenticação de dois fatores com SMS no ASP.NET Core

08/01/2019 • 8 minutes to read • [Edit Online](#)

Por [Rick Anderson](#) e [suíço desenvolvedores](#)

## WARNING

Dois aplicativos de autenticador 2FA (autenticação) fator, usando uma baseada em tempo avulso senha algoritmo TOTP (), são o setor de abordagem 2fa recomendado. 2FA usar TOTP é preferencial para SMS 2FA. Para obter mais informações, consulte [geração de código de QR habilitar TOTP para aplicativos de autenticador no ASP.NET Core para ASP.NET Core 2.0 e versões posteriores](#).

Este tutorial mostra como configurar a autenticação de dois fatores (2FA) usando o SMS. As instruções são fornecidas para [twilio](#) e [ASPSMS](#), mas você pode usar qualquer outro provedor SMS. Recomendamos que você conclua [confirmação de conta e recuperação de senha](#) antes de iniciar este tutorial.

[Exibir ou baixar o código de exemplo.](#) [Como baixar.](#)

## Criar um projeto ASP.NET Core

Criar um novo aplicativo web de ASP.NET Core chamado `Web2FA` com contas de usuário individuais. Siga as instruções em [Impor HTTPS no ASP.NET Core](#) para configurar e exigir HTTPS.

### Criar uma conta do SMS

Criar uma conta SMS, por exemplo, no [twilio](#) ou [ASPSMS](#). Registre as credenciais de autenticação (para o twilio: accountSid e authToken para ASPSMS: Userkey e senha).

### Descobrir as credenciais do provedor de SMS

**Twilio:** Na guia Painel de sua conta do Twilio, copie o **SID da conta** e **token de autenticação**.

**ASPSMS:** Em suas configurações de conta, navegue até **Userkey** e copie-o junto com seus **senha**.

Posteriormente, armazenaremos esses valores com a ferramenta secret manager nas chaves

`SMSAccountIdentification` e `SMSAccountPassword`.

### Especificando SenderID / originador

**Twilio:** Na guia números, copie o Twilio **número de telefone**.

**ASPSMS:** Dentro do Menu originadores de desbloqueio, desbloquear originadores de um ou mais ou escolha um originador alfanumérico (não é suportado por todas as redes).

Posteriormente, armazenaremos esse valor com a ferramenta secret manager na chave do `SMSAccountFrom`.

### Forneça credenciais para o serviço SMS

Vamos usar o [padrão de opções](#) para acessar as configurações de conta e chave de usuário.

- Crie uma classe para buscar a chave segura do SMS. Para este exemplo, o `SMSOptions` classe é criada na `Services/SMSOptions.cs` arquivo.

```

namespace Web2FA.Services
{
    public class SMSOptions
    {
        public string SMSAccountIdentification { get; set; }
        public string SMSAccountPassword { get; set; }
        public string SMSAccountFrom { get; set; }
    }
}

```

Defina as `SMSAccountIdentification`, `SMSAccountPassword` e `SMSAccountFrom` com o [ferramenta secret manager](#). Por exemplo:

```
C:/Web2FA/src/WebApp1>dotnet user-secrets set SMSAccountIdentification 12345
info: Successfully saved SMSAccountIdentification = 12345 to the secret store.
```

- Adicione o pacote NuGet do provedor de SMS. Do Manager Console (PMC) execute:

**Twilio:** `Install-Package Twilio`

**ASPSMS:** `Install-Package ASPSMS`

- Adicione código a `Services/MessageServices.cs` arquivo para habilitar o SMS. Use o Twilio ou seção ASPSMS:

**Twilio:** `![code-csharp]`

**ASPSMS:** `![code-csharp]`

**Configurar a inicialização para usar** `SMSOptions`

Adicione `SMSOptions` ao contêiner de serviço na `ConfigureServices` método na `Startup.cs`:

```

// Add application services.
services.AddTransient<IEmailSender, AuthMessageSender>();
services.AddTransient<ISmsSender, AuthMessageSender>();
services.Configure<SMSOptions>(Configuration);
}

```

### Habilitar a autenticação de dois fatores

Abra o `Views/Manage/Index.cshtml` arquivo de exibição do Razor e remova o comentário caracteres (de modo que nenhuma marcação é commnted out).

## Faça logon com a autenticação de dois fatores

- Execute o aplicativo e registrar um novo usuário

Register - WebApplication1

localhost:44300/

# WebApplication1

## Register.

Create a new account.

Email  
joe@contoso.com

Password  
\*\*\*\*\*

Confirm password  
\*\*\*\*\*

Register

© 2015 - WebApplication1

- Toque em seu nome de usuário que ativa o `Index` método de ação no controlador de gerenciar. Em seguida, toque no número de telefone **adicionar** link.

Manage your account -

localhost:44300/Manage

# WebApplication1

Hello joe@contoso.com! Log off

## Manage your account.

Change your account settings

Password: [ Change ]

External Logins: 0 [ Manage ]

Phone Number: Phone Numbers can used as a second factor of verification in two-factor authentication. See [this article](#) for details on setting up this ASP.NET application to support two-factor authentication using SMS.

Two-Factor Authentic... None [ Add ]

Disabled [Enable](#)

© 2015 - WebApplication1

- Adicionar um número de telefone que receberá o código de verificação e toque **enviar código de verificação**.

Add Phone Number.

Add a phone number.

Phone number

206-555-1234

Send verification code

© 2015 - WebApplication1

- Você receberá uma mensagem de texto com o código de verificação. Insira-o e toque em **enviar**

Verify Phone Number - We...

WebApplication1 Home About Contact Hello joe@contoso.com! Log off

Verify Phone Number.

Add a phone number.

Code

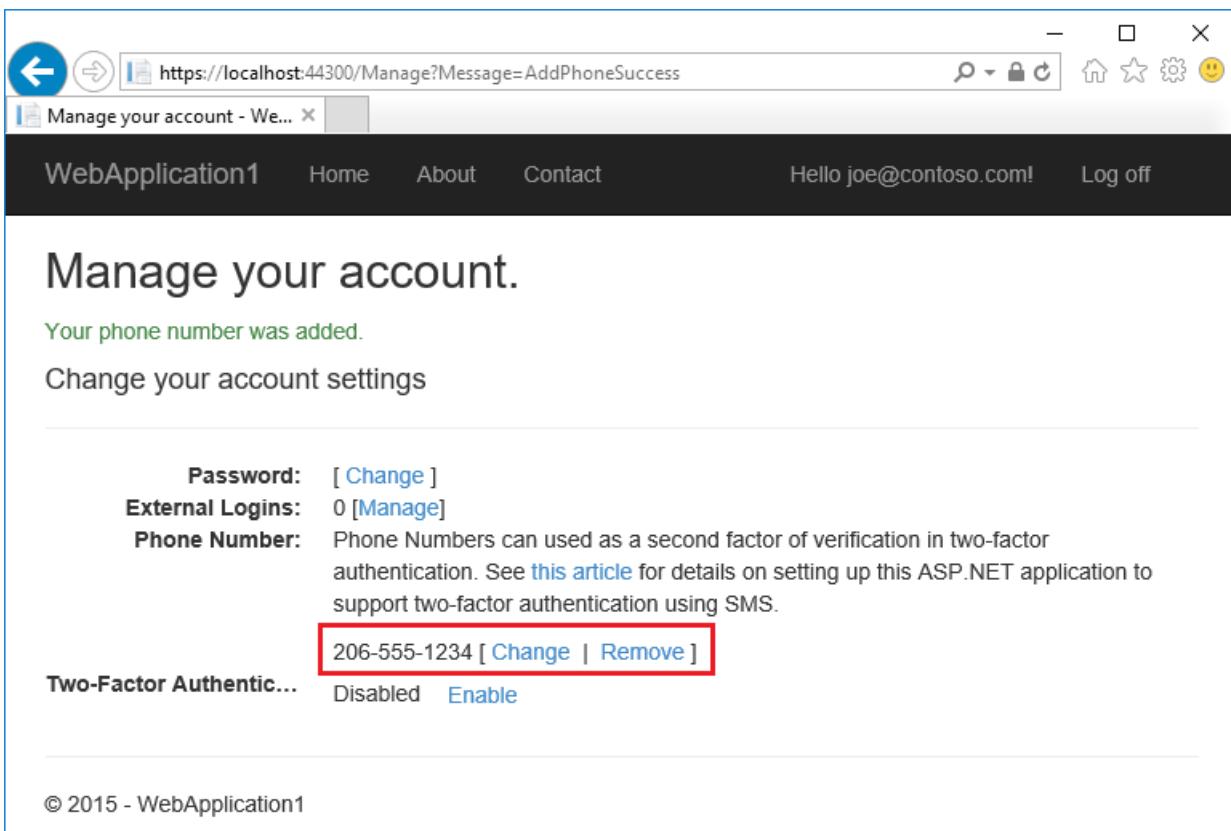
5879

Submit

© 2015 - WebApplication1

Se você não receber uma mensagem de texto, consulte a página de registro do twilio.

- O modo de gerenciar mostra que o número de telefone foi adicionado com êxito.



Manage your account - We... Home About Contact Hello joe@contoso.com! Log off

## Manage your account.

Your phone number was added.

Change your account settings

**Password:** [ [Change](#) ]  
**External Logins:** 0 [\[Manage\]](#)

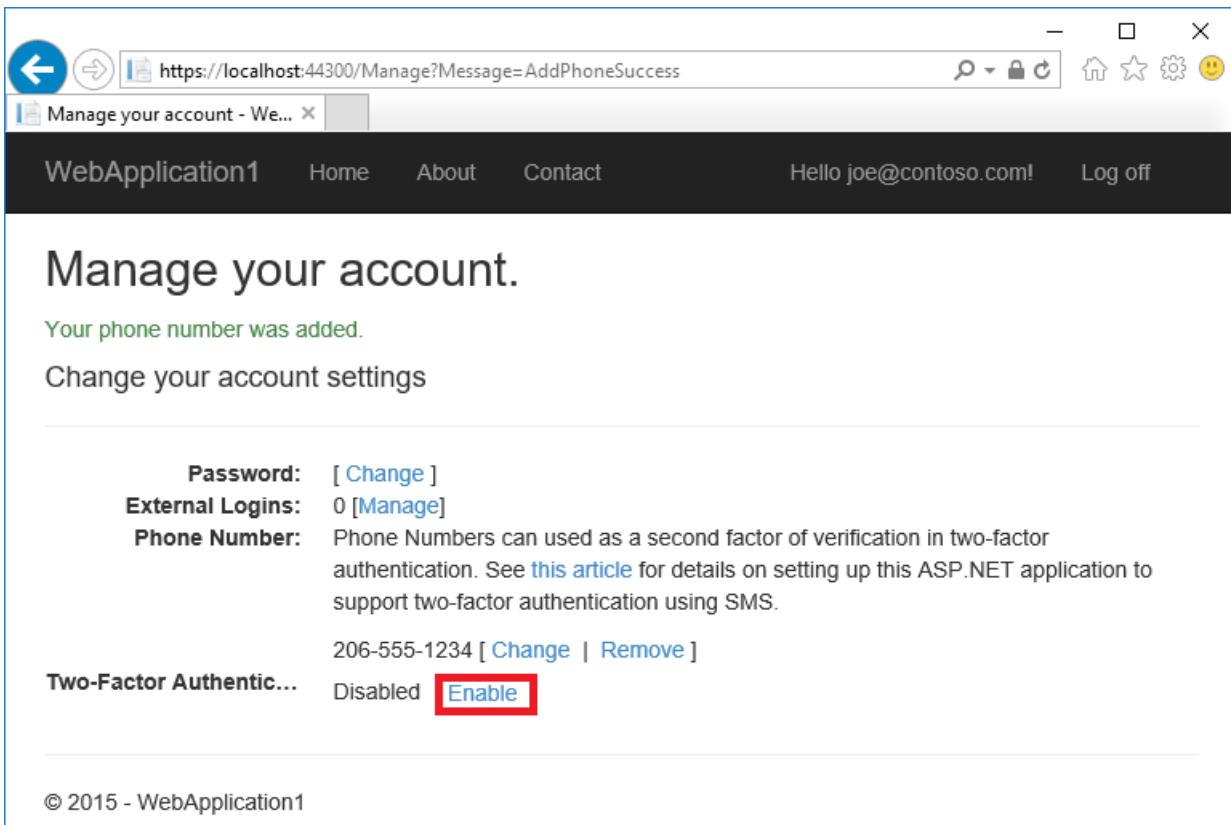
**Phone Number:** Phone Numbers can be used as a second factor of verification in two-factor authentication. See [this article](#) for details on setting up this ASP.NET application to support two-factor authentication using SMS.

206-555-1234 [ [Change](#) | [Remove](#) ]

**Two-Factor Authentic...** Disabled [Enable](#)

© 2015 - WebApplication1

- Toque **habilitar** para habilitar a autenticação de dois fatores.



Manage your account - We... Home About Contact Hello joe@contoso.com! Log off

## Manage your account.

Your phone number was added.

Change your account settings

**Password:** [ [Change](#) ]  
**External Logins:** 0 [\[Manage\]](#)

**Phone Number:** Phone Numbers can be used as a second factor of verification in two-factor authentication. See [this article](#) for details on setting up this ASP.NET application to support two-factor authentication using SMS.

206-555-1234 [ [Change](#) | [Remove](#) ]

**Two-Factor Authentic...** [Disabled](#) [Enable](#)

© 2015 - WebApplication1

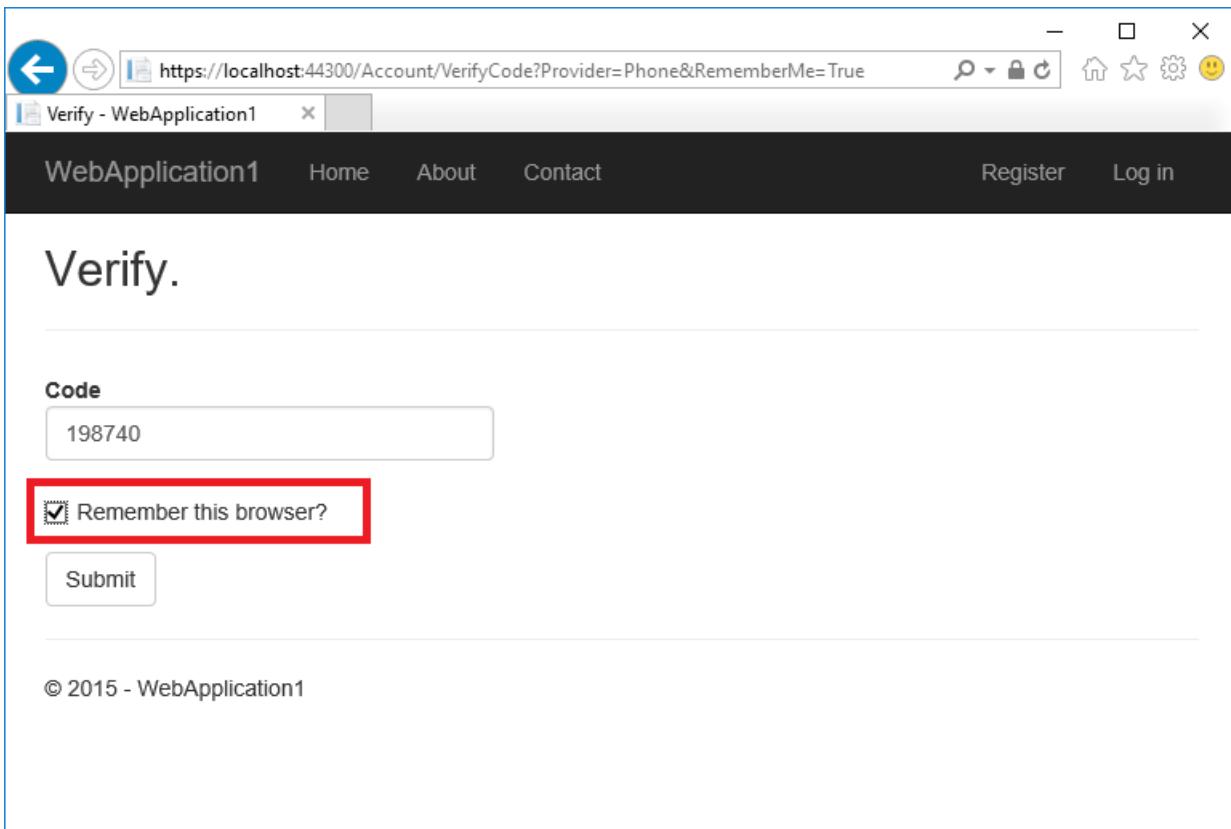
### Autenticação de dois fatores do teste

- Faça logoff.
- Iniciar sessão.
- A conta de usuário tiver habilitado a autenticação de dois fatores, portanto, você precisa fornecer o segundo fator de autenticação. Neste tutorial, você habilitou a verificação por telefone. Os modelos internos permitem que você configure o email como o segundo fator. Você pode configurar os fatores adicionais de

segundo para autenticação, como códigos QR. Toque **enviar**.

The screenshot shows a web browser window with the URL <https://localhost:44300/Account/SendCode?RememberMe=True>. The title bar says "Send Verification Code - W...". The page header includes the application name "WebApplication1" and navigation links for "Home", "About", "Contact", "Register", and "Log in". The main content area has a heading "Send Verification Code." and a form with the instruction "Select Two-Factor Authentication Provider:  ". A copyright notice at the bottom left reads "© 2015 - WebApplication1".

- Insira o código que você pode obter na mensagem SMS.
- Clicar na **lembra deste navegador** caixa de seleção serão isentos da necessidade de usar 2FA para fazer logon usando o mesmo dispositivo e o navegador. Habilitar a 2FA e clicando em **lembra deste navegador** lhe fornecerá 2FA forte proteção contra usuários mal-intencionados que tentam acessar sua conta, desde que não tiverem acesso ao seu dispositivo. Você pode fazer isso em qualquer dispositivo privado que você usa regularmente. Definindo **lembra deste navegador**, você obtém a segurança adicional de 2FA de dispositivos que você não usa regularmente, e você obtém a conveniência em não ter que passar por 2FA em seus próprios dispositivos.



## Bloqueio de conta para proteção contra ataques de força bruta

Bloqueio de conta é recomendado com 2FA. Depois que um usuário faz logon por meio de uma conta local ou social, cada tentativa com falha no 2FA é armazenada. Se as tentativas de acesso com falha máximo for atingido, o usuário está bloqueado (padrão: 5 minutos bloqueio após 5 falhas em tentativas de acesso). Uma autenticação bem-sucedida redefine a contagem de tentativas de acesso com falha e redefine o relógio. O máximo falhas em tentativas de acesso e tempo de bloqueio pode ser definido com [MaxFailedAccessAttempts](#) e [DefaultLockoutTimeSpan](#). O exemplo a seguir configura o bloqueio de conta por 10 minutos após 10 tentativas de acesso:

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();

    services.AddMvc();

    services.Configure<IdentityOptions>(options =>
    {
        options.Lockout.MaxFailedAccessAttempts = 10;
        options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(10);
    });

    // Add application services.
    services.AddTransient<IEmailSender, AuthMessageSender>();
    services.AddTransient<ISmsSender, AuthMessageSender>();
    services.Configure<SMSOptions>(Configuration);
}
```

Confirme [PasswordSignInAsync](#) define `lockoutOnFailure` para `true`:

```
var result = await _signInManager.PasswordSignInAsync(  
    Input.Email, Input.Password, Input.RememberMe, lockoutOnFailure: true);
```

# Usar autenticação de cookie sem o ASP.NET Core Identity

07/02/2019 • 29 minutes to read • [Edit Online](#)

Por [Rick Anderson](#) e [Luke Latham](#)

Como você viu nos tópicos anteriores de autenticação, [ASP.NET Core Identity](#) é um provedor de autenticação completa e a versão completa para criar e manter os logons. No entanto, você talvez queira usar sua própria lógica de autenticação personalizada com autenticação baseada em cookie às vezes. Você pode usar a autenticação baseada em cookie como um provedor de autenticação autônomo sem o ASP.NET Core Identity.

## [Exibir ou baixar código de exemplo \(como baixar\)](#)

Para fins de demonstração no aplicativo de exemplo, a conta de usuário para o usuário hipotético, Maria Rodriguez, é codificados no aplicativo. Use o nome de usuário de Email "maria.rodriguez@contoso.com" e nenhuma senha para a entrada do usuário. O usuário é autenticado na `AuthenticateUser` método na `Pages/Account/Login.cshtml.cs` arquivo. Em um exemplo do mundo real, o usuário deve ser autenticado em relação a um banco de dados.

Para obter informações sobre a autenticação baseada em cookie Migrando do ASP.NET Core 1.x para 2.0, consulte [migrar autenticação e identidade para o tópico do ASP.NET Core 2.0 \(autenticação baseada em Cookie\)](#).

Para usar a identidade do ASP.NET Core, consulte o [Introdução à identidade](#) tópico.

## Configuração

Se o aplicativo não usa o [metapacote do Microsoft](#), criar uma referência de pacote no arquivo de projeto para o `Microsoft.AspNetCore.Authentication.Cookies` pacote (versão 2.1.0 ou mais tarde).

No `ConfigureServices` método, crie o serviço de Middleware de autenticação com o `AddAuthentication` e `AddCookie` métodos:

```
services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie();
```

`AuthenticationScheme` passado para `AddAuthentication` define o esquema de autenticação padrão para o aplicativo. `AuthenticationScheme` é útil quando há várias instâncias de autenticação de cookie e você deseja autorizar com um esquema específico. Definindo o `AuthenticationScheme` para `CookieAuthenticationDefaults.AuthenticationScheme` fornece um valor de "Cookies" para o esquema. Você pode fornecer qualquer valor de cadeia de caracteres que diferencia o esquema.

Esquema de autenticação do aplicativo é diferente do esquema de autenticação de cookie do aplicativo. Quando um esquema de autenticação de cookie não é fornecido para `AddCookie`, ele usa `CookieAuthenticationDefaults.AuthenticationScheme` ("Cookies").

No `Configure` método, use o `UseAuthentication` método para invocar o Middleware de autenticação define o `HttpContext.User` propriedade. Chame o `UseAuthentication` método antes de chamar `UseMvcWithDefaultRoute` ou `UseMvc`:

```
app.UseAuthentication();
```

## Opções de AddCookie

O `CookieAuthenticationOptions` classe é usada para configurar as opções de provedor de autenticação.

OPÇÃO	DESCRIÇÃO
<code>AccessDeniedPath</code>	Fornece o caminho para fornecer a uma 302 não encontrado (redirecionamento de URL) quando disparado por <code>HttpContext.ForbidAsync</code> . O valor padrão é <code>/Account/AccessDenied</code> .
<code>ClaimsIssuer</code>	O emissor a ser usado para o <code>emissor</code> propriedade em quaisquer declarações criadas pelo serviço de autenticação de cookie.
<code>Cookie.Domain</code>	O nome de domínio no qual o cookie é atendido. Por padrão, isso é o nome do host da solicitação. O navegador envia apenas o cookie em solicitações para um nome de host correspondente. Talvez você queira ajustar isso para ter cookies disponíveis para qualquer host no seu domínio. Por exemplo, definir o domínio do cookie <code>.contoso.com</code> disponibiliza para <code>contoso.com</code> , <code>www.contoso.com</code> , e <code>staging.www.contoso.com</code> .
<code>Cookie.HttpOnly</code>	Um sinalizador que indica se o cookie deve ser acessível somente aos servidores. A alteração desse valor para <code>false</code> permite que os scripts do lado do cliente para acessar o cookie e pode abrir seu aplicativo ao roubo de cookie deve ter de seu aplicativo uma <a href="#">Cross-site scripting (XSS)</a> vulnerabilidade. O valor padrão é <code>true</code> .
<code>Cookie.Name</code>	Define o nome do cookie.
<code>Cookie.Path</code>	Usado para isolar aplicativos em execução no mesmo nome de host. Se você tiver um aplicativo em execução no <code>/app1</code> e para restringir os cookies para o aplicativo, defina a <code>CookiePath</code> propriedade <code>/app1</code> . Ao fazer isso, o cookie só está disponível em solicitações para <code>/app1</code> e qualquer aplicativo abaixo dela.
<code>Cookie.SameSite</code>	Indica se o navegador deve permitir que o cookie a ser anexado a somente solicitações do mesmo site ( <code>SameSiteMode.Strict</code> ) ou solicitações entre sites usando métodos seguros de HTTP e as solicitações do mesmo site ( <code>SameSiteMode.Lax</code> ). Quando definido como <code>SameSiteMode.None</code> , o valor do cabeçalho de cookie não está definido. Observe que <a href="#">Middleware do Cookie política</a> pode substituir o valor fornecido por você. Para dar suporte à autenticação OAuth, o valor padrão é <code>SameSiteMode.Lax</code> . Para obter mais informações, consulte <a href="#">autenticação OAuth interrompida devido à política de cookies SameSite</a> .
<code>Cookie.SecurePolicy</code>	Um sinalizador que indica se o cookie criado deve ser limitado a HTTPS ( <code>CookieSecurePolicy.Always</code> ), HTTP ou HTTPS ( <code>CookieSecurePolicy.None</code> ), ou o mesmo protocolo usado na solicitação ( <code>CookieSecurePolicy.SameAsRequest</code> ). O valor padrão é <code>CookieSecurePolicy.SameAsRequest</code> .

OPÇÃO	DESCRIÇÃO
<code>DataProtectionProvider</code>	Define o <code>DataProtectionProvider</code> que é usado para criar o padrão <code>TicketDataFormat</code> . Se o <code>TicketDataFormat</code> propriedade for definida, o <code>DataProtectionProvider</code> opção não é usada. Se não for fornecido, o provedor de proteção de dados do aplicativo padrão é usado.
<code>Eventos</code>	O manipulador chama métodos no provedor que fornecem o controle de aplicativo em determinados pontos de processamento. Se <code>Events</code> não são fornecidas, uma instância padrão é fornecida que não faz nada quando os métodos são chamados.
<code>EventsType</code>	Usado como o tipo de serviço para obter o <code>Events</code> instância em vez da propriedade.
<code>ExpireTimeSpan</code>	O <code>TimeSpan</code> depois que o tíquete de autenticação armazenado dentro do cookie expira. <code>ExpireTimeSpan</code> é adicionado à hora atual para criar o tempo de expiração para o tíquete. O <code>ExpiredTimeSpan</code> valor sempre entra no AuthTicket criptografado verificado pelo servidor. Isso também pode acontecer na <code>Set-Cookie</code> cabeçalho, mas somente se <code>IsPersistent</code> está definido. Para definir <code>IsPersistent</code> à <code>true</code> , configure o <code>AuthenticationProperties</code> passado para <code>SignInAsync</code> . O valor padrão de <code>ExpireTimeSpan</code> é de 14 dias.
<code>LoginPath</code>	Fornece o caminho para fornecer a uma 302 não encontrado (redirecionamento de URL) quando disparado por <code>HttpContext.ChallengeAsync</code> . A URL atual que gerou o 401 é adicionada para o <code>LoginPath</code> como um parâmetro de cadeia de caracteres de consulta nomeado pelo <code>ReturnUrlParameter</code> . Uma vez uma solicitação para o <code>LoginPath</code> concede uma nova entrada identidade, o <code>ReturnUrlParameter</code> valor é usado para redirecionar o navegador para a URL que causou o código de status não autorizado original. O valor padrão é <code>/Account/Login</code> .
<code>LogoutPath</code>	Se o <code>LogoutPath</code> é fornecido para o manipulador, em seguida, redireciona uma solicitação para o caminho com base no valor da <code>ReturnUrlParameter</code> . O valor padrão é <code>/Account/Logout</code> .
<code>ReturnUrlParameter</code>	Determina o nome do parâmetro de cadeia de consulta que é acrescentado pelo manipulador para uma resposta 302 do encontrado (redirecionamento de URL). <code>ReturnUrlParameter</code> é usado quando uma solicitação chega na <code>LoginPath</code> ou <code>LogoutPath</code> para retornar o navegador a URL original depois que a ação de logon ou logoff é executada. O valor padrão é <code>ReturnUrl</code> .
<code>SessionStore</code>	Um contêiner opcional usado para armazenar a identidade entre solicitações. Quando usado, somente um identificador de sessão é enviado ao cliente. <code>SessionStore</code> pode ser usado para atenuar problemas potenciais com identidades grandes.

OPÇÃO	DESCRIÇÃO
SlidingExpiration	Um sinalizador que indica se um novo cookie com um tempo de expiração atualizados deve ser emitido dinamicamente. Isso pode acontecer em qualquer solicitação em que o período de expiração do cookie atual mais de 50% expirou. A nova data de expiração é movida para frente até ser a data atual mais o <code>ExpireTimespan</code> . Uma <a href="#">tempo de expiração do cookie absoluto</a> pode ser definida usando o <code>AuthenticationProperties</code> classe ao chamar <code>SignInAsync</code> . Um tempo de expiração absoluto pode melhorar a segurança do seu aplicativo, limitando a quantidade de tempo que o cookie de autenticação é válido. O valor padrão é <code>true</code> .
TicketDataFormat	O <code>TicketDataFormat</code> é usado para proteger e desproteger a identidade e outras propriedades que são armazenadas no valor do cookie. Se não for fornecido, um <code>TicketDataFormat</code> é criado usando o <a href="#">DataProtectionProvider</a> .
Validar	Método que verifica que as opções são válidas.

Definir `CookieAuthenticationOptions` na configuração do serviço para autenticação no `ConfigureServices` método:

```
services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie(options =>
{
    ...
});
```

ASP.NET Core 1.x usa cookie [middleware](#) que serializa uma entidade de usuário em um cookie criptografado. Em solicitações subsequentes, o cookie é validado, e a entidade de segurança é recriada e atribuída ao `HttpContext.User` propriedade.

Instalar o [Microsoft.AspNetCore.Authentication.Cookies](#) pacote do NuGet em seu projeto. Este pacote contém o cookie de middleware.

Use o `UseCookieAuthentication` método no `Configure` método na sua `Startup.cs` antes do arquivo `UseMvc` ou `UseMvcWithDefaultRoute`:

```
app.UseCookieAuthentication(new CookieAuthenticationOptions()
{
    AccessDeniedPath = "/Account/Forbidden/",
    AuthenticationScheme = CookieAuthenticationDefaults.AuthenticationScheme,
    AutomaticAuthenticate = true,
    AutomaticChallenge = true,
    LoginPath = "/Account/Unauthorized/"
});
```

## Opções de `CookieAuthenticationOptions`

O `CookieAuthenticationOptions` classe é usada para configurar as opções de provedor de autenticação.

OPÇÃO	DESCRIÇÃO
-------	-----------

OPÇÃO	DESCRIÇÃO
<a href="#">AuthenticationScheme</a>	Define o esquema de autenticação. <code>AuthenticationScheme</code> é útil quando há várias instâncias de autenticação e você deseja autorizar com um esquema específico. Definindo o <code>AuthenticationScheme</code> para <code>CookieAuthenticationDefaults.AuthenticationScheme</code> fornece um valor de "Cookies" para o esquema. Você pode fornecer qualquer valor de cadeia de caracteres que diferencia o esquema.
<a href="#">AutomaticAuthenticate</a>	Define um valor para indicar que a autenticação de cookie deve executar em cada solicitação e tente validar e reconstrua qualquer entidade de segurança serializada criado por ele.
<a href="#">AutomaticChallenge</a>	Se for true, o middleware de autenticação lida com desafios automática. Se false, o middleware de autenticação altera somente as respostas quando explicitamente indicado pelo <code>AuthenticationScheme</code> .
<a href="#">ClaimsIssuer</a>	O emissor a ser usado para o <a href="#">emissor</a> propriedade em quaisquer declarações criado pelo middleware de autenticação de cookie.
<a href="#">CookieDomain</a>	O nome de domínio no qual o cookie é atendido. Por padrão, isso é o nome do host da solicitação. O navegador só serve o cookie para um nome de host correspondente. Talvez você queira ajustar isso para ter cookies disponíveis para qualquer host no seu domínio. Por exemplo, definir o domínio do cookie <code>.contoso.com</code> disponibiliza para <code>contoso.com</code> , <code>www.contoso.com</code> , e <code>staging.www.contoso.com</code> .
<a href="#">CookieHttpOnly</a>	Um sinalizador que indica se o cookie deve ser acessível somente aos servidores. A alteração desse valor para <code>false</code> permite que os scripts do lado do cliente para acessar o cookie e pode abrir seu aplicativo ao roubo de cookie deve ter de seu aplicativo uma <a href="#">Cross-site scripting (XSS)</a> vulnerabilidade. O valor padrão é <code>true</code> .
<a href="#">CookiePath</a>	Usado para isolar aplicativos em execução no mesmo nome de host. Se você tiver um aplicativo em execução no <code>/app1</code> e para restringir os cookies para o aplicativo, defina a <code>CookiePath</code> propriedade <code>/app1</code> . Ao fazer isso, o cookie só está disponível em solicitações para <code>/app1</code> e qualquer aplicativo abaixo dela.
<a href="#">CookieSecure</a>	Um sinalizador que indica se o cookie criado deve ser limitado a HTTPS ( <code>CookieSecurePolicy.Always</code> ), HTTP ou HTTPS ( <code>CookieSecurePolicy.None</code> ), ou o mesmo protocolo usado na solicitação ( <code>CookieSecurePolicy.SameAsRequest</code> ). O valor padrão é <code>CookieSecurePolicy.SameAsRequest</code> .
<a href="#">Descrição</a>	Informações adicionais sobre o tipo de autenticação que será disponibilizado para o aplicativo.

OPÇÃO	DESCRIÇÃO
ExpireTimeSpan	O <code>TimeSpan</code> depois que o tiquete de autenticação expira. Ele é adicionado à hora atual para criar o tempo de expiração para o tiquete. Para usar <code>ExpireTimeSpan</code> , você deve definir <code>IsPersistent</code> ao <code>true</code> no <code>AuthenticationProperties</code> passado para <code>SignInAsync</code> . O valor padrão é 14 dias.
SlidingExpiration	Um sinalizador que indica se a data de expiração do cookie redefine quando houver mais de metade do <code>ExpireTimeSpan</code> intervalo tenha decorrido. A nova hora <code>expiration</code> é movida para frente até ser a data atual mais o <code>ExpireTimespan</code> . Uma <a href="#">tempo de expiração do cookie absoluto</a> pode ser definida usando o <code>AuthenticationProperties</code> classe ao chamar <code>SignInAsync</code> . Um tempo de expiração absoluto pode melhorar a segurança do seu aplicativo, limitando a quantidade de tempo que o cookie de autenticação é válido. O valor padrão é <code>true</code> .

Definir `CookieAuthenticationOptions` para o Middleware de autenticação de Cookie no `Configure` método:

```
app.UseCookieAuthentication(new CookieAuthenticationOptions
{
    ...
});
```

## Cookie de Middleware de política

[Middleware do cookie política](#) habilita recursos de política de cookie em um aplicativo. Adicionar o middleware ao pipeline de processamento de aplicativos é a ordem de minúsculas; ela afeta apenas os componentes registrados depois no pipeline.

```
app.UseCookiePolicy(cookiePolicyOptions);
```

O `CookiePolicyOptions` fornecido para o Cookie de Middleware de política permitem controlar características globais do processamento de cookie e gancho em manipuladores de processamento do cookie quando os cookies são acrescentados ou excluídos.

PROPRIEDADE	DESCRIÇÃO
<code>HttpOnly</code>	Afeta se os cookies devem ser <code>HttpOnly</code> , que é um sinalizador que indica se o cookie deve ser acessível somente aos servidores. O valor padrão é <code>HttpOnlyPolicy.None</code> .
<code>MinimumSameSitePolicy</code>	Afeta o atributo de mesmo site do cookie (veja abaixo). O valor padrão é <code>SameSiteMode.Lax</code> . Essa opção está disponível para o ASP.NET Core 2.0 +.
<code>OnAppendCookie</code>	Chamado quando um cookie será acrescentado.
<code>OnDeleteCookie</code>	Chamado quando um cookie é excluído.

PROPRIEDADE	DESCRIÇÃO
Proteger	Afeta se os cookies devem ser seguro. O valor padrão é <code>CookieSecurePolicy.None</code> .

### MinimumSameSitePolicy (ASP.NET Core 2.0 ou posterior somente)

O padrão `MinimumSameSitePolicy` valor é `SameSiteMode.Lax` para permitir a autenticação OAuth2. Estritamente impor uma política do mesmo site do `SameSiteMode.Strict`, defina o `MinimumSameSitePolicy`. Embora essa configuração interrompe OAuth2 e outras esquemas de autenticação entre origens, ela eleva o nível de segurança do cookie para outros tipos de aplicativos que não dependem de processamento de solicitação entre origens.

```
var cookiePolicyOptions = new CookiePolicyOptions
{
    MinimumSameSitePolicy = SameSiteMode.Strict,
};
```

A configuração de Middleware de política de Cookie para `MinimumSameSitePolicy` pode afetar sua configuração de `Cookie.SameSite` em `CookieAuthenticationOptions` configurações de acordo com a matriz a seguir.

MINIMUMSAMESITEPOLICY	COOKIE.SAMESITE	CONFIGURAÇÃO DE COOKIE.SAMESITE RESULTANTE
SameSiteMode.None	SameSiteMode.None SameSiteMode.Lax SameSiteMode.Strict	SameSiteMode.None SameSiteMode.Lax SameSiteMode.Strict
SameSiteMode.Lax	SameSiteMode.None SameSiteMode.Lax SameSiteMode.Strict	SameSiteMode.Lax SameSiteMode.Lax SameSiteMode.Strict
SameSiteMode.Strict	SameSiteMode.None SameSiteMode.Lax SameSiteMode.Strict	SameSiteMode.Strict SameSiteMode.Strict SameSiteMode.Strict

## Criar um cookie de autenticação

Para criar um cookie contendo informações de usuário, você precisa construir uma `ClaimsPrincipal`. As informações do usuário são serializadas e armazenadas no cookie.

Criar uma `ClaimsIdentity` com qualquer necessárias `declarações` e chame `SignInAsync` para a entrada do usuário:

```

var claims = new List<Claim>
{
    new Claim(ClaimTypes.Name, user.Email),
    new Claim("FullName", user.FullName),
    new Claim(ClaimTypes.Role, "Administrator"),
};

var claimsIdentity = new ClaimsIdentity(
    claims, CookieAuthenticationDefaults.AuthenticationScheme);

var authProperties = new AuthenticationProperties
{
    //AllowRefresh = <bool>,
    // Refreshing the authentication session should be allowed.

    //ExpiresUtc = DateTimeOffset.UtcNow.AddMinutes(10),
    // The time at which the authentication ticket expires. A
    // value set here overrides the ExpireTimeSpan option of
    // CookieAuthenticationOptions set with AddCookie.

    //IsPersistent = true,
    // Whether the authentication session is persisted across
    // multiple requests. Required when setting the
    // ExpireTimeSpan option of CookieAuthenticationOptions
    // set with AddCookie. Also required when setting
    // ExpiresUtc.

    //IssuedUtc = <DateTimeOffset>,
    // The time at which the authentication ticket was issued.

    //RedirectUri = <string>
    // The full path or absolute URI to be used as an http
    // redirect response value.
};

await HttpContext.SignInAsync(
    CookieAuthenticationDefaults.AuthenticationScheme,
    new ClaimsPrincipal(claimsIdentity),
    authProperties);

```

Chame [SignInAsync](#) para a entrada do usuário:

```

await HttpContext.Authentication.SignInAsync(
    CookieAuthenticationDefaults.AuthenticationScheme,
    new ClaimsPrincipal(claimsIdentity));

```

`SignInAsync` cria um cookie criptografado e o adiciona à resposta atual. Se você não especificar um `AuthenticationScheme`, o esquema padrão é usado.

Nos bastidores, a criptografia usada é ASP.NET Core [proteção de dados](#) sistema. Se você estiver hospedando o aplicativo em várias máquinas, balanceamento de carga entre aplicativos ou usar uma web farm, você deve [configurar a proteção de dados](#) para usar o mesmo anel de chave e o identificador do aplicativo.

## Sair

Para desconectar o usuário atual e excluir seus cookies, chame [SignOutAsync](#):

```

await HttpContext.SignOutAsync(
    CookieAuthenticationDefaults.AuthenticationScheme);

```

Para desconectar o usuário atual e excluir seus cookies, chame [SignOutAsync](#):

```
await HttpContext.Authentication.SignOutAsync(  
    CookieAuthenticationDefaults.AuthenticationScheme);
```

Se você não estiver usando `CookieAuthenticationDefaults.AuthenticationScheme` (ou "Cookies") como o esquema (por exemplo, "ContosoCookie"), forneça o esquema usado ao configurar o provedor de autenticação. Caso contrário, o esquema padrão é usado.

## Reagir às alterações de back-end

Depois que um cookie é criado, ele se torna a única fonte de identidade. Mesmo se você desabilitar um usuário em seus sistemas de back-end, o sistema de autenticação de cookie não tem conhecimento disso, e um usuário permaneça conectado enquanto seu cookie é válido.

O `ValidatePrincipal` evento no ASP.NET Core 2.x ou o `ValidateAsync` método no ASP.NET Core 1.x pode ser usado para interceptar e substituir a validação da identidade do cookie. Essa abordagem minimiza o risco de usuários revogados acessando o aplicativo.

Uma abordagem de validação de cookie baseia-se em manter o controle de quando o banco de dados do usuário foi alterado. Se o banco de dados não foi alterado desde que o cookie do usuário foi emitido, não é necessário para autenticar o usuário novamente se o cookie ainda é válido. Para implementar este cenário, o banco de dados, que é implementado no `IUserRepository` para este exemplo armazena um `LastChanged` valor. Quando nenhum usuário é atualizado no banco de dados, o `LastChanged` valor é definido como a hora atual.

Para invalidar um cookie, quando as alterações de banco de dados com base nas `LastChanged` o valor, criar o cookie com um `LastChanged` contendo atual de declaração `LastChanged` valor do banco de dados:

```
var claims = new List<Claim>  
{  
    new Claim(ClaimTypes.Name, user.Email),  
    new Claim("LastChanged", {Database Value})  
};  
  
var claimsIdentity = new ClaimsIdentity(  
    claims,  
    CookieAuthenticationDefaults.AuthenticationScheme);  
  
await HttpContext.SignInAsync(  
    CookieAuthenticationDefaults.AuthenticationScheme,  
    new ClaimsPrincipal(claimsIdentity));
```

Para implementar uma substituição para o `ValidatePrincipal` eventos, escreva um método com a assinatura a seguir em uma classe que você deriva `CookieAuthenticationEvents`:

```
ValidatePrincipal(CookieValidatePrincipalContext)
```

Um exemplo é semelhante ao seguinte:

```

using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;

public class CustomCookieAuthenticationEvents : CookieAuthenticationEvents
{
    private readonly IUserRepository _userRepository;

    public CustomCookieAuthenticationEvents(IUserRepository userRepository)
    {
        // Get the database from registered DI services.
        _userRepository = userRepository;
    }

    public override async Task ValidatePrincipal(CookieValidatePrincipalContext context)
    {
        var userPrincipal = context.Principal;

        // Look for the LastChanged claim.
        var lastChanged = (from c in userPrincipal.Claims
                           where c.Type == "LastChanged"
                           select c.Value).FirstOrDefault();

        if (string.IsNullOrEmpty(lastChanged) ||
            !_userRepository.ValidateLastChanged(lastChanged))
        {
            context.RejectPrincipal();

            await context.HttpContext.SignOutAsync(
                CookieAuthenticationDefaults.AuthenticationScheme);
        }
    }
}

```

Registrar a instância de eventos durante o registro do serviço de cookie no `ConfigureServices` método. Fornecer um registro de serviço com escopo para sua `CustomCookieAuthenticationEvents` classe:

```

services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie(options =>
{
    options.EventsType = typeof(CustomCookieAuthenticationEvents);
});

services.AddScoped<CustomCookieAuthenticationEvents>();

```

Para implementar uma substituição para o `ValidateAsync` eventos, escreva um método com a seguinte assinatura:

```
ValidateAsync(CookieValidatePrincipalContext)
```

Identidade do ASP.NET Core implementa essa verificação como parte de sua `SecurityStampValidator`. Um exemplo é semelhante ao seguinte:

```

public static class LastChangedValidator
{
    public static async Task ValidateAsync(CookieValidatePrincipalContext context)
    {
        // Pull database from registered DI services.
        var userRepository =
            context.HttpContext.RequestServices
                .GetRequiredService<IUserRepository>();
        var userPrincipal = context.Principal;

        // Look for the last changed claim.
        var lastChanged = (from c in userPrincipal.Claims
                           where c.Type == "LastChanged"
                           select c.Value).FirstOrDefault();

        if (string.IsNullOrEmpty(lastChanged) ||
            !userRepository.ValidateLastChanged(lastChanged))
        {
            context.RejectPrincipal();

            await context.HttpContext.SignOutAsync(
                CookieAuthenticationDefaults.AuthenticationScheme);
        }
    }
}

```

Registrar o evento durante a configuração de autenticação de cookie no `Configure` método:

```

app.UseCookieAuthentication(new CookieAuthenticationOptions
{
    Events = new CookieAuthenticationEvents
    {
        OnValidatePrincipal = LastChangedValidator.ValidateAsync
    }
});

```

Considere uma situação em que o nome do usuário é atualizado — uma decisão que não afeta a segurança de qualquer forma. Se você quiser atualizar forma não destrutiva a entidade de usuário, chame `context.ReplacePrincipal` e defina o `context.ShouldRenew` propriedade `true`.

#### **WARNING**

A abordagem descrita aqui é disparada em cada solicitação. Isso pode resultar em uma penalidade de desempenho grande para o aplicativo.

## Cookies persistentes

Talvez você queira que o cookie para persistir entre as sessões do navegador. Essa persistência deve ser habilitada apenas com o consentimento explícito do usuário com uma caixa de seleção "Lembrar-Me" no logon ou um mecanismo semelhante.

O trecho de código a seguir cria uma identidade e o cookie correspondente que sobrevive a por meio de fechamentos de navegador. Quaisquer configurações de expiração deslizante configuradas anteriormente são consideradas. Se o cookie expirar enquanto o navegador é fechado, o navegador limpa o cookie depois que ele seja reiniciado.

```
await HttpContext.SignInAsync(
    CookieAuthenticationDefaults.AuthenticationScheme,
    new ClaimsPrincipal(claimsIdentity),
    new AuthenticationProperties
    {
        IsPersistent = true
    });

```

O [AuthenticationProperties](#) classe reside no `Microsoft.AspNetCore.Authentication` namespace.

```
await HttpContext.Authentication.SignInAsync(
    CookieAuthenticationDefaults.AuthenticationScheme,
    new ClaimsPrincipal(claimsIdentity),
    new AuthenticationProperties
    {
        IsPersistent = true
    });

```

O [AuthenticationProperties](#) classe reside no `Microsoft.AspNetCore.Http.Authentication` namespace.

## Expiração do cookie absoluto

Você pode definir um tempo de expiração absoluto com `ExpiresUtc`. Você também deve definir `IsPersistent`; caso contrário, `ExpiresUtc` será ignorado e um cookie de sessão único é criado. Quando `ExpiresUtc` é definida em `SignInAsync`, ele substitui o valor da `ExpireTimeSpan` opção de `CookieAuthenticationOptions`, se definido.

O trecho de código a seguir cria uma identidade e o cookie correspondente que dura por 20 minutos. Isso ignora as configurações de expiração deslizante configuradas anteriormente.

```
await HttpContext.SignInAsync(
    CookieAuthenticationDefaults.AuthenticationScheme,
    new ClaimsPrincipal(claimsIdentity),
    new AuthenticationProperties
    {
        IsPersistent = true,
        ExpiresUtc = DateTime.UtcNow.AddMinutes(20)
    });

```

```
await HttpContext.Authentication.SignInAsync(
    CookieAuthenticationDefaults.AuthenticationScheme,
    new ClaimsPrincipal(claimsIdentity),
    new AuthenticationProperties
    {
        IsPersistent = true,
        ExpiresUtc = DateTime.UtcNow.AddMinutes(20)
    });

```

## Recursos adicionais

- [As alterações do AUTH 2.0 / migração comunicado](#)
- [Autorizar com um esquema específico no ASP.NET Core](#)
- [Autorização baseada em declarações no núcleo do ASP.NET](#)
- [Verificações de função baseado em políticas](#)
- [Hospedar o ASP.NET Core em um web farm](#)

# Azure Active Directory com o ASP.NET Core

10/12/2018 • 2 minutes to read • [Edit Online](#)

## Exemplos do Azure Active Directory V1

Os exemplos a seguir mostram como integrar o Azure Active Directory V1, permitindo que os usuários entrem com uma conta corporativa ou de estudante:

- [Integrando o Azure AD em um aplicativo Web ASP.NET Core](#)
- [Chamando uma API Web ASP.NET Core em um aplicativo do WPF usando o Azure AD](#)
- [Chamando uma API Web em um aplicativo Web ASP.NET Core usando o Azure AD](#)

## Exemplos do Azure Active Directory V2

Os exemplos a seguir mostram como integrar o Azure Active Directory V2, permitindo que os usuários entrem com uma conta corporativa ou de estudante ou com uma conta pessoal da Microsoft (antiga conta Live):

- [Integrando o Azure Active Directory V2 em um aplicativo Web ASP.NET Core 2.0:](#)
  - Consulte [este vídeo associado](#)
- [Chamando uma API Web ASP.NET Core 2.0 em um aplicativo do WPF usando o Azure Active Directory V2:](#)
  - Consulte [este vídeo associado](#)

## Exemplo do Azure Active Directory B2C

Este exemplo mostra como integrar o Azure Active Directory B2C, permitindo que os usuários entrem com contas sociais (como Facebook, Google...)

- [Um aplicativo API Web ASP.NET Core com o Azure AD B2C](#)

# Autenticação de nuvem com o Azure Active Directory B2C no ASP.NET Core

08/01/2019 • 10 minutes to read • [Edit Online](#)

Por [Cam Soper](#)

[Azure Active Directory B2C do diretório](#) (Azure AD B2C) é uma solução de gerenciamento de identidade de nuvem para aplicativos web e móveis. O serviço fornece autenticação para aplicativos hospedados na nuvem e locais. Tipos de autenticação incluem contas individuais, contas de rede social e contas corporativas de federado. Além disso, o Azure AD B2C pode fornecer a autenticação multifator com configuração mínima.

## TIP

Azure Active Directory (Azure AD) e o Azure AD B2C são ofertas de produtos separados. Um locatário do AD do Azure representa uma organização, enquanto que um locatário do Azure AD B2C representa uma coleção de identidades a serem usados com aplicativos de terceira parte confiável. Para obter mais informações, consulte [do Azure AD B2C: Perguntas frequentes \(FAQ\)](#).

Neste tutorial, saiba como:

- Criar um locatário do Azure Active Directory B2C
- Registrar um aplicativo no Azure B2C do AD
- Usar o Visual Studio para criar um aplicativo web do ASP.NET Core configurado para usar o locatário do Azure AD B2C para autenticação
- Configurar políticas para controlar o comportamento do locatário do Azure AD B2C

## Pré-requisitos

A seguir é necessários para este passo a passo:

- [Assinatura do Microsoft Azure](#)
- [Visual Studio 2017](#) (qualquer edição)

## Criar o locatário do Azure Active Directory B2C

Criar um locatário do Azure Active Directory B2C [conforme descrito na documentação do](#). Quando solicitado, associar o locatário com uma assinatura do Azure é opcional para este tutorial.

## Registrar o aplicativo no Azure B2C do AD

No locatário do Azure AD B2C recém-criado, registre seu aplicativo usando [as etapas na documentação do](#) sob o **registrar um aplicativo web** seção. Parar na **criar um segredo do cliente de aplicativo web** seção. Um segredo do cliente não é necessário para este tutorial.

Use os seguintes valores:

CONFIGURAÇÃO	VALOR	OBSERVAÇÕES
--------------	-------	-------------

CONFIGURAÇÃO	VALOR	OBSERVAÇÕES
<b>Nome</b>	<Nome do aplicativo>	Insira um <b>nome</b> para o aplicativo que descrevem seu aplicativo para os consumidores.
<b>Incluir aplicativo web / API web</b>	Sim	
<b>Permitir fluxo implícito</b>	Sim	
<b>URL de resposta</b>	<code>https://localhost:44300/signin-oidc</code>	URLs de resposta são pontos de extremidade onde o Azure AD B2C retornará os tokens que o aplicativo solicitar. O Visual Studio fornece a URL de resposta para usar. Por enquanto, insira <code>https://localhost:44300/signin-oidc</code> preencha o formulário.
<b>URI da ID do aplicativo</b>	Deixe em branco	Não é necessário para este tutorial.
<b>Incluir cliente nativo</b>	Não	

#### WARNING

Se estiver ciente de como configurar uma URL de resposta não sejam localhost, o [restrições sobre o que é permitido na lista de URL de resposta](#).

Depois que o aplicativo é registrado, é exibida a lista de aplicativos no locatário. Selecione o aplicativo que acabou de registrar. Selecione o **cópia** ícone à direita do **ID do aplicativo** campo para copiá-lo na área de transferência.

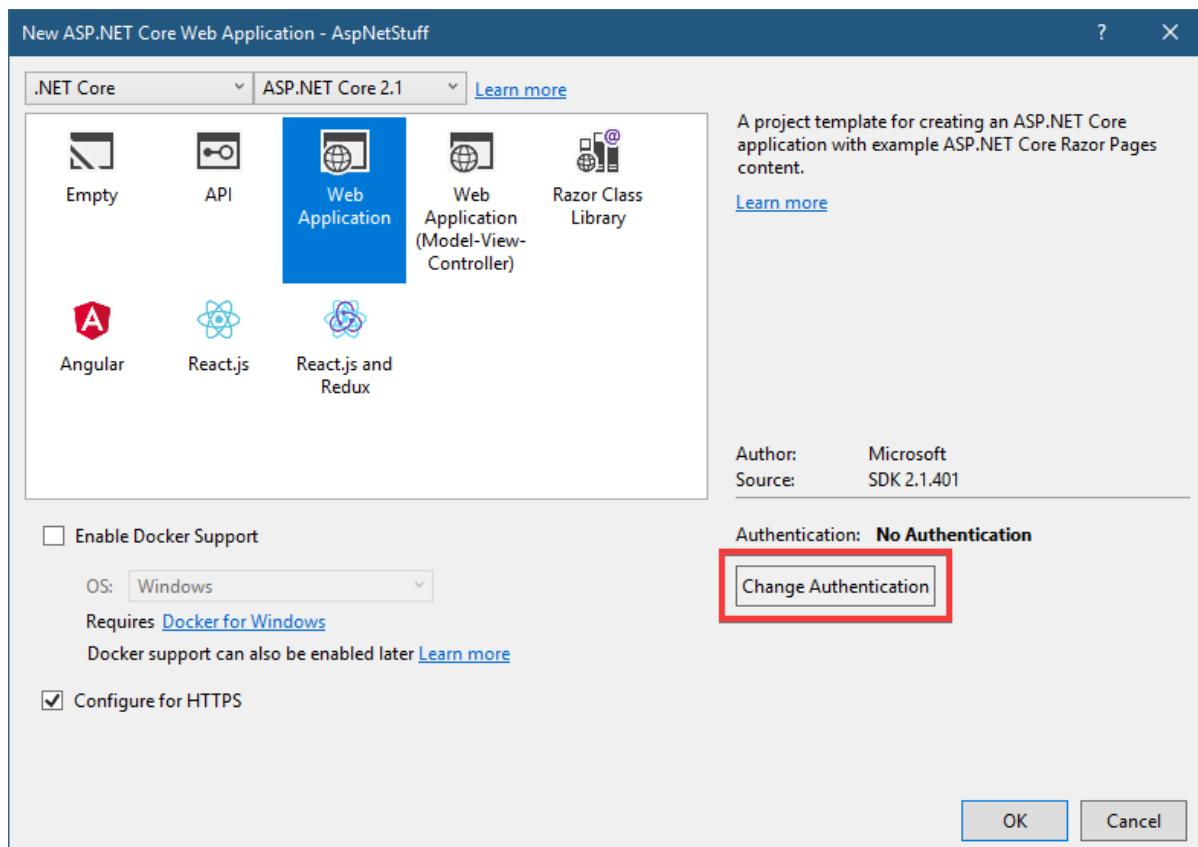
Nada mais podem ser configurado no locatário do Azure AD B2C neste momento, mas deixe a janela do navegador aberta. Não há mais de configuração depois que o aplicativo ASP.NET Core é criado.

## Criar um aplicativo ASP.NET Core no Visual Studio 2017

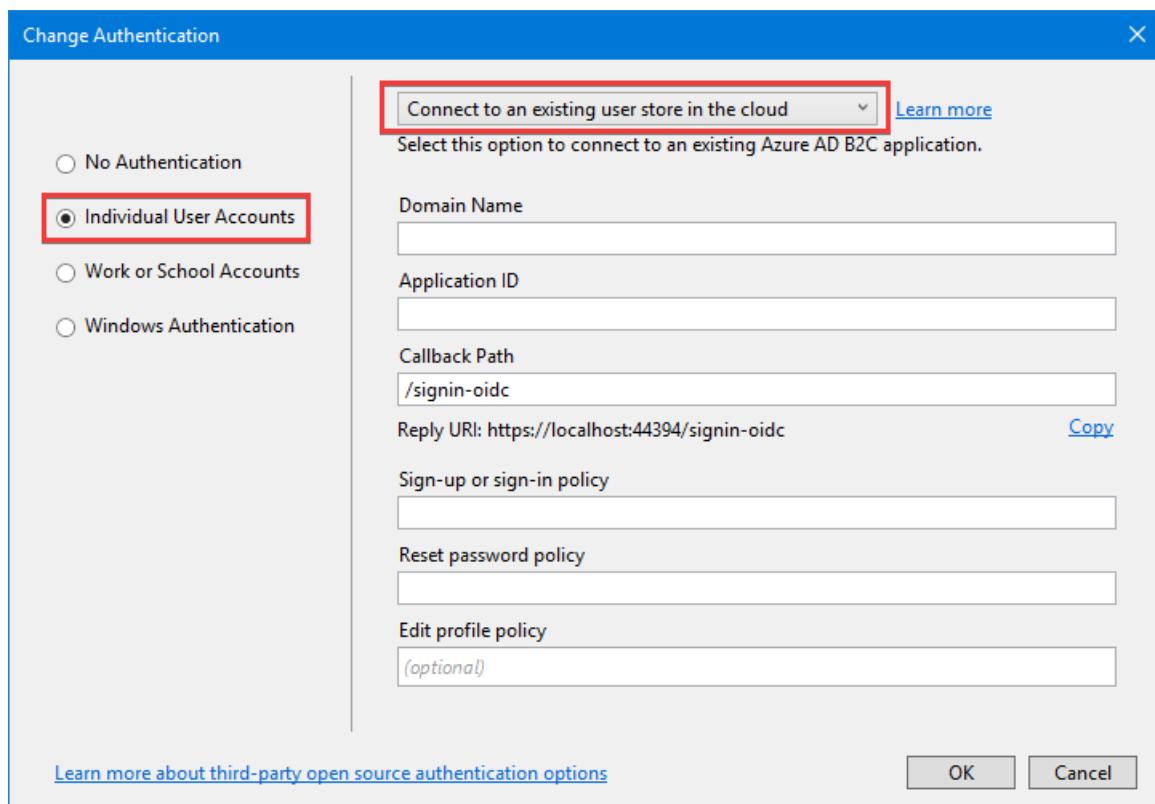
O modelo de aplicativo Web Visual Studio pode ser configurado para usar o locatário do Azure AD B2C para autenticação.

No Visual Studio:

1. Crie um novo Aplicativo Web ASP.NET Core.
2. Selecione **aplicativo Web** da lista de modelos.
3. Selecione o **alterar autenticação** botão.



4. No **alterar autenticação** caixa de diálogo, selecione **contas de usuário individuais**, em seguida, selecione **conectar-se ao repositório de usuário existente na nuvem** na lista suspensa.



5. Preencha o formulário com os seguintes valores:

CONFIGURAÇÃO	VALOR
Nome de domínio	<o nome de domínio do seu locatário do B2C>

CONFIGURAÇÃO	VALOR
<b>ID do aplicativo</b>	<Cole a ID do aplicativo da área de transferência>
<b>Caminho de retorno de chamada</b>	<Use o valor padrão>
<b>Política de inscrição ou entrada</b>	B2C_1_SiUpIn
<b>Política de redefinição de senha</b>	B2C_1_SSPr
<b>Editar política de perfil</b>	<Deixe em branco>

Selecione o **cópia** próximo ao link **URI de resposta** para copiar o URI de resposta para a área de transferência. Selecione **Okey** para fechar o **alterar autenticação** caixa de diálogo. Selecione **Okey** para criar o aplicativo web.

## Concluir o registro do aplicativo B2C

Retornar à janela do navegador com as propriedades do aplicativo B2C ainda abertas. Alterar temporários **URL de resposta** especificado anteriormente para o valor copiado do Visual Studio. Selecione **salvar** na parte superior da janela.

### TIP

Se você não copiar a URL de resposta, use o endereço HTTPS na guia Debug nas propriedades do projeto da web e acrescente o **CallbackPath** o valor da *appSettings.json*.

## Configurar políticas

Use as etapas na documentação do Azure AD B2C para [criar uma política de inscrição ou entrada](#) e então [criar uma política de redefinição de senha](#). Use os valores de exemplo fornecidos na documentação do **provedores de identidade, atributos de inscrição, e declarações do aplicativo**. Usando o **executar agora** botão para testar as políticas, conforme descrito na documentação é opcional.

### WARNING

Verifique se os nomes de política são exatamente como descrito na documentação, como essas políticas foram usadas na **alterar autenticação** caixa de diálogo no Visual Studio. Os nomes de política podem ser verificados no *appSettings.json*.

## Executar o aplicativo

No Visual Studio, pressione **F5** para compilar e executar o aplicativo. Depois que o aplicativo web for iniciado, selecione **Accept** para aceitar o uso de cookies (se solicitado) e, em seguida, selecione **entrar**.

The screenshot shows a web browser window with the URL <https://localhost:44394/>. The page has a dark header with the text "AzureB2CTutorial" and navigation links for "Home", "About", and "Contact". A red box highlights the "Sign in" button in the top right corner. Below the header is a green banner with the word "Packages" and links for "NuGet", "npm", "Bower", and "Gulp". The banner also contains text about bringing in libraries from NuGet and npm, and automating tasks using Grunt or Gulp, with a "Learn More" button. At the bottom of the banner are three small circular icons.

## Application uses

- Sample pages using ASP.NET Core Razor Pages
- Theming using [Bootstrap](#)

## How to

- Working with Razor Pages.
- Manage User Secrets using [Secret Manager](#).
- Use logging to log a message.
- Add packages using [NuGet](#).

O navegador é redirecionado para o locatário do Azure AD B2C. Entrar com uma conta existente (se uma foi criada a testar as políticas) ou selecione **Inscreva-se agora** para criar uma nova conta. O **esqueceu sua senha?** link é usado para redefinir uma senha esquecida.

The screenshot shows a web browser window with the URL <https://login.microsoftonline.com/te/camthegeekb2c.onmicrosoft.com>. The page features a blue background with a lightbulb icon and a city skyline illustration at the bottom. It displays a "Sign in with your existing account" message and two input fields for "Email Address" and "Password". Below the fields is a "Sign in" button. A "Forgot your password?" link is located above the "Password" field. At the bottom, there is a link for users who don't have an account: "Don't have an account? [Sign up now](#)".

Depois de entrar com êxito, o navegador é redirecionado para o aplicativo web.

The screenshot shows a web browser window with the title "Home page - AzureB2C". The address bar displays the URL "https://localhost:44394/". The page content includes a navigation bar with links for "AzureB2CTutorial", "Home", "About", "Contact", "Hello Tutorial User!", and "Sign out". Below the navigation bar is a purple banner with the Visual Studio logo and icons for HTML, CSS, and JS. The banner text reads: "There are powerful new features in Visual Studio for building modern web apps." with a "Learn More" button. The main content area has a heading "Application uses" followed by a bulleted list: "Sample pages using ASP.NET Core Razor Pages" and "Theming using Bootstrap". Another section titled "How to" contains a bulleted list: "Working with Razor Pages.", "Manage User Secrets using Secret Manager.", "Use logging to log a message.", and "Add packages using NuGet."

## Próximas etapas

Neste tutorial, você aprendeu como:

- Criar um locatário do Azure Active Directory B2C
- Registrar um aplicativo no Azure B2C do AD
- Usar o Visual Studio para criar um aplicativo Web do ASP.NET Core configurados para usar o locatário do Azure AD B2C para autenticação
- Configurar políticas para controlar o comportamento do locatário do Azure AD B2C

Agora que o aplicativo ASP.NET Core está configurado para usar o Azure AD B2C para autenticação, o [atributo Authorize](#) pode ser usado para proteger seu aplicativo. Continue a desenvolver seu aplicativo aprendendo a:

- Personalizar a interface do usuário do Azure AD B2C.
- Configurar os requisitos de complexidade de senha.
- Habilitar a autenticação multifator.
- Configurar provedores de identidade adicional, como Microsoft, Facebook, Google, Amazon, do Twitter e outros.
- Usar a API do Graph do Azure AD para recuperar informações de usuário adicionais, como associação de grupo, do locatário do Azure AD B2C.
- Proteger um ASP.NET Core API da web usando o Azure AD B2C.
- Chamar uma API web de um aplicativo web do .NET usando o Azure AD B2C.

# Autenticação em APIs web com o Azure Active Directory B2C no ASP.NET Core

08/01/2019 • 17 minutes to read • [Edit Online](#)

Por [Cam Soper](#)

Azure Active Directory B2C do diretório (Azure AD B2C) é uma solução de gerenciamento de identidade de nuvem para aplicativos web e móveis. O serviço fornece autenticação para aplicativos hospedados na nuvem e locais. Tipos de autenticação incluem contas individuais, contas de rede social e contas corporativas de federado. B2C do AD do Azure também fornece a autenticação multifator com configuração mínima.

Azure Active Directory (Azure AD) e o Azure AD B2C são ofertas de produtos separados. Um locatário do AD do Azure representa uma organização, enquanto que um locatário do Azure AD B2C representa uma coleção de identidades a serem usados com aplicativos de terceira parte confiável. Para obter mais informações, consulte [do Azure AD B2C: Perguntas frequentes \(FAQ\)](#).

Uma vez que as APIs da web não tem nenhuma interface do usuário, eles são não é possível redirecionar o usuário a um serviço de token seguro, como o Azure AD B2C. Em vez disso, a API é passada um token de portador do aplicativo de chamada, que já tiver autenticado o usuário com o Azure AD B2C. A API, em seguida, valida o token sem interação direta do usuário.

Neste tutorial, saiba como:

- Crie um locatário do Azure Active Directory B2C.
- Registre uma API da Web no B2C do AD do Azure.
- Use o Visual Studio para criar uma API da Web configurado para usar o locatário do Azure AD B2C para autenticação.
- Configure políticas para controlar o comportamento do locatário do Azure AD B2C.
- Usar o Postman para simular um aplicativo web que apresenta uma caixa de diálogo de logon, recupera um token e usa-o para fazer uma solicitação em relação a API da web.

## Pré-requisitos

A seguir é necessários para este passo a passo:

- [Assinatura do Microsoft Azure](#)
- [Visual Studio 2017](#) (qualquer edição)
- [Postman](#)

## Criar o locatário do Azure Active Directory B2C

Criar um locatário do Azure AD B2C [conforme descrito na documentação do](#). Quando solicitado, associar o locatário com uma assinatura do Azure é opcional para este tutorial.

## Configurar uma política de inscrição ou entrada

Use as etapas na documentação do Azure AD B2C para [criar uma política de inscrição ou entrada](#). Nomeie a diretiva **SiUpIn**. Use os valores de exemplo fornecidos na documentação do **provedores de identidade**, **atributos de inscrição**, e **declarações do aplicativo**. Usando o **executar agora** botão para testar a política conforme descrito na documentação é opcional.

## Registrar a API no B2C do AD do Azure

No locatário do Azure AD B2C recém-criado, registre sua API usando [as etapas na documentação do](#) sob o **registrar uma API web** seção.

Use os seguintes valores:

CONFIGURAÇÃO	VALOR	OBSERVAÇÕES
<b>Nome</b>	{Nome da API}	Insira um <b>nome</b> para o aplicativo que descrevem seu aplicativo para os consumidores.
<b>Incluir aplicativo web / API web</b>	Sim	
<b>Permitir fluxo implícito</b>	Sim	
<b>URL de resposta</b>	<code>https://localhost</code>	URLs de resposta são pontos de extremidade onde o Azure AD B2C retornará os tokens que o aplicativo solicitar.
<b>URI da ID do aplicativo</b>	<code>api</code>	O URI não precisa resolver para um endereço físico. Ele só precisa ser exclusivo.
<b>Incluir cliente nativo</b>	Não	

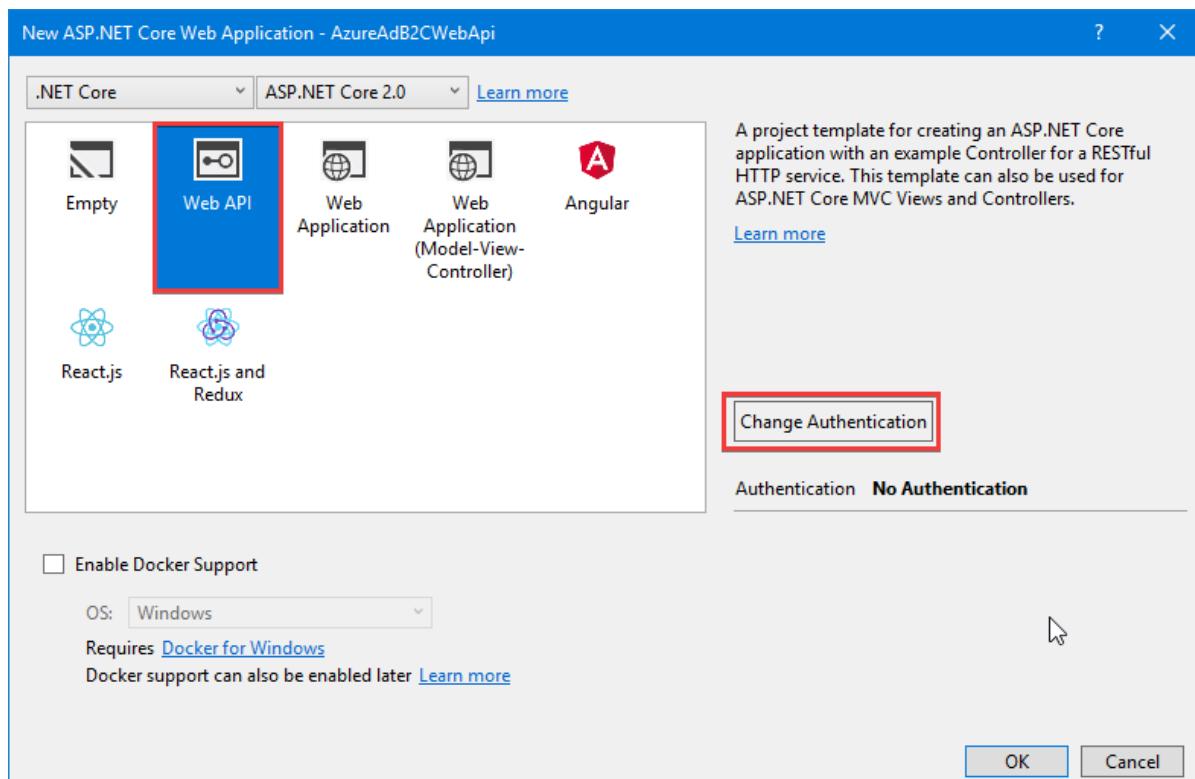
Depois que a API é registrada, é exibida a lista de aplicativos e APIs no locatário. Selecione a API que foi registrada anteriormente. Selecione o **cópia** ícone à direita do **ID do aplicativo** campo para copiá-lo na área de transferência. Selecione **escopos publicados** e verifique se o padrão `user_impersonation` escopo está presente.

## Criar um aplicativo ASP.NET Core no Visual Studio 2017

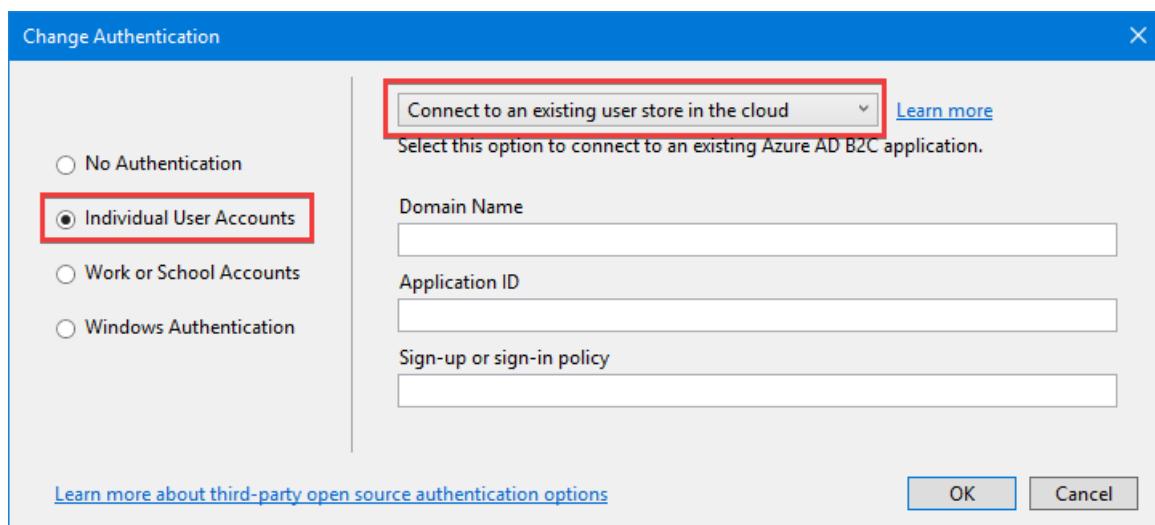
O modelo de aplicativo Web Visual Studio pode ser configurado para usar o locatário do Azure AD B2C para autenticação.

No Visual Studio:

1. Crie um novo Aplicativo Web ASP.NET Core.
2. Selecione **API Web** da lista de modelos.
3. Selecione o **alterar autenticação** botão.



4. No **alterar autenticação** caixa de diálogo, selecione **contas de usuário individuais > conectar-se ao repositório de usuário existente na nuvem**.



5. Preencha o formulário com os seguintes valores:

CONFIGURAÇÃO	VALOR
<b>Nome de domínio</b>	{o nome de domínio do seu locatário B2C}
<b>ID do aplicativo</b>	{Colar a ID do aplicativo da área de transferência}
<b>Política de inscrição ou entrada</b>	B2C_1_SiUpIn

Selecione **Okey** para fechar o **alterar autenticação** caixa de diálogo. Selecione **Okey** para criar o aplicativo web.

O Visual Studio cria a API da web com um controlador chamado `ValuesController.cs` que retorna valores embutidos para solicitações GET. A classe é decorada com o [atributo Authorize](#), portanto, todas as solicitações requerem autenticação.

## Executar a API da web

No Visual Studio, execute a API. O Visual Studio inicia um navegador apontado na URL da raiz da API. Anote a URL na barra de endereços e deixar a API em execução em segundo plano.

### NOTE

Como não há nenhum controlador definido para a URL da raiz, o navegador pode exibir um erro 404 de (página não encontrada). Esse comportamento é esperado.

## Usar o Postman para obter um token e a API de teste

[Postman](#) é uma ferramenta para testar APIs web. Para este tutorial, carteiro simula um aplicativo web que acessa a API da web em nome do usuário.

### Registre o Postman como um aplicativo web

Uma vez que o Postman simula um aplicativo web que obtém tokens de locatário do Azure AD B2C, ele deve ser registrado no locatário como um aplicativo web. Registrar o Postman usando [as etapas na documentação do](#) sob o **registrar um aplicativo web** seção. Parar na **criar um segredo do cliente de aplicativo web** seção. Um segredo do cliente não é necessário para este tutorial.

Use os seguintes valores:

CONFIGURAÇÃO	VALOR	OBSERVAÇÕES
<b>Nome</b>	Postman	
<b>Incluir aplicativo web / API web</b>	Sim	
<b>Permitir fluxo implícito</b>	Sim	
<b>URL de resposta</b>	<code>https://getpostman.com/postman</code>	
<b>URI da ID do aplicativo</b>	{Deixe em branco}	Não é necessário para este tutorial.
<b>Incluir cliente nativo</b>	Não	

O aplicativo web registrado recentemente precisa de permissão para acessar a API da web em nome do usuário.

1. Selecione **Postman** na lista de aplicativos e selecione **acesso à API** no menu à esquerda.
2. Selecione **+ adicionar**.
3. No **selecionar API** lista suspensa, selecione o nome da API web.
4. No **selecionar escopos** lista suspensa, verifique se todos os escopos são selecionados.
5. Selecione **Okey**.

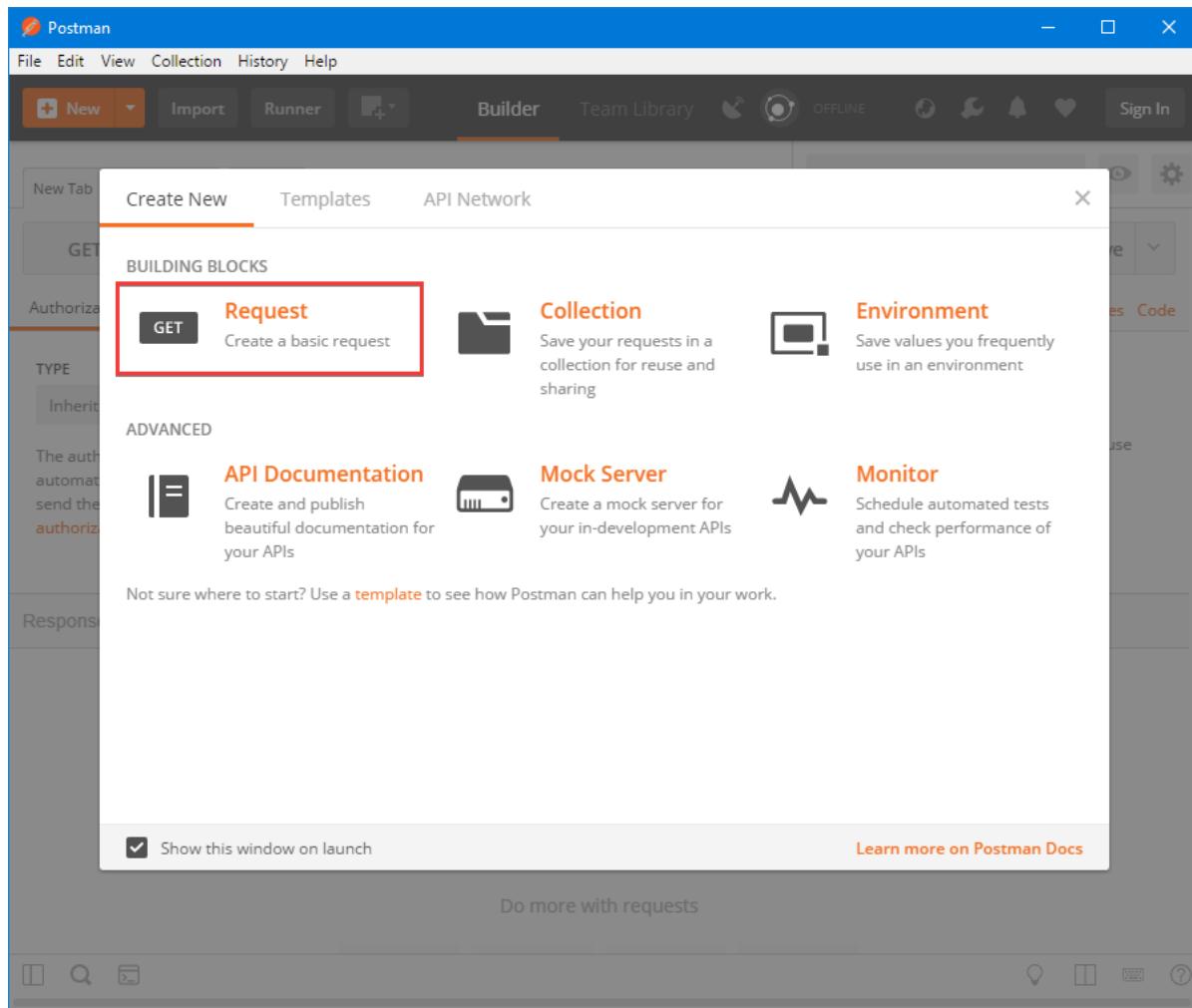
Observe a ID do aplicativo do aplicativo Postman, pois ele é necessário para obter um token de portador.

### Criar uma solicitação do Postman

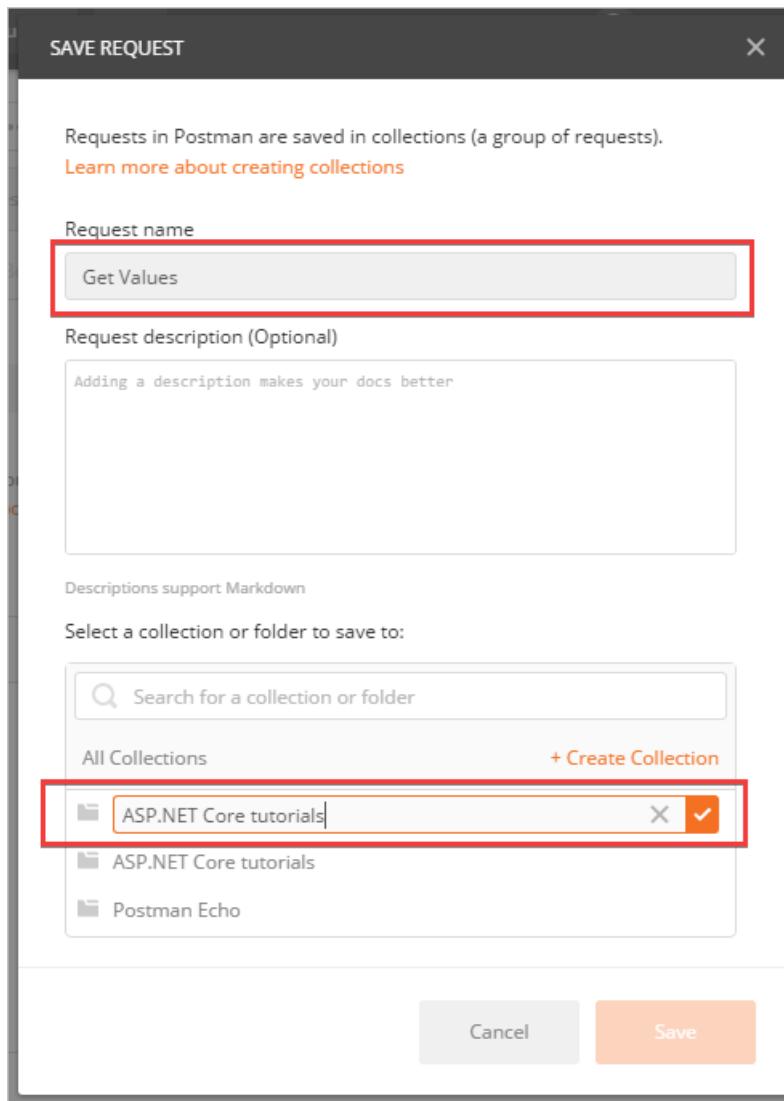
Inicie o Postman. Por padrão, o Postman exibe a **criar novo** caixa de diálogo após a inicialização. Se a caixa de diálogo não for exibida, selecione a **+ novo** botão no canto superior esquerdo.

Dos **criar novo** caixa de diálogo:

1. Selecione **solicitar**.



2. Insira *obter valores* na **nome da solicitação** caixa.
3. Selecione + **Criar coleção** para criar uma nova coleção para armazenar a solicitação. Nomear a coleção *tutoriais do ASP.NET Core* e, em seguida, selecione a marca de seleção.

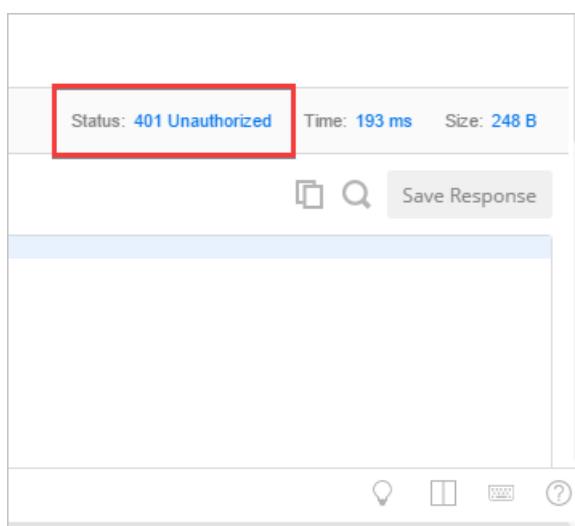


4. Selecione o **salvar em tutoriais do ASP.NET Core** botão.

#### Testar a API da web sem autenticação

Para verificar que a API da web requer autenticação, primeiro verifique uma solicitação sem autenticação.

1. No **insira a URL da solicitação**, digite a URL para `valuesController`. A URL é o mesmo exibido no navegador com **api/valores** acrescentado. Por exemplo, `https://localhost:44375/api/values`.
2. Selecione o **enviar** botão.
3. Observe o status da resposta é *401 não autorizado*.



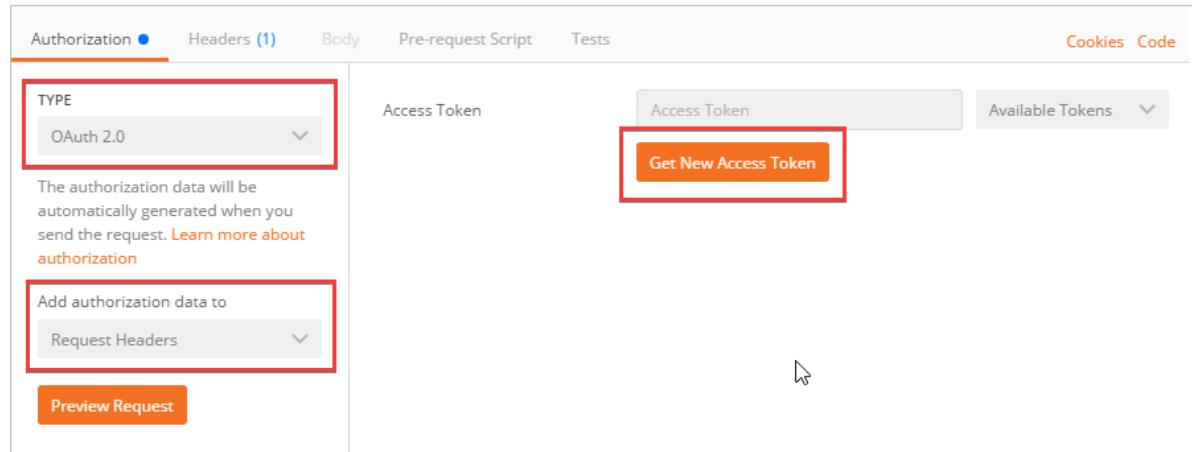
## IMPORTANT

Se você receber um erro "Não foi possível obter qualquer resposta", você talvez precise desabilitar a verificação do certificado SSL na [configurações do Postman](#).

## Obter um token de portador

Para fazer uma solicitação autenticada a API da web, é necessário um token de portador. Postman torna mais fácil entrar no locatário do Azure AD B2C e obter um token.

1. Sobre o **autorização** guia da **tipo** lista suspensa, selecione **OAuth 2.0**. No **adicionar dados de autorização** lista suspensa, selecione **cabeçalhos de solicitação**. Selecione **obter Token de acesso novo**.



2. Conclua o **obter novo TOKEN de acesso** caixa de diálogo da seguinte maneira:

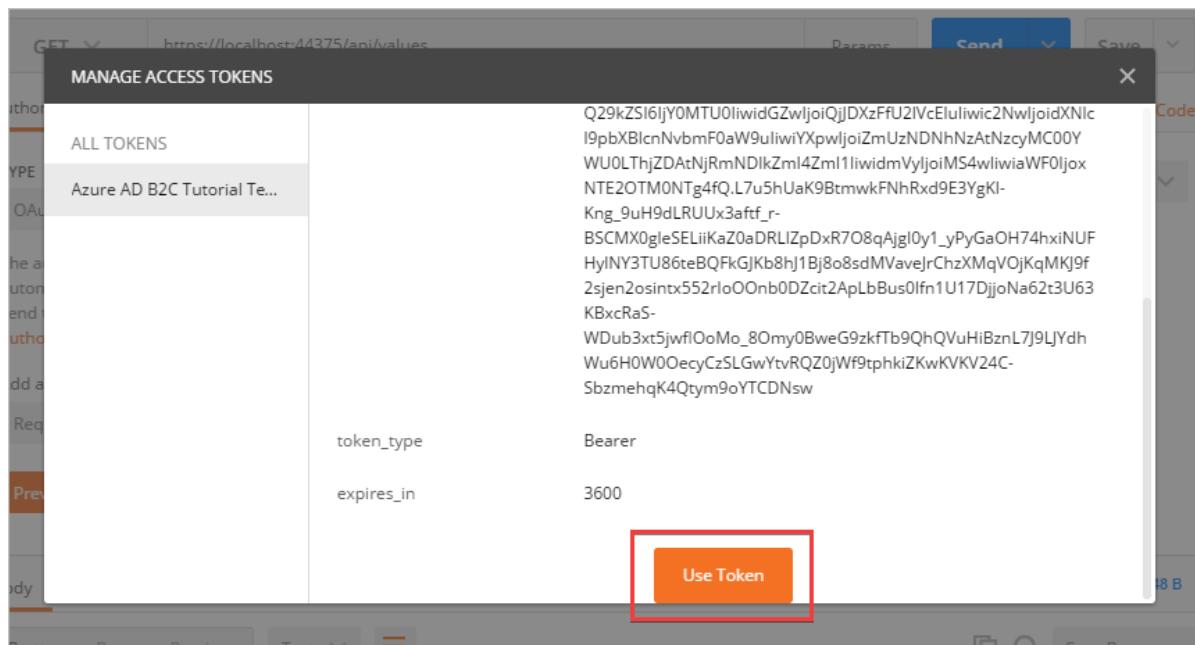
CONFIGURAÇÃO	VALOR	OBSERVAÇÕES
<b>Nome do token</b>	{nome do token}	Insira um nome descritivo para o token.
<b>Tipo de concessão</b>	Implícita	
<b>URL de retorno de chamada</b>	<a href="https://getpostman.com/postman">https://getpostman.com/postman</a>	
<b>URL de autenticação</b>	<a href="https://login.microsoftonline.com/{tSubstitua {nome de domínio do domain name}}/oauth2/v2.0/authorize?locatário">https://login.microsoftonline.com/{tSubstitua {nome de domínio do domain name}}/oauth2/v2.0/authorize?locatário</a> com o nome de domínio do locatário. <b>IMPORTANTE:</b> Essa URL deve ter o mesmo nome de domínio que o que for encontrado na <code>AzureAdB2C.Instance</code> da API web <code>appSettings.json</code> arquivo. Consulte a Observação.	<small>Substitua {t} por {tSubstitua}, {n} por {nome de domínio}, {d} por {domain name} e {l} por {locatário}. O nome de domínio deve ser o mesmo que o nome de domínio da sua API web.</small>
<b>ID do cliente</b>	{entrar no aplicativo de Postman ID do aplicativo}	

CONFIGURAÇÃO	VALOR	OBSERVAÇÕES
<b>Escopo</b>	<code>https://{{tenant domain name}}/{{api}}/user_impersonation openid offline_access</code>	Substitua <code>{nome de domínio do locatário}</code> com o nome de domínio do locatário. Substitua <code>{api}</code> com o URI da ID do aplicativo você deu a API da web ao registrado pela primeira vez (nesse caso, <code>api</code> ). O padrão para a URL é: <code>https://{{tenant}}.onmicrosoft.com/{{api-id-uri}}/{{scope name}}</code>
<b>Estado</b>	<i>{Deixe em branco}</i>	
<b>Autenticação de cliente</b>	Enviar as credenciais do cliente no corpo	

#### NOTE

+ A caixa de diálogo de configurações de política no portal do Azure Active Directory B2C exibe duas URLs possíveis: Uma no formato `https://login.microsoftonline.com/ {{nome de domínio do locatário}} / {informações adicionais de caminho}` e o outro no formato `https://{{tenant name}}.b2clogin.com/ {{nome de domínio do locatário}} / {informações adicionais de caminho}`. Ele tem **críticos** que o domínio encontrado na `AzureAdB2C.Instance` da API `web appSettings. JSON` arquivo corresponde ao usado no aplicativo de `web appSettings. JSON` arquivo. Isso é o mesmo domínio usado para o campo de URL do Auth no Postman. Observe que o Visual Studio usa um formato de URL ligeiramente diferente que o que é exibido no portal. Desde que os domínios corresponderem, a URL funciona.

3. Selecione o **solicitar Token** botão.
4. Postman abre uma nova janela que contém o diálogo de logon do locatário do Azure AD B2C. Entrar com uma conta existente (se uma foi criada a testar as políticas) ou selecione **Inscreva-se agora** para criar uma nova conta. O **esqueceu sua senha?** link é usado para redefinir uma senha esquecida.
5. Depois de entrar com êxito, a janela será fechada e o **gerenciar TOKENS de acesso** caixa de diálogo é exibida. Role para baixo até a parte inferior e selecione o **uso Token** botão.



#### A API da web com autenticação de teste

Selecione o **enviar** botão para enviar a solicitação novamente. Neste momento, o status de resposta estiver 200 Okey e o conteúdo do JSON está visível na resposta **corpo** guia.

The screenshot shows the Postman interface with the following details:

- Header tabs: Body, Cookies, Headers (6), Test Results.
- Status bar: Status: 200 OK, Time: 111 ms, Size: 287 B.
- Content area:
  - Format: JSON (Pretty).
  - Response body:

```
1 [  
2   "value1",  
3   "value2"  
4 ]
```
- Buttons: Save Response.

## Próximas etapas

Neste tutorial, você aprendeu como:

- Crie um locatário do Azure Active Directory B2C.
- Registre uma API da Web no B2C do AD do Azure.
- Use o Visual Studio para criar uma API da Web configurado para usar o locatário do Azure AD B2C para autenticação.
- Configure políticas para controlar o comportamento do locatário do Azure AD B2C.
- Usar o Postman para simular um aplicativo web que apresenta uma caixa de diálogo de logon, recupera um token e usa-o para fazer uma solicitação em relação a API da web.

Continue a desenvolver sua API pelo learning para:

- [Proteger um ASP.NET Core em aplicativo web usando o Azure AD B2C.](#)
- [Chamar uma API web de um aplicativo web do .NET usando o Azure AD B2C.](#)
- [Personalizar a interface do usuário do Azure AD B2C.](#)
- [Configurar os requisitos de complexidade de senha.](#)
- [Habilitar a autenticação multifator.](#)
- Configurar provedores de identidade adicional, como [Microsoft](#), [Facebook](#), [Google](#), [Amazon](#), [do Twitter](#) e outros.
- [Usar a API do Graph do Azure AD para recuperar informações de usuário adicionais, como associação de grupo, do locatário do Azure AD B2C.](#)

# Artigos baseados em projetos do ASP.NET Core criados com contas de usuário individuais

21/09/2018 • 2 minutes to read • [Edit Online](#)

O ASP.NET Core Identity é incluído nos modelos de projeto no Visual Studio com a opção "Contas de usuário individuais".

Os modelos de autenticação estão disponíveis na CLI do .NET Core com `-au Individual`:

```
dotnet new mvc -au Individual  
dotnet new webapi -au Individual  
dotnet new webapp -au Individual
```

```
dotnet new mvc -au Individual  
dotnet new webapi -au Individual  
dotnet new razor -au Individual
```

## Sem autenticação

Autenticação é especificada na CLI do .NET Core com o `-au` opção. No Visual Studio, o **alterar autenticação** caixa de diálogo está disponível para novos aplicativos da web. É o padrão para novos aplicativos web no Visual Studio **sem autenticação**.

Projetos criados com nenhuma autenticação:

- Não contêm páginas da web e a interface do usuário para entrar e sair.
- Não contêm o código de autenticação.

## Autenticação do Windows

Autenticação do Windows é especificada para novos aplicativos web na CLI do .NET Core com o `-au Windows` opção. No Visual Studio, o **alterar autenticação** caixa de diálogo fornece o **autenticação do Windows** opções.

Se a autenticação do Windows é selecionada, o aplicativo está configurado para usar o [módulo do IIS da autenticação Windows](#). Autenticação do Windows destina-se a sites da Intranet.

## Recursos adicionais

Os artigos a seguir mostram como usar o código gerado em modelos do ASP.NET Core que usam contas de usuário individual:

- [Autenticação de dois fatores com SMS](#)
- [Confirmação de conta e de recuperação de senha no ASP.NET Core](#)
- [Criar um aplicativo ASP.NET Core com os dados de usuário protegidos por autorização](#)

# Introdução à autorização no núcleo do ASP.NET

22/06/2018 • 2 minutes to read • [Edit Online](#)

Autorização é o processo que determina o que um usuário pode fazer. Por exemplo, um usuário administrativo tem permissão para criar uma biblioteca de documentos e adicionar, editar e excluir documentos. Um usuário não administrativo trabalhando com esta biblioteca só está autorizado a ler os documentos.

A autorização é ortogonal e independente da autenticação. No entanto, a autorização requer um mecanismo de autenticação. Autenticação é o processo de verificação de quem é um usuário. A autenticação pode criar uma ou mais identidades para o usuário atual.

## Tipos de autorização

A autorização do ASP.NET Core fornece um modelo simples de [funções declarativas baseado em políticas](#). Ela é expressa em requisitos e os manipuladores avaliam as reivindicações de um usuário em relação aos requisitos. Verificações imperativas podem ser baseadas em políticas simples ou políticas que avaliem a identidade do usuário e propriedades do recurso que o usuário está tentando acessar.

## Namespaces

Componentes de autorização, incluindo os atributo `AuthorizeAttribute` e `AllowAnonymousAttribute`, são encontrados no namespace `Microsoft.AspNetCore.Authorization`.

Consulte a documentação em [autorização simples](#).

# Criar um aplicativo ASP.NET Core com os dados de usuário protegidos por autorização

08/01/2019 • 28 minutes to read • [Edit Online](#)

Por [Rick Anderson](#) e [Joe Audette](#)

Ver [esse PDF](#) para a versão do ASP.NET Core MVC. A versão 1.1 do ASP.NET Core deste tutorial está [nesta](#) pasta. O exemplo do ASP.NET Core 1.1 está em [exemplos](#).

Consulte [esse pdf](#)

Este tutorial mostra como criar um aplicativo web do ASP.NET Core com dados de usuário protegidos por autorização. Ele exibe uma lista de contatos criada por usuários autenticados (registrados). Há três grupos de segurança:

- **Usuários registrados** podem exibir todos os dados aprovados e podem editar/excluir seus próprios dados.
- **Gerenciadores de** podem aprovar ou rejeitar dados de um contato. Apenas os contatos aprovados são visíveis aos usuários.
- **Os administradores** podem rejeitar/aprovar e editar/excluir todos os dados.

Na imagem a seguir, o usuário Rick (`rick@example.com`) está conectado. Rick só pode ver os contatos aprovados e os links **editar/excluir/criar novo** de seus contatos. Somente o último registro criado por Rick exibe os links **editar** e **excluir**. Outros usuários não verão o último registro até que um gerente ou administrador altere o status para "Aprovado".

The screenshot shows a browser window with the URL `https://localhost:44380/Contacts`. The page title is "ContactManager". The top navigation bar includes links for "Home", "About", and "Contact". On the right, there is a welcome message "Hello rick@example.com!" and a "Log off" link. Below the navigation, a heading "Create New" is followed by a table displaying contact information. The table has columns: Address, City, Email, Name, State, Zip, and Status. The contacts listed are:

Address	City	Email	Name	State	Zip	Status	
5678 1st Ave W	Redmond	<a href="#">thorsten@example.com</a>	Thorsten Weinrich	WA	10999	Approved	<a href="#">Details</a>
9012 State St	Redmond	<a href="#">yuhong@example.com</a>	Yuhong Li	WA	10999	Approved	<a href="#">Details</a>
3456 Maple St	Redmond	<a href="#">jon@example.com</a>	Jon Orton	WA	10999	Approved	<a href="#">Details</a>
123 N 456 E	GF	<a href="#">rick@example.com</a>	Rick Anderson	MT	59405	Submitted	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

At the bottom left, there is a copyright notice: "© 2017 - ContactManager".

Na imagem a seguir, `manager@contoso.com` está conectado e na função de gerenciadores:

The screenshot shows the ContactManager application's index page. At the top, there is a navigation bar with links for Home, About, and Contact. On the right side of the navigation bar, there is a yellow box highlighting the text "Hello manager@contoso.com!". Below the navigation bar, there is a heading "Create New". The main content area displays a table with columns: Address, City, Email, Name, State, Zip, and Status. The table contains six rows of contact information. The last row, which corresponds to the contact with ID 1006, has its "Status" cell highlighted with a red border. The status value for this row is "Submitted". At the bottom of the page, there is a copyright notice: "© 2016 - ContactManager".

Address	City	Email	Name	State	Zip	Status
1234 Main St	Redmond	<a href="#">debra@example.com</a>	Debra Garcia	WA	10999	Rejected
5678 1st Ave W	Redmond	<a href="#">thorsten@example.com</a>	Thorsten Weinrich	WA	10999	Approved
9012 State st	Redmond	<a href="#">yuhong@example.com</a>	Yuhong Li	WA	10999	Approved
3456 Maple St	Redmond	<a href="#">jon@example.com</a>	Jon Orton	WA	10999	Approved
7890 2nd Ave E	Redmond	<a href="#">diliana@example.com</a>	Diliana Alexieva-Bosseva	WA	10999	Submitted
123 N 456 E	GF	<a href="#">rick@example.com</a>	Rick Anderson	MT	59405	Approved

A imagem a seguir mostra a tela de exibição de detalhes de um contato dos gerentes:

The screenshot shows the ContactManager application's details page for a contact. At the top, there is a navigation bar with links for Home, About, and Contact. On the right side of the navigation bar, there is a yellow box highlighting the text "Hello manager@contoso.com!". Below the navigation bar, there is a heading "Contact". The main content area displays a table with columns: Name, Email, Address, City, State, Zip, and Status. The table contains one row of contact information for Rick Anderson. Below the table, there are two buttons: "Approve" (green) and "Reject" (red). At the bottom of the page, there is a copyright notice: "© 2016 - ContactManager".

Name	Rick Anderson
Email	<a href="#">rick@example.com</a>
Address	123 N 456 E
City	GF
State	MT
Zip	59405
Status	Submitted

[Approve](#) [Reject](#)  
[Edit](#) | [Back to List](#)

O botões **aprovar** e **rejeitar** são exibidos somente para administradores e gerentes.

Na imagem a seguir, `admin@contoso.com` está conectado e na função de gerenciadores:

Address	City	Email	Name	State	Zip	Status	
1234 Main St	Redmond	<a href="#">debra@example.com</a>	Debra Garcia	WA	10999	Rejected	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
5678 1st Ave W	Redmond	<a href="#">thorsten@example.com</a>	Thorsten Weinrich	WA	10999	Approved	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
9012 State st	Redmond	<a href="#">yuhong@example.com</a>	Yuhong Li	WA	10999	Approved	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
3456 Maple St	Redmond	<a href="#">jon@example.com</a>	Jon Orton	WA	10999	Approved	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

O administrador tem todos os privilégios. Ele pode ler/editar/excluir todos os contatos e alterar os status deles.

O aplicativo foi criado fazendo [scaffolding](#) do seguinte modelo de `Contact`:

```
public class Contact
{
    public int ContactId { get; set; }
    public string Name { get; set; }
    public string Address { get; set; }
    public string City { get; set; }
    public string State { get; set; }
    public string Zip { get; set; }
    [DataType(DataType.EmailAddress)]
    public string Email { get; set; }
}
```

O exemplo contém os seguintes manipuladores de autorização:

- `ContactIsOwnerAuthorizationHandler`: Garante que um usuário só pode editar seus dados.
- `ContactManagerAuthorizationHandler`: Permite que os gerentes aprovar ou rejeitar os contatos.
- `ContactAdministratorsAuthorizationHandler`: Permite aos administradores para aprovar ou rejeitar os contatos e editar/excluir contatos.

## Pré-requisitos

Este tutorial é avançado. Você deve estar familiarizado com:

- [ASP.NET Core](#)
- [Autenticação](#)

- Confirmação de conta e recuperação de senha
- Autorização
- Entity Framework Core

No ASP.NET Core 2.1, `User.IsInRole` Falha ao usar `AddDefaultIdentity`. Este tutorial usa `AddDefaultIdentity` e, portanto, exige o ASP.NET Core 2.2 ou posterior. Ver [esse problema de GitHub](#) para uma solução alternativa.

## O aplicativo inicial e o concluído

Baixe as [concluída](#) aplicativo. Teste o aplicativo concluído para que você se familiarize com seus recursos de segurança.

### O aplicativo inicial

Baixe o aplicativo [inicial](#).

Execute o aplicativo, clique em **ContactManager** e verifique se você pode criar, editar e excluir um contato.

## Proteger os dados de usuário

As seções a seguir têm todas as principais etapas para criar um aplicativo com dados de usuários seguros. Talvez seja útil para fazer referência ao projeto concluído.

### Vincular os dados de contato ao usuário

Usar o ASP.NET [identidade](#) ID de usuário para garantir que os usuários pode editar seus dados, mas não outros dados de usuários. Adicione `OwnerID` e `ContactStatus` para o `Contact` modelo:

```
public class Contact
{
    public int ContactId { get; set; }

    // user ID from AspNetUser table.
    public string OwnerID { get; set; }

    public string Name { get; set; }
    public string Address { get; set; }
    public string City { get; set; }
    public string State { get; set; }
    public string Zip { get; set; }
    [DataType(DataType.EmailAddress)]
    public string Email { get; set; }

    public ContactStatus Status { get; set; }
}

public enum ContactStatus
{
    Submitted,
    Approved,
    Rejected
}
```

`OwnerID` é a ID do usuário da tabela `AspNetUser` no banco de dados de [identidade](#). O campo `Status` determina se um contato pode ser visto por usuários gerais.

Criar uma nova migração e atualizar o banco de dados:

```
dotnet ef migrations add userID_Status
dotnet ef database update
```

## Adicionar serviços de função à identidade

Acrescente [AddRoles](#) para adicionar serviços de função:

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddDbContext<ApplicationContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));
    services.AddDefaultIdentity<IdentityUser>().AddRoles<IdentityRole>()
        .AddEntityFrameworkStores<ApplicationContext>();
}
```

## Exigir usuários autenticados

Defina a política de autenticação padrão para exigir que os usuários sejam autenticados:

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddDbContext<ApplicationContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));
    services.AddDefaultIdentity<IdentityUser>().AddRoles<IdentityRole>()
        .AddEntityFrameworkStores<ApplicationContext>();

    services.AddMvc(config =>
    {
        // using Microsoft.AspNetCore.Mvc.Authorization;
        // using Microsoft.AspNetCore.Authorization;
        var policy = new AuthorizationPolicyBuilder()
            .RequireAuthenticatedUser()
            .Build();
        config.Filters.Add(new AuthorizeFilter(policy));
    })
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
}
```

Você pode recusar a autenticação em nível de método de ação, controlador ou página Razor com o `[AllowAnonymous]` atributo. Configurar a política de autenticação padrão para exigir que os usuários sejam autenticados protege recém-adicionado páginas do Razor e controladores. Autenticação necessária por padrão é mais segura do que contar com novos controladores e páginas do Razor para incluir o `[Authorize]` atributo.

Adicionar `AllowAnonymous` às páginas Índice, Sobre e Contatos para que usuários anônimos possam obter informações sobre o site antes de eles se registrarem.

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace ContactManager.Pages
{
    [AllowAnonymous]
    public class IndexModel : PageModel
    {
        public void OnGet()
        {

        }
    }
}

```

## Configurar a conta de teste

A classe `SeedData` cria duas contas: administrador e gerenciador. Use a [ferramenta Gerenciador de segredo](#) para definir uma senha para essas contas. Defina a senha do diretório do projeto (o diretório que contém `Program.cs`):

```
dotnet user-secrets set SeedUserPW <PW>
```

Se uma senha forte não for especificada, uma exceção é gerada quando `SeedData.Initialize` é chamado.

Atualização `Main` para usar a senha de teste:

```

public class Program
{
    public static void Main(string[] args)
    {
        var host = CreateWebHostBuilder(args).Build();

        using (var scope = host.Services.CreateScope())
        {
            var services = scope.ServiceProvider;
            var context = services.GetRequiredService<ApplicationDbContext>();
            context.Database.Migrate();

            // requires using Microsoft.Extensions.Configuration;
            var config = host.Services.GetRequiredService< IConfiguration>();
            // Set password with the Secret Manager tool.
            // dotnet user-secrets set SeedUserPW <pw>

            var testUserPw = config["SeedUserPW"];
            try
            {
                SeedData.Initialize(services, testUserPw).Wait();
            }
            catch (Exception ex)
            {
                var logger = services.GetRequiredService< ILogger<Program>>();
                logger.LogError(ex.Message, "An error occurred seeding the DB.");
            }
        }

        host.Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>();
}

```

## Criar as contas de teste e atualizar os contatos

Atualize o método `Initialize` na classe `SeedData` para criar as contas de teste:

```
public static async Task Initialize(IServiceProvider serviceProvider, string testUserPw)
{
    using (var context = new ApplicationDbContext(
        serviceProvider.GetRequiredService<DbContextOptions<ApplicationDbContext>>()))
    {
        // For sample purposes seed both with the same password.
        // Password is set with the following:
        // dotnet user-secrets set SeedUserPw <pw>
        // The admin user can do anything

        var adminID = await EnsureUser(serviceProvider, testUserPw, "admin@contoso.com");
        await EnsureRole(serviceProvider, adminID, Constants.ContactAdministratorsRole);

        // allowed user can create and edit contacts that they create
        var managerID = await EnsureUser(serviceProvider, testUserPw, "manager@contoso.com");
        await EnsureRole(serviceProvider, managerID, Constants.ContactManagersRole);

        SeedDB(context, adminID);
    }
}

private static async Task<string> EnsureUser(IServiceProvider serviceProvider,
                                              string testUserPw, string UserName)
{
    var userManager = serviceProvider.GetService<UserManager<IdentityUser>>();

    var user = await userManager.FindByNameAsync(UserName);
    if (user == null)
    {
        user = new IdentityUser { UserName = UserName };
        await userManager.CreateAsync(user, testUserPw);
    }

    return user.Id;
}

private static async Task<IdentityResult> EnsureRole(IServiceProvider serviceProvider,
                                                       string uid, string role)
{
    IdentityResult IR = null;
    var roleManager = serviceProvider.GetService<RoleManager<IdentityRole>>();

    if (roleManager == null)
    {
        throw new Exception("roleManager null");
    }

    if (!await roleManager.RoleExistsAsync(role))
    {
        IR = await roleManager.CreateAsync(new IdentityRole(role));
    }

    var userManager = serviceProvider.GetService<UserManager<IdentityUser>>();

    var user = await userManager.FindByIdAsync(uid);

    IR = await userManager.AddToRoleAsync(user, role);

    return IR;
}
```

Adicione a ID de usuário de administrador e `ContactStatus` aos contatos. Torne um dos contatos "Enviado" e um

"Rejeitado". Adicione a ID de usuário e o status para todos os contatos. Somente um contato é exibido:

```
public static void SeedDB(ApplicationDbContext context, string adminID)
{
    if (context.Contact.Any())
    {
        return; // DB has been seeded
    }

    context.Contact.AddRange(
        new Contact
        {
            Name = "Debra Garcia",
            Address = "1234 Main St",
            City = "Redmond",
            State = "WA",
            Zip = "10999",
            Email = "debra@example.com",
            Status = ContactStatus.Approved,
            OwnerID = adminID
        },
    );
}
```

## Criar o proprietário, manager e manipuladores de autorização do administrador

Criar uma classe `ContactIsOwnerAuthorizationHandler` na pasta *autorização*. O `ContactIsOwnerAuthorizationHandler` verifica que o usuário que atua em um recurso possui o recurso.

```

using System.Threading.Tasks;
using ContactManager.Data;
using ContactManager.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Authorization.Infrastructure;
using Microsoft.AspNetCore.Identity;

namespace ContactManager.Authorization
{
    public class ContactIsOwnerAuthorizationHandler
        : AuthorizationHandler<OperationAuthorizationRequirement, Contact>
    {
        UserManager<IdentityUser> _userManager;

        public ContactIsOwnerAuthorizationHandler(UserManager<IdentityUser>
            userManager)
        {
            _userManager = userManager;
        }

        protected override Task
            HandleRequirementAsync(AuthorizationHandlerContext context,
                OperationAuthorizationRequirement requirement,
                Contact resource)
        {
            if (context.User == null || resource == null)
            {
                // Return Task.FromResult(0) if targeting a version of
                // .NET Framework older than 4.6:
                return Task.CompletedTask;
            }

            // If we're not asking for CRUD permission, return.

            if (requirement.Name != Constants.CreateOperationName &&
                requirement.Name != Constants.ReadOperationName &&
                requirement.Name != Constants.UpdateOperationName &&
                requirement.Name != Constants.DeleteOperationName )
            {
                return Task.CompletedTask;
            }

            if (resource.OwnerID == _userManager.GetUserId(context.User))
            {
                context.Succeed(requirement);
            }

            return Task.CompletedTask;
        }
    }
}

```

O `ContactIsOwnerAuthorizationHandler` chamadas `contexto.Êxito` se o usuário autenticado atual for o proprietário do contato. Manipuladores de autorização geralmente:

- Retornar `context.Succeed` quando os requisitos sejam atendidos.
- Retornar `Task.CompletedTask` quando os requisitos não forem atendidos. `Task.CompletedTask` não é sucesso ou falha—permite que outros manipuladores de autorização executar.

Se você precisar fazer explicitamente, retornar `contexto.Falha`.

O aplicativo permite que proprietários de contato possam editar/excluir/criar seus dados.

`ContactIsOwnerAuthorizationHandler` não precisa verificar a operação passada no parâmetro de requisito.

## Criar um manipulador de autorização do Gerenciador

Criar uma classe `ContactManagerAuthorizationHandler` na pasta *autorização*. O `ContactManagerAuthorizationHandler` verifica o usuário atuando no recurso é um gerente. Somente gerentes possam aprovar ou rejeitar as alterações de conteúdo (novas ou alteradas).

```
using System.Threading.Tasks;
using ContactManager.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Authorization.Infrastructure;
using Microsoft.AspNetCore.Identity;

namespace ContactManager.Authorization
{
    public class ContactManagerAuthorizationHandler :
        AuthorizationHandler<OperationAuthorizationRequirement, Contact>
    {
        protected override Task
            HandleRequirementAsync(AuthorizationHandlerContext context,
                OperationAuthorizationRequirement requirement,
                Contact resource)
        {
            if (context.User == null || resource == null)
            {
                return Task.CompletedTask;
            }

            // If not asking for approval/reject, return.
            if (requirement.Name != Constants.ApproveOperationName &&
                requirement.Name != Constants.RejectOperationName)
            {
                return Task.CompletedTask;
            }

            // Managers can approve or reject.
            if (context.User.IsInRole(Constants.ContactManagersRole))
            {
                context.Succeed(requirement);
            }

            return Task.CompletedTask;
        }
    }
}
```

## Crie um manipulador de autorização do administrador

Criar uma classe `ContactAdministratorsAuthorizationHandler` na pasta *autorização*. O `ContactAdministratorsAuthorizationHandler` verifica se o usuário atuando no recurso é um administrador. Administrador pode fazer todas as operações.

```

using System.Threading.Tasks;
using ContactManager.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Authorization.Infrastructure;

namespace ContactManager.Authorization
{
    public class ContactAdministratorsAuthorizationHandler
        : AuthorizationHandler<OperationAuthorizationRequirement, Contact>
    {
        protected override Task HandleRequirementAsync(
            AuthorizationHandlerContext context,
            OperationAuthorizationRequirement requirement,
            Contact resource)
        {
            if (context.User == null)
            {
                return Task.CompletedTask;
            }

            // Administrators can do anything.
            if (context.User.IsInRole(Constants.ContactAdministratorsRole))
            {
                context.Succeed(requirement);
            }

            return Task.CompletedTask;
        }
    }
}

```

## O registro dos manipuladores de autorização

Serviços usando o Entity Framework Core devem ser registrados [injeção de dependência](#) usando `AddScoped`. O `ContactIsOwnerAuthorizationHandler` usa o ASP.NET Core [identidade](#), que se baseia no Entity Framework Core. O registro dos manipuladores com a coleção de serviço para que eles estejam disponíveis para o `ContactsController` por meio [injeção de dependência](#). Adicione o seguinte código ao final da `ConfigureServices` :

```

public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));
    services.AddDefaultIdentity<IdentityUser>().AddRoles<IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>();

    services.AddMvc(config =>
    {
        // using Microsoft.AspNetCore.Mvc.Authorization;
        // using Microsoft.AspNetCore.Authorization;
        var policy = new AuthorizationPolicyBuilder()
            .RequireAuthenticatedUser()
            .Build();
        config.Filters.Add(new AuthorizeFilter(policy));
    })
        .SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    // Authorization handlers.
    services.AddScoped<IAuthorizationHandler,
        ContactIsOwnerAuthorizationHandler>();

    services.AddSingleton<IAuthorizationHandler,
        ContactAdministratorsAuthorizationHandler>();

    services.AddSingleton<IAuthorizationHandler,
        ContactManagerAuthorizationHandler>();
}

```

`ContactAdministratorsAuthorizationHandler` e `ContactManagerAuthorizationHandler` são adicionados como singletons. Eles são singletons porque eles não usam o EF e todas as informações necessárias na `Context` parâmetro do `HandleRequirementAsync` método.

## Supporte à autorização

Nesta seção, você atualize as páginas Razor e adicione uma classe de requisitos de operações.

### Examine a classe de requisitos de operações de contato

Examine o `ContactOperations` classe. Essa classe contém os requisitos de aplicativo dá suporte:

```

using Microsoft.AspNetCore.Authorization.Infrastructure;

namespace ContactManager.Authorization
{
    public static class ContactOperations
    {
        public static OperationAuthorizationRequirement Create =
            new OperationAuthorizationRequirement {Name=Constants.CreateOperationName};
        public static OperationAuthorizationRequirement Read =
            new OperationAuthorizationRequirement {Name=Constants.ReadOperationName};
        public static OperationAuthorizationRequirement Update =
            new OperationAuthorizationRequirement {Name=Constants.UpdateOperationName};
        public static OperationAuthorizationRequirement Delete =
            new OperationAuthorizationRequirement {Name=Constants.DeleteOperationName};
        public static OperationAuthorizationRequirement Approve =
            new OperationAuthorizationRequirement {Name=Constants.ApproveOperationName};
        public static OperationAuthorizationRequirement Reject =
            new OperationAuthorizationRequirement {Name=Constants.RejectOperationName};
    }

    public class Constants
    {
        public static readonly string CreateOperationName = "Create";
        public static readonly string ReadOperationName = "Read";
        public static readonly string UpdateOperationName = "Update";
        public static readonly string DeleteOperationName = "Delete";
        public static readonly string ApproveOperationName = "Approve";
        public static readonly string RejectOperationName = "Reject";

        public static readonly string ContactAdministratorsRole =
            "ContactAdministrators";
        public static readonly string ContactManagersRole = "ContactManagers";
    }
}

```

## Criar uma classe base para as páginas do Razor de contatos

Crie uma classe base que contém os serviços usados em contatos páginas do Razor. A classe base coloca o código de inicialização em um único local:

```

using ContactManager.Data;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace ContactManager.Pages.Contacts
{
    public class DI_BasePageModel : PageModel
    {
        protected ApplicationDbContext Context { get; }
        protected IAuthorizationService AuthorizationService { get; }
        protected UserManager<IdentityUser> UserManager { get; }

        public DI_BasePageModel(
            ApplicationDbContext context,
            IAuthorizationService authorizationService,
            UserManager<IdentityUser> userManager) : base()
        {
            Context = context;
            UserManager = userManager;
            AuthorizationService = authorizationService;
        }
    }
}

```

O código anterior:

- Adiciona o serviço `IAuthorizationService` para acessar os manipuladores de autorização.
- Adiciona o serviço de identidade `UserManager`.
- Adicione a `ApplicationDbContext`.

## Atualizar o CreateModel

Atualize o construtor de criar modelo de página para usar a classe base `DI_BasePageModel`:

```
public class CreateModel : DI_BasePageModel
{
    public CreateModel(
        ApplicationDbContext context,
        IAuthorizationService authorizationService,
        UserManager<IdentityUser> userManager)
        : base(context, authorizationService, userManager)
    {
    }
}
```

Atualize o método `CreateModel.OnPostAsync` para:

- Adicione a ID de usuário ao modelo `Contact`.
- Chame o manipulador de autorização para verificar se que o usuário tem permissão para criar contatos.

```
public async Task<IActionResult> OnPostAsync()
{
    if (!ModelState.IsValid)
    {
        return Page();
    }

    Contact.OwnerID = UserManager.GetUserId(User);

    // requires using ContactManager.Authorization;
    var isAuthorized = await AuthorizationService.AuthorizeAsync(
        User, Contact,
        ContactOperations.Create);
    if (!isAuthorized.Succeeded)
    {
        return new ChallengeResult();
    }

    Context.Contact.Add(Contact);
    await Context.SaveChangesAsync();

    return RedirectToAction("./Index");
}
```

## Atualizar o IndexModel

Atualize o método `OnGetAsync` para que apenas os contatos aprovados sejam mostrados aos usuários gerais:

```

public class IndexModel : DI_BasePageModel
{
    public IndexModel(
        ApplicationDbContext context,
        IAuthorizationService authorizationService,
        UserManager<IdentityUser> userManager)
        : base(context, authorizationService, userManager)
    {
    }

    public IList<Contact> Contact { get; set; }

    public async Task OnGetAsync()
    {
        var contacts = from c in Context.Contact
                      select c;

        var isAuthorized = User.IsInRole(Constants.ContactManagersRole) ||
                           User.IsInRole(Constants.ContactAdministratorsRole);

        var currentUserID = UserManager.GetUserId(User);

        // Only approved contacts are shown UNLESS you're authorized to see them
        // or you are the owner.
        if (!isAuthorized)
        {
            contacts = contacts.Where(c => c.Status == ContactStatus.Approved
                                      || c.OwnerID == currentUserID);
        }

        Contact = await contacts.ToListAsync();
    }
}

```

## Atualizar o EditModel

Adicione um manipulador de autorização para verificar se que o usuário é proprietária do contato. Como a autorização de recursos está sendo validada, o `[Authorize]` atributo não é suficiente. O aplicativo não tiver acesso ao recurso quando atributos são avaliados. Autorização baseada em recursos deve ser imperativa. As verificações devem ser executadas depois que o aplicativo tem acesso ao recurso, carregá-los no modelo de página ou carregá-los dentro do manipulador em si. Com frequência, você acessa o recurso, passando a chave de recurso.

```

public class EditModel : DI_BasePageModel
{
    public EditModel(
        ApplicationDbContext context,
        IAuthorizationService authorizationService,
        UserManager<IdentityUser> userManager)
        : base(context, authorizationService, userManager)
    {
    }

    [BindProperty]
    public Contact Contact { get; set; }

    public async Task<IActionResult> OnGetAsync(int id)
    {
        Contact = await Context.Contact.FirstOrDefaultAsync(
            m => m.ContactId == id);

        if (Contact == null)
        {
            return NotFound();
        }
    }
}

```

```

        var isAuthorized = await AuthorizationService.AuthorizeAsync(
            User, Contact,
            ContactOperations.Update);
        if (!isAuthorized.Succeeded)
        {
            return new ChallengeResult();
        }

        return Page();
    }

    public async Task<IActionResult> OnPostAsync(int id)
    {
        if (!ModelState.IsValid)
        {
            return Page();
        }

        // Fetch Contact from DB to get OwnerID.
        var contact = await Context
            .Contact.AsNoTracking()
            .FirstOrDefaultAsync(m => m.ContactId == id);

        if (contact == null)
        {
            return NotFound();
        }

        var isAuthorized = await AuthorizationService.AuthorizeAsync(
            User, contact,
            ContactOperations.Update);
        if (!isAuthorized.Succeeded)
        {
            return new ChallengeResult();
        }

        Contact.OwnerID = contact.OwnerID;

        Context.Attach(Contact).State = EntityState.Modified;

        if (contact.Status == ContactStatus.Approved)
        {
            // If the contact is updated after approval,
            // and the user cannot approve,
            // set the status back to submitted so the update can be
            // checked and approved.
            var canApprove = await AuthorizationService.AuthorizeAsync(User,
                contact,
                ContactOperations.Approve);

            if (!canApprove.Succeeded)
            {
                contact.Status = ContactStatus.Submitted;
            }
        }

        await Context.SaveChangesAsync();

        return RedirectToPage("./Index");
    }

    private bool ContactExists(int id)
    {
        return Context.Contact.Any(e => e.ContactId == id);
    }
}

```

## Atualizar o DeleteModel

Atualize o modelo de página de exclusão para usar o manipulador de autorização para verificar se o usuário tem permissão de exclusão no contato.

```
public class DeleteModel : DI_BasePageModel
{
    public DeleteModel(
        ApplicationDbContext context,
        IAuthorizationService authorizationService,
        UserManager<IdentityUser> userManager)
        : base(context, authorizationService, userManager)
    {
    }

    [BindProperty]
    public Contact Contact { get; set; }

    public async Task<IActionResult> OnGetAsync(int id)
    {
        Contact = await Context.Contact.FirstOrDefaultAsync(
            m => m.ContactId == id);

        if (Contact == null)
        {
            return NotFound();
        }

        var isAuthorized = await AuthorizationService.AuthorizeAsync(
            User, Contact,
            ContactOperations.Delete);

        if (!isAuthorized.Succeeded)
        {
            return new ChallengeResult();
        }

        return Page();
    }

    public async Task<IActionResult> OnPostAsync(int id)
    {
        Contact = await Context.Contact.FindAsync(id);

        var contact = await Context
            .Contact.AsNoTracking()
            .FirstOrDefaultAsync(m => m.ContactId == id);

        if (contact == null)
        {
            return NotFound();
        }

        var isAuthorized = await AuthorizationService.AuthorizeAsync(
            User, contact,
            ContactOperations.Delete);

        if (!isAuthorized.Succeeded)
        {
            return new ChallengeResult();
        }

        Context.Contact.Remove(Contact);
        await Context.SaveChangesAsync();

        return RedirectToPage("./Index");
    }
}
```

## Injetar o serviço de autorização em exibições

Atualmente, mostra a interface do usuário a editar e excluir links para os contatos, que o usuário não é possível modificar.

Injetar o serviço de autorização no arquivo `Views/_ViewImports.cshtml` para que ele esteja disponível para todos os modos de exibição:

```
@using Microsoft.AspNetCore.Identity  
@using ContactManager  
@using ContactManager.Data  
@namespace ContactManager.Pages  
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers  
@using ContactManager.Authorization;  
@using Microsoft.AspNetCore.Authorization  
@using ContactManager.Models  
@inject IAuthorizationService AuthorizationService
```

A marcação anterior adiciona várias `using` instruções.

Atualizar o **editar** e **excluir** vincula na `Pages/Contacts/Index.cshtml` para que eles sejam renderizados somente para usuários com as permissões apropriadas:

```
@page  
@model ContactManager.Pages.Contacts.IndexModel  
  
{@  
    ViewData["Title"] = "Index";  
}  
  
<h2>Index</h2>  
  
<p>  
    <a asp-page="Create">Create New</a>  
</p>  
<table class="table">  
    <thead>  
        <tr>  
            <th>  
                @Html.DisplayNameFor(model => model.Contact[0].Name)  
            </th>  
            <th>  
                @Html.DisplayNameFor(model => model.Contact[0].Address)  
            </th>  
            <th>  
                @Html.DisplayNameFor(model => model.Contact[0].City)  
            </th>  
            <th>  
                @Html.DisplayNameFor(model => model.Contact[0].State)  
            </th>  
            <th>  
                @Html.DisplayNameFor(model => model.Contact[0].Zip)  
            </th>  
            <th>  
                @Html.DisplayNameFor(model => model.Contact[0].Email)  
            </th>  
            <th>  
                @Html.DisplayNameFor(model => model.Contact[0].Status)  
            </th>  
            <th></th>  
        </tr>  
    </thead>  
    <tbody>  
        @foreach (var item in Model.Contact)  
    ,
```

```

    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Name)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Address)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.City)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.State)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Zip)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Email)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Status)
        </td>
        <td>
            @if ((await AuthorizationService.AuthorizeAsync(
                User, item,
                ContactOperations.Update)).Succeeded)
            {
                <a asp-page=".Edit" asp-route-id="@item.ContactId">Edit</a>
                <text> | </text>
            }
            <a asp-page=".Details" asp-route-id="@item.ContactId">Details</a>
            @if ((await AuthorizationService.AuthorizeAsync(
                User, item,
                ContactOperations.Delete)).Succeeded)
            {
                <text> | </text>
                <a asp-page=".Delete" asp-route-id="@item.ContactId">Delete</a>
            }
        </td>
    </tr>
}
</tbody>
</table>

```

### WARNING

Ocultar links de usuários que não tem permissão para alterar os dados não proteger o aplicativo. Ocultar links torna o aplicativo mais fácil de usar, exibindo links só é válidas. Os usuários podem gerar URLs para invocar editar e excluir operações em dados que não possuem. A página do Razor ou controlador deve impor verificações de acesso para proteger os dados.

### Detalhes da atualização

Atualize a exibição de detalhes para que os gerentes possam aprovar ou rejeitar contatos:

```

    /*Precedng markup omitted for brevity.*@
    <dt>
        @Html.DisplayNameFor(model => model.Contact.Email)
    </dt>
    <dd>
        @Html.DisplayFor(model => model.Contact.Email)
    </dd>
    <dt>
        @Html.DisplayNameFor(model => model.Contact.Status)
    </dt>
    <dd>
        @Html.DisplayFor(model => model.Contact.Status)
    </dd>
</dl>
</div>

@if (Model.Contact.Status != ContactStatus.Approved)
{
    @if ((await AuthorizationService.AuthorizeAsync(
        User, Model.Contact, ContactOperations.Approve)).Succeeded)
    {
        <form style="display:inline;" method="post">
            <input type="hidden" name="id" value="@Model.Contact.ContactId" />
            <input type="hidden" name="status" value="@ContactStatus.Approved" />
            <button type="submit" class="btn btn-xs btn-success">Approve</button>
        </form>
    }
}

@if (Model.Contact.Status != ContactStatus.Rejected)
{
    @if ((await AuthorizationService.AuthorizeAsync(
        User, Model.Contact, ContactOperations.Reject)).Succeeded)
    {
        <form style="display:inline;" method="post">
            <input type="hidden" name="id" value="@Model.Contact.ContactId" />
            <input type="hidden" name="status" value="@ContactStatus.Rejected" />
            <button type="submit" class="btn btn-xs btn-success">Reject</button>
        </form>
    }
}

<div>
    @if ((await AuthorizationService.AuthorizeAsync(
        User, Model.Contact,
        ContactOperations.Update)).Succeeded)
    {
        <a asp-page="./Edit" asp-route-id="@Model.Contact.ContactId">Edit</a>
        <text> | </text>
    }
    <a asp-page="./Index">Back to List</a>
</div>

```

Atualize o modelo de página de detalhes:

```

public class DetailsModel : DI_BasePageModel
{
    public DetailsModel(
        ApplicationDbContext context,
        IAuthorizationService authorizationService,
        UserManager<IdentityUser> userManager)
        : base(context, authorizationService, userManager)
    {
    }

    public Contact Contact { get; set; }

    public async Task<IActionResult> OnGetAsync(int id)
    {
        Contact = await Context.Contact.FirstOrDefaultAsync(m => m.ContactId == id);

        if (Contact == null)
        {
            return NotFound();
        }

        var isAuthorized = User.IsInRole(Constants.ContactManagersRole) ||
                           User.IsInRole(Constants.ContactAdministratorsRole);

        var currentUserID = UserManager.GetUserId(User);

        if (!isAuthorized
            && currentUserID != Contact.OwnerID
            && Contact.Status != ContactStatus.Approved)
        {
            return new ChallengeResult();
        }

        return Page();
    }

    public async Task<IActionResult> OnPostAsync(int id, ContactStatus status)
    {
        var contact = await Context.Contact.FirstOrDefaultAsync(
            m => m.ContactId == id);

        if (contact == null)
        {
            return NotFound();
        }

        var contactOperation = (status == ContactStatus.Approved)
            ? ContactOperations.Approve
            : ContactOperations.Reject;

        var isAuthorized = await AuthorizationService.AuthorizeAsync(User, contact,
            contactOperation);
        if (!isAuthorized.Succeeded)
        {
            return new ChallengeResult();
        }
        contact.Status = status;
        Context.Contact.Update(contact);
        await Context.SaveChangesAsync();

        return RedirectToPage("./Index");
    }
}

```

Adicionar ou remover um usuário a uma função

Ver [esse problema](#) para obter informações sobre:

- Remoção de privilégios de um usuário. Por exemplo silenciamos um usuário em um aplicativo de bate-papo.
- Adicionando privilégios a um usuário.

## Testar o aplicativo concluído

Se você ainda não tiver configurado uma senha para contas de usuário propagados, use o [ferramenta Secret Manager](#) para definir uma senha:

- Escolha uma senha forte: Usar oito ou mais caracteres e pelo menos um caractere maiúsculo, número e símbolo. Por exemplo, `Passw0rd!` atende aos requisitos de senha forte.
- Execute o seguinte comando na pasta do projeto, onde `<PW>` é a senha:

```
dotnet user-secrets set SeedUserPW <PW>
```

Se o aplicativo tem contatos:

- Excluir todos os registros no `Contact` tabela.
- Reinicie o aplicativo para propagar o banco de dados.

Uma maneira fácil de testar o aplicativo concluído é iniciar três diferentes navegadores (ou incógnita/InPrivate sessões). Em um navegador, registre um novo usuário (por exemplo, `test@example.com`). Entrar para cada navegador com um usuário diferente. Verifique se as seguintes operações:

- Usuários registrados podem exibir todos os dados de contato aprovados.
- Os usuários registrados podem editar/excluir seus próprios dados.
- Os gerentes podem Aprovar/rejeitar dados de contato. A tela `Details` mostra os botões **aprovar** e **rejeitar**.
- Os administradores podem Aprovar/rejeitar e editar/excluir todos os dados.

USER	PROPAGADA PELO APLICATIVO	OPÇÕES
test@example.com	Não	Editar/Excluir os próprios dados.
manager@contoso.com	Sim	Aprovar/rejeitar e editar/excluir os próprios dados.
admin@contoso.com	Sim	Aprovar/rejeitar e editar/excluir todos os dados.

Crie um contato no navegador do administrador. Copie a URL para excluir e editar a partir do contato do administrador. Cole esses links no navegador do usuário de teste para verificar se o usuário de teste não é possível executar essas operações.

## Criar o aplicativo inicial

- Criar um aplicativo páginas Razor denominado "ContactManager"
  - Criar o aplicativo com **contas de usuário individuais**.
  - O nome "ContactManager" para o namespace coincide com o namespace usado no exemplo.
  - `-uld` Especifica o LocalDB em vez do SQLite

```
dotnet new webapp -o ContactManager -au Individual -uld
```

- Adicione `Models/Contact.cs`:

```
public class Contact
{
    public int ContactId { get; set; }
    public string Name { get; set; }
    public string Address { get; set; }
    public string City { get; set; }
    public string State { get; set; }
    public string Zip { get; set; }
    [DataType(DataType.EmailAddress)]
    public string Email { get; set; }
}
```

- Scaffold o `Contact` modelo.
- Criar a migração inicial e atualizar o banco de dados:

```
dotnet aspnet-codegenerator razorpage -m Contact -udl -dc ApplicationDbContext -outDir Pages\Contacts -
-referenceScriptLibraries
dotnet ef database drop -f
dotnet ef migrations add initial
dotnet ef database update
```

- Atualizar o link de **ContactManager** no arquivo `Pages/_Layout.cshtml`:

```
<a asp-page="/Contacts/Index" class="navbar-brand">ContactManager</a>
```

- Testar o aplicativo criando, editando e excluindo um contato

## Propagar o banco de dados

Adicione a `SeedData` de classe para o *dados* pasta.

Chame `SeedData.Initialize` de `Main`:

```

public class Program
{
    public static void Main(string[] args)
    {
        var host = CreateWebHostBuilder(args).Build();

        using (var scope = host.Services.CreateScope())
        {
            var services = scope.ServiceProvider;

            try
            {
                var context = services.GetRequiredService<ApplicationDbContext>();
                context.Database.Migrate();
                SeedData.Initialize(services, "not used");
            }
            catch (Exception ex)
            {
                var logger = services.GetRequiredService<ILogger<Program>>();
                logger.LogError(ex, "An error occurred seeding the DB.");
            }
        }

        host.Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>();
}

```

Se o aplicativo propagado o banco de dados de teste. Se não houver nenhuma linha no banco de dados de contato, o método de propagação não será executado.

## Recursos adicionais

- [Criar um aplicativo web .NET Core e o banco de dados SQL no serviço de aplicativo do Azure](#)
- [Laboratório de autorização do ASP.NET Core](#). Este laboratório apresenta mais detalhes sobre os recursos de segurança introduzidos neste tutorial.
- [Introdução à autorização no núcleo do ASP.NET](#)
- [Autorização baseada em política personalizada](#)

# Convenções de autorização de páginas do Razor no ASP.NET Core

30/10/2018 • 6 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

Uma maneira de controlar o acesso em seu aplicativo de páginas do Razor é usar as convenções de autorização na inicialização. Essas convenções permitem que você autorizar usuários e permitir que usuários anônimos acessem páginas individuais ou pastas de páginas. Aplicam as convenções descritas neste tópico automaticamente [filtros de autorização](#) para controlar o acesso.

## [Exibir ou baixar código de exemplo \(como baixar\)](#)

O aplicativo de exemplo usa [autenticação de Cookie sem o ASP.NET Core Identity](#). A conta de usuário para o usuário hipotético, Maria Rodriguez, é codificados no aplicativo. Use o nome de usuário de Email "maria.rodriguez@contoso.com" e nenhuma senha para a entrada do usuário. O usuário é autenticado na `AuthenticateUser` método na `Pages/Account/Login.cshtml.cs` arquivo. Em um exemplo do mundo real, o usuário deve ser autenticado em relação a um banco de dados. Para usar o ASP.NET Core Identity, siga as diretrizes a [Introdução à identidade no ASP.NET Core](#) tópico. Os conceitos e os exemplos mostrados neste tópico se aplicam igualmente a aplicativos que usam o ASP.NET Core Identity.

## Exigir autorização para acessar uma página

Use o [AuthorizePage](#) convenção via [AddRazorPagesOptions](#) para adicionar um [AuthorizeFilter](#) para a página no caminho especificado:

```
services.AddMvc()
    .AddRazorPagesOptions(options =>
{
    options.Conventions.AuthorizePage("/Contact");
    options.Conventions.AuthorizeFolder("/Private");
    options.Conventions.AllowAnonymousToPage("/Private/PublicPage");
    options.Conventions.AllowAnonymousToFolder("/Private/PublicPages");
})
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
```

O caminho especificado é o caminho de mecanismo de exibição, que é o caminho relativo de raiz de páginas do Razor sem uma extensão e que contém apenas barras "/".

Uma [AuthorizePage sobrecarga](#) estará disponível se você precisa especificar uma política de autorização.

### NOTE

Uma `AuthorizeFilter` pode ser aplicado a uma classe de modelo de página com o `[Authorize]` atributo de filtro. Para obter mais informações, consulte [atributo de filtro Authorize](#).

## Exigir autorização para acessar uma pasta de páginas

Use o [AuthorizeFolder](#) convenção via [AddRazorPagesOptions](#) para adicionar um [AuthorizeFilter](#) para todas as páginas em uma pasta no caminho especificado:

```
services.AddMvc()
    .AddRazorPagesOptions(options =>
{
    options.Conventions.AuthorizePage("/Contact");
    options.Conventions.AuthorizeFolder("/Private");
    options.Conventions.AllowAnonymousToPage("/Private/PublicPage");
    options.Conventions.AllowAnonymousToFolder("/Private/PublicPages");
})
.SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
```

O caminho especificado é o caminho de mecanismo de exibição, que é o caminho relativo de raiz de páginas do Razor.

Uma [AuthorizeFolder sobrecarga](#) estará disponível se você precisa especificar uma política de autorização.

## Exigir autorização para acessar uma página de área

Use o [AuthorizeAreaPage](#) convenção via [AddRazorPagesOptions](#) para adicionar um [AuthorizeFilter](#) para a página de área no caminho especificado:

```
options.Conventions.AuthorizeAreaPage("Identity", "/Manage/Accounts");
```

O nome da página é o caminho do arquivo sem uma extensão relativo ao diretório raiz páginas para a área especificada. Por exemplo, o nome da página para o arquivo *Areas/Identity/Pages/Manage/Accounts.cshtml* é *contas/gerenciar*.

Uma [AuthorizeAreaPage sobrecarga](#) estará disponível se você precisa especificar uma política de autorização.

## Exigir autorização para acessar uma pasta de áreas

Use o [AuthorizeAreaFolder](#) convenção via [AddRazorPagesOptions](#) para adicionar um [AuthorizeFilter](#) para todas as áreas em uma pasta no caminho especificado:

```
options.Conventions.AuthorizeAreaFolder("Identity", "/Manage");
```

O caminho da pasta é o caminho da pasta, relativo ao diretório raiz páginas para a área especificada. Por exemplo, o caminho da pasta para os arquivos sob *áreas/identidade/páginas/gerenciar/* é */gerenciar*.

Uma [AuthorizeAreaFolder sobrecarga](#) estará disponível se você precisa especificar uma política de autorização.

## Permitir o acesso anônimo para uma página

Use o [AllowAnonymousToPage](#) convenção via [AddRazorPagesOptions](#) para adicionar um [AllowAnonymousFilter](#) para uma página no caminho especificado:

```
services.AddMvc()
    .AddRazorPagesOptions(options =>
{
    options.Conventions.AuthorizePage("/Contact");
    options.Conventions.AuthorizeFolder("/Private");
    options.Conventions.AllowAnonymousToPage("/Private/PublicPage");
    options.Conventions.AllowAnonymousToFolder("/Private/PublicPages");
})
.SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
```

O caminho especificado é o caminho de mecanismo de exibição, que é o caminho relativo de raiz de páginas do

Razor sem uma extensão e que contém apenas barras "/".

## Permitir o acesso anônimo para uma pasta de páginas

Use o [AllowAnonymousToFolder](#) convenção via [AddRazorPagesOptions](#) para adicionar um [AllowAnonymousFilter](#) para todas as páginas em uma pasta no caminho especificado:

```
services.AddMvc()
    .AddRazorPagesOptions(options =>
{
    options.Conventions.AuthorizePage("/Contact");
    options.Conventions.AuthorizeFolder("/Private");
    options.Conventions.AllowAnonymousToPage("/Private/PublicPage");
    options.Conventions.AllowAnonymousToFolder("/Private/PublicPages");
})
.SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
```

O caminho especificado é o caminho de mecanismo de exibição, que é o caminho relativo de raiz de páginas do Razor.

## Observação sobre combinação autorizada e acesso anônimo

É perfeitamente válido para especificar que uma pasta de páginas exigem autorização e especificar uma página dentro da pasta permite o acesso anônimo:

```
// This works.
.AuthorizeFolder("/Private").AllowAnonymousToPage("/Private/Public")
```

O inverso, no entanto, não é verdadeiro. Você não pode declarar uma pasta de páginas para acesso anônimo e especificar uma página dentro de autorização:

```
// This doesn't work!
.AllowAnonymousToFolder("/Public").AuthorizePage("/Public/Private")
```

Exigir autorização na página privada não funcionará porque quando tanto o [AllowAnonymousFilter](#) e [AuthorizeFilter](#) filtros são aplicados para a página, o [AllowAnonymousFilter](#) wins e controla o acesso.

## Recursos adicionais

- [Provedores de modelo personalizado de página e rota de Páginas Razor](#)
- [PageConventionCollection classe](#)

# Simples de autorização no núcleo do ASP.NET

27/06/2018 • 2 minutes to read • [Edit Online](#)

No MVC é controlada por meio de `[AuthorizeAttribute]` atributo e seus vários parâmetros. Em sua forma mais simples, aplicando o `[AuthorizeAttribute]` de atributo para um controlador ou ação limites acesse o controlador ou ação para qualquer usuário autenticado.

Por exemplo, o código a seguir limita o acesso para o `AccountController` para qualquer usuário autenticado.

```
[Authorize]
public class AccountController : Controller
{
    public ActionResult Login()
    {

    }

    public ActionResult Logout()
    {
    }
}
```

Se você deseja aplicar a autorização para uma ação em vez do controlador, aplique a `[AuthorizeAttribute]` de atributo para a ação em si:

```
public class AccountController : Controller
{
    public ActionResult Login()
    {

    }

    [Authorize]
    public ActionResult Logout()
    {
    }
}
```

Agora, somente usuários autenticados podem acessar o `Logout` função.

Você também pode usar o `[AllowAnonymous]` atributo para permitir o acesso por usuários não autenticados para ações individuais. Por exemplo:

```
[Authorize]
public class AccountController : Controller
{
    [AllowAnonymous]
    public ActionResult Login()
    {

    }

    public ActionResult Logout()
    {
    }
}
```

Isso permitiria que somente usuários autenticados para o `AccountController`, exceto para o `Login` ação, que

pode ser acessada por todos os usuários, independentemente de seu status de autenticado ou anônimo / não autenticado.

**WARNING**

[AllowAnonymous] Ignora todas as declarações de autorização. Se você combinar [AllowAnonymous] e [Authorize] atributo, o [Authorize] atributos são ignorados. Por exemplo, se você aplicar [AllowAnonymous] no nível do controlador, qualquer [Authorize] atributos no mesmo controlador (ou em qualquer ação dentro dele) é ignorada.

# Autorização baseada em função no ASP.NET Core

31/07/2018 • 4 minutes to read • [Edit Online](#)

Quando uma identidade é criada ele pode pertencer a uma ou mais funções. Por exemplo, Tracy pode pertencer às funções de administrador e usuário, embora Scott só pode pertencer à função de usuário. Como essas funções são criadas e gerenciadas dependem do repositório de backup do processo de autorização. As funções são expostas para o desenvolvedor por meio de `IsInRole` método na `ClaimsPrincipal` classe.

## IMPORTANT

Este tópico **não** se aplica a Páginas Razor. Dá suporte a páginas do Razor `IPageFilter` e `IAsyncPageFilter`. Para obter mais informações, confira [Métodos de filtro para Páginas Razor](#).

## Adicionar verificações de função

Verificações de autorização baseado em função são declarativas—o desenvolvedor incorpora-los dentro de seu código, em relação a um controlador ou uma ação dentro de um controlador, especificando as funções que o usuário atual deve ser um membro de acessar o recurso solicitado.

Por exemplo, o código a seguir limita o acesso a quaisquer ações sobre o `AdministrationController` aos usuários que são membros do `Administrator` função:

```
[Authorize(Roles = "Administrator")]
public class AdministrationController : Controller
{}
```

Você pode especificar várias funções como uma lista separada por vírgulas:

```
[Authorize(Roles = "HRManager,Finance")]
public class SalaryController : Controller
{}
```

Esse controlador seria apenas ser acessado por usuários que são membros do `HRManager` função ou o `Finance` função.

Se você aplicar vários atributos de um usuário ao acessar deve ser um membro de todas as funções especificadas; o exemplo a seguir requer que um usuário deve ser um membro de ambos os `PowerUser` e `ControlPanelUser` função.

```
[Authorize(Roles = "PowerUser")]
[Authorize(Roles = "ControlPanelUser")]
public class ControlPanelController : Controller
{}
```

Você pode limitar o acesso ainda mais aplicando atributos de autorização de função adicionais em nível de ação:

```
[Authorize(Roles = "Administrator, PowerUser")]
public class ControlPanelController : Controller
{
    public ActionResult SetTime()
    {
    }

    [Authorize(Roles = "Administrator")]
    public ActionResult ShutDown()
    {
    }
}
```

Nos membros de trecho de código anterior do `Administrator` função ou o `PowerUser` função pode acessar o controlador e o `SetTime` ação, mas somente os membros dos `Administrator` função pode acessar o `ShutDown` ação.

Você também pode bloquear um controlador mas permitir o acesso anônimo, não autenticado a ações individuais.

```
[Authorize]
public class ControlPanelController : Controller
{
    public ActionResult SetTime()
    {

    }

    [AllowAnonymous]
    public ActionResult Login()
    {
    }
}
```

## Verificações de função baseada em política

Requisitos da função também podem ser expressos usando a nova sintaxe de diretiva, em que um desenvolvedor registra uma política na inicialização como parte da configuração do serviço de autorização. Isso normalmente ocorre `ConfigureServices()` em seu `Startup.cs` arquivo.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    services.AddAuthorization(options =>
    {
        options.AddPolicy("RequireAdministratorRole", policy => policy.RequireRole("Administrator"));
    });
}
```

As políticas são aplicadas usando o `Policy` propriedade no `AuthorizeAttribute` atributo:

```
[Authorize(Policy = "RequireAdministratorRole")]
public IActionResult Shutdown()
{
    return View();
}
```

Se você deseja especificar várias funções permitidas em um requisito, você pode especificá-los como parâmetros para o `RequireRole` método:

```
options.AddPolicy("ElevatedRights", policy =>
    policy.RequireRole("Administrator", "PowerUser", "BackupAdministrator"));
```

Este exemplo autoriza usuários que pertencem à `Administrator`, `PowerUser` OU `BackupAdministrator` funções.

# Autorização baseada em declarações no núcleo do ASP.NET

22/06/2018 • 6 minutes to read • [Edit Online](#)

Quando uma identidade é criada ele pode ser atribuído uma ou mais declarações emitidas por um terceiro confiável. Uma declaração é um par nome-valor que representa o assunto, não que a entidade pode fazer. Por exemplo, você pode ter de motorista uma carteira, emitida por uma autoridade de licença de um local. Licença do driver tem sua data de nascimento. Nesse caso seria o nome da declaração `DateOfBirth`, o valor da declaração seria sua data de nascimento, por exemplo `8th June 1970` e o emissor de um autoridade de licença. Autorização baseada em declarações, em sua forma mais simples, verifica o valor de uma declaração e permite o acesso a um recurso com base no valor. Por exemplo, se você quiser que o processo de autorização de acesso para uma sociedade noite poderia ser:

A analista de segurança de porta deve avaliar o valor da sua data de nascimento declaração e se elas têm confiança do emissor (de um autoridade de licença) antes de conceder a que você acessar.

Uma identidade pode conter várias declarações com vários valores e pode conter várias declarações do mesmo tipo.

## Adicionar declarações verificações

Declaração de verificações de autorização com base são declarativas - o desenvolvedor incorpora dentro de seu código, em relação a um controlador ou uma ação dentro de um controlador, especificando que o usuário atual deve ter, e, opcionalmente, o valor de declaração deve conter para acessar o recurso solicitado. Declarações de requisitos são baseada em política, o desenvolvedor deve criar e registrar uma política de expressar os requisitos de declarações.

O tipo mais simples de política procura a presença de uma declaração de declaração e não verifica o valor.

Primeiro você precisa criar e registrar a política. Isso ocorre como parte da configuração do serviço de autorização, que normalmente faz parte do `ConfigureServices()` no seu `Startup.cs` arquivo.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    services.AddAuthorization(options =>
    {
        options.AddPolicy("EmployeeOnly", policy => policy.RequireClaim("EmployeeNumber"));
    });
}
```

Nesse caso o `EmployeeOnly` política verifica a presença de um `EmployeeNumber` de declaração de identidade atual.

Em seguida, aplique a política usando o `Policy` propriedade no `AuthorizeAttribute` atributo para especificar o nome da política

```
[Authorize(Policy = "EmployeeOnly")]
public IActionResult VacationBalance()
{
    return View();
}
```

O `AuthorizeAttribute` atributo pode ser aplicado a um controlador de inteiro, nesta instância, somente a política de correspondência de identidades terão acesso permitidas para qualquer ação no controlador.

```
[Authorize(Policy = "EmployeeOnly")]
public class VacationController : Controller
{
    public ActionResult VacationBalance()
    {
    }
}
```

Se você tiver um controlador que é protegido pelo `AuthorizeAttribute` de atributo, mas deseja permitir acesso anônimo a ações específicas que você aplicar o `AllowAnonymousAttribute` atributo.

```
[Authorize(Policy = "EmployeeOnly")]
public class VacationController : Controller
{
    public ActionResult VacationBalance()
    {
    }

    [AllowAnonymous]
    public ActionResult VacationPolicy()
    {
    }
}
```

A maioria das declarações vem com um valor. Você pode especificar uma lista de valores permitido ao criar a política. O exemplo a seguir seria êxito apenas para os funcionários cujo número de funcionário foi 1, 2, 3, 4 ou 5.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    services.AddAuthorization(options =>
    {
        options.AddPolicy("Founders", policy =>
            policy.RequireClaim("EmployeeNumber", "1", "2", "3", "4", "5"));
    });
}
```

### Adicionar uma verificação de declaração genérico

Se o valor da declaração não é um valor único ou uma transformação é necessária, use [RequireAssertion](#). Para obter mais informações, consulte [usando um func para atender a uma política de](#).

## Vários avaliação de política

Se você aplicar várias políticas para um controlador ou ação, todas as políticas devem passar antes que o acesso é concedido. Por exemplo:

```
[Authorize(Policy = "EmployeeOnly")]
public class SalaryController : Controller
{
    public ActionResult Payslip()
    {
    }

    [Authorize(Policy = "HumanResources")]
    public ActionResult UpdateSalary()
    {
    }
}
```

No exemplo acima qualquer identidade que atende a `EmployeeOnly` política pode acessar o `Payslip` ação como essa política é aplicada no controlador. No entanto para chamar o `UpdateSalary` ação de identidade deve ser atendidos *ambos* o `EmployeeOnly` política e o `HumanResources` política.

Se você quiser políticas mais complicadas, como colocar uma data de nascimento declaração, calcular uma idade dele e verificando a idade for 21 ou anterior, você precisa gravar [manipuladores de política personalizada](#).

# Autorização baseada em política no ASP.NET Core

11/01/2019 • 11 minutes to read • [Edit Online](#)

Nos bastidores, [autoração baseada em funções](#) e [autoração baseada em declarações](#) usam um requisito, um manipulador de requisito e uma política previamente configurada. Esses blocos de construção dão suporte a expressão de avaliações de autorização no código. O resultado é uma estrutura de autorização mais sofisticados, reutilizáveis e testáveis.

Uma política de autorização consiste em um ou mais requisitos. Ele é registrado como parte da configuração do serviço de autorização no `Startup.ConfigureServices` método:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    services.AddAuthorization(options =>
    {
        options.AddPolicy("AtLeast21", policy =>
            policy.Requirements.Add(new MinimumAgeRequirement(21)));
    });
}
```

No exemplo anterior, uma política de "AtLeast21" é criada. Ele tem um requisito—da idade mínima, que é fornecida como um parâmetro para o requisito.

As políticas são aplicadas usando o `[Authorize]` atributo com o nome da política. Por exemplo:

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

[Authorize(Policy = "AtLeast21")]
public class AlcoholPurchaseController : Controller
{
    public IActionResult Login() => View();

    public IActionResult Logout() => View();
}
```

## Requisitos

Um requisito de autorização é uma coleção de parâmetros de dados que uma política pode usar para avaliar a entidade de segurança do usuário atual. Em nossa política de "AtLeast21", o requisito é um único parâmetro—a idade mínima. Implementa um requisito [IAuthorizationRequirement](#), que é uma interface de marcador vazio. Um requisito de idade mínima parametrizada poderia ser implementado da seguinte maneira:

```
using Microsoft.AspNetCore.Authorization;

public class MinimumAgeRequirement : IAuthorizationRequirement
{
    public int MinimumAge { get; private set; }

    public MinimumAgeRequirement(int minimumAge)
    {
        MinimumAge = minimumAge;
    }
}
```

#### NOTE

Um requisito não precisa ter dados ou propriedades.

## Manipuladores de autorização

Um manipulador de autorização é responsável para a avaliação das propriedades de um requisito. O manipulador de autorização avalia os requisitos em relação a um fornecido [AuthorizationHandlerContext](#) para determinar se o acesso é permitido.

Um requisito pode ter [vários manipuladores](#). Um manipulador pode herdar [AuthorizationHandler<TRequirement>](#), onde `TRequirement` é o requisito para ser tratada. Como alternativa, um manipulador pode implementar [IAuthorizationHandler](#) para lidar com mais de um tipo de requisito.

### Usar um manipulador para um requisito

Este é um exemplo de uma relação um para um em que um manipulador de idade mínima utiliza um requisito:

```

using Microsoft.AspNetCore.Authorization;
using PoliciesAuthApp1.Services.Requirements;
using System;
using System.Security.Claims;
using System.Threading.Tasks;

public class MinimumAgeHandler : AuthorizationHandler<MinimumAgeRequirement>
{
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,
                                                MinimumAgeRequirement requirement)
    {
        if (!context.User.HasClaim(c => c.Type == ClaimTypes.DateOfBirth &&
                                   c.Issuer == "http://contoso.com"))
        {
            //TODO: Use the following if targeting a version of
            ///.NET Framework older than 4.6:
            //    return Task.FromResult(0);
            return Task.CompletedTask;
        }

        var dateOfBirth = Convert.ToDateTime(
            context.User.FindFirst(c => c.Type == ClaimTypes.DateOfBirth &&
                                   c.Issuer == "http://contoso.com").Value);

        int calculatedAge = DateTime.Today.Year - dateOfBirth.Year;
        if (dateOfBirth > DateTime.Today.AddYears(-calculatedAge))
        {
            calculatedAge--;
        }

        if (calculatedAge >= requirement.MinimumAge)
        {
            context.Succeed(requirement);
        }

        //TODO: Use the following if targeting a version of
        ///.NET Framework older than 4.6:
        //    return Task.FromResult(0);
        return Task.CompletedTask;
    }
}

```

O código anterior determina se a entidade de usuário atual tem uma data de nascimento de declaração que foi emitido por um emissor conhecido e confiável. A autorização não pode ocorrer quando a declaração está ausente, caso em que uma tarefa concluída será retornada. Quando uma declaração estiver presente, a idade do usuário é calculada. Se o usuário atender a idade mínima definida pelo requisito, a autorização seja considerada bem-sucedida. Quando a autorização for bem-sucedida, `context.Succeed` é invocado com o requisito satisfeito como seu único parâmetro.

### **Usar um manipulador para várias necessidades**

Este é um exemplo de uma relação um-para-muitos em que um manipulador de permissão pode lidar com três tipos diferentes de requisitos:

```

using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using PoliciesAuthApp1.Services.Requirements;

public class PermissionHandler : IAuthorizationHandler
{
    public Task HandleAsync(AuthorizationHandlerContext context)
    {
        var pendingRequirements = context.PendingRequirements.ToList();

        foreach (var requirement in pendingRequirements)
        {
            if (requirement is ReadPermission)
            {
                if (IsOwner(context.User, context.Resource) ||
                    IsSponsor(context.User, context.Resource))
                {
                    context.Succeed(requirement);
                }
            }
            else if (requirement is EditPermission ||
                     requirement is DeletePermission)
            {
                if (IsOwner(context.User, context.Resource))
                {
                    context.Succeed(requirement);
                }
            }
        }

        //TODO: Use the following if targeting a version of
        // .NET Framework older than 4.6:
        //     return Task.FromResult(0);
        return Task.CompletedTask;
    }

    private bool IsOwner(ClaimsPrincipal user, object resource)
    {
        // Code omitted for brevity

        return true;
    }

    private bool IsSponsor(ClaimsPrincipal user, object resource)
    {
        // Code omitted for brevity

        return true;
    }
}

```

O código anterior percorre `PendingRequirements`—uma propriedade que contém requisitos não marcada como bem-sucedido. Para um `ReadPermission` requisito, o usuário deve ser um proprietário ou um responsável para acessar o recurso solicitado. No caso de um `EditPermission` OU `DeletePermission` requisito, ele ou ela deve ser um proprietário para acessar o recurso solicitado.

## Registro do manipulador

Manipuladores são registrados na coleção de serviços durante a configuração. Por exemplo:

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    services.AddAuthorization(options =>
    {
        options.AddPolicy("AtLeast21", policy =>
            policy.Requirements.Add(new MinimumAgeRequirement(21)));
    });

    services.AddSingleton<IAuthorizationHandler, MinimumAgeHandler>();
}

```

Cada manipulador é adicionado à coleção de serviços, invocando

```
services.AddSingleton<IAuthorizationHandler, YourHandlerClass>();
```

## O que deve retornar um manipulador?

Observe que o `Handle` método na [exemplo de manipulador](#) não retorna nenhum valor. Como é um status de êxito ou falha indicado?

- Um manipulador indica êxito chamando `context.Succeed(IAuthorizationRequirement requirement)`, passando o requisito de que foi validada com êxito.
- Um manipulador não precisa lidar com falhas em geral, como outros manipuladores para o mesmo requisito podem ter êxito.
- Para garantir a falha, mesmo se outros manipuladores de requisito tenha êxito, chame `context.Fail`.

Quando definido como `false`, o `InvokeHandlersAfterFailure` propriedade (disponível no ASP.NET Core 1.1 e posterior) causam curto-círcuito a execução de manipuladores quando `context.Fail` é chamado.

`InvokeHandlersAfterFailure` o padrão é `true`, caso em que todos os manipuladores são chamados. Isso permite que os requisitos produzir efeitos colaterais, como registro em log, que sempre ocorrem, mesmo se `context.Fail` foi chamado no manipulador de outro.

## Por que eu desejaria vários manipuladores para um requisito?

Em casos onde você deseja avaliação esteja em um **ou** base, implementar vários manipuladores para um requisito. Por exemplo, a Microsoft tem portas que apenas abrir com cartões de chave. Se você deixar o seu cartão de chave em casa, o recepcionista imprime um adesivo temporário e abre as portas para você. Nesse cenário, você teria um requisito *BuildingEntry*, mas vários manipuladores, cada um deles examinando um requisito.

*BuildingEntryRequirement.cs*

```

using Microsoft.AspNetCore.Authorization;

public class BuildingEntryRequirement : IAuthorizationRequirement
{
}

```

*BadgeEntryHandler.cs*

```

using Microsoft.AspNetCore.Authorization;
using PoliciesAuthApp1.Services.Requirements;
using System.Security.Claims;
using System.Threading.Tasks;

public class BadgeEntryHandler : AuthorizationHandler<BuildingEntryRequirement>
{
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,
                                                BuildingEntryRequirement requirement)
    {
        if (context.User.HasClaim(c => c.Type == ClaimTypes.BadgeId &&
                                  c.Issuer == "http://microsoftsecurity"))
        {
            context.Succeed(requirement);
        }

        //TODO: Use the following if targeting a version of
        // .NET Framework older than 4.6:
        //     return Task.FromResult(0);
        return Task.CompletedTask;
    }
}

```

### *TemporaryStickerHandler.cs*

```

using Microsoft.AspNetCore.Authorization;
using PoliciesAuthApp1.Services.Requirements;
using System.Security.Claims;
using System.Threading.Tasks;

public class TemporaryStickerHandler : AuthorizationHandler<BuildingEntryRequirement>
{
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,
                                                BuildingEntryRequirement requirement)
    {
        if (context.User.HasClaim(c => c.Type == ClaimTypes.TemporaryBadgeId &&
                                  c.Issuer == "https://microsoftsecurity"))
        {
            // We'd also check the expiration date on the sticker.
            context.Succeed(requirement);
        }

        //TODO: Use the following if targeting a version of
        // .NET Framework older than 4.6:
        //     return Task.FromResult(0);
        return Task.CompletedTask;
    }
}

```

Certifique-se de que ambos os manipuladores são [registrado](#). Se o manipulador é bem-sucedida quando uma política avalia o `BuildingEntryRequirement`, a avaliação da política é bem-sucedida.

## Usando um func para atender a uma política

Pode haver situações nas quais cumprir uma política é simples de expressar em código. É possível fornecer um `Func<AuthorizationHandlerContext, bool>` ao configurar a política com o `RequireAssertion` criador de políticas.

Por exemplo, o anterior `BadgeEntryHandler` poderia ser reescrito da seguinte maneira:

```
services.AddAuthorization(options =>
{
    options.AddPolicy("BadgeEntry", policy =>
        policy.RequireAssertion(context =>
            context.User.HasClaim(c =>
                (c.Type == ClaimTypes.BadgeId ||
                 c.Type == ClaimTypes.TemporaryBadgeId) &&
                 c.Issuer == "https://microsoftsecurity")));
});
```

## Acessando o contexto de solicitação do MVC nos manipuladores

O `HandleRequirementAsync` implementar em um manipulador de autorização de método tem dois parâmetros: uma `AuthorizationHandlerContext` e o `TRequirement` manipulada. Estruturas como MVC ou Jabbr serão livres para adicionar qualquer objeto para o `Resource` propriedade no `AuthorizationHandlerContext` para passar informações adicionais.

Por exemplo, o MVC passa uma instância do `AuthorizationFilterContext` no `Resource` propriedade. Esta propriedade fornece acesso aos `HttpContext`, `RouteData` e tudo o que outro fornecidos pelo MVC e páginas do Razor.

O uso do `Resource` é de propriedade específica do framework. Usando as informações no `Resource` propriedade limita suas políticas de autorização para determinadas estruturas. Você deve converter o `Resource` propriedade usando o `as` palavra-chave e, em seguida, confirme se a conversão foi bem-sucedida para garantir que seu código não falha com um `InvalidOperationException` quando executado em outras estruturas:

```
// Requires the following import:
//     using Microsoft.AspNetCore.Mvc.Filters;
if (context.Resource is AuthorizationFilterContext mvcContext)
{
    // Examine MVC-specific things like routing data.
}
```

# Provedores de política de autorização personalizados usando IAuthorizationPolicyProvider no ASP.NET Core

22/01/2019 • 9 minutes to read • [Edit Online](#)

Por [Mike Rousos](#)

Normalmente ao usar [baseado em políticas de autorização](#), as políticas são registradas por meio da chamada `AuthorizationOptions.AddPolicy` como parte da configuração do serviço de autorização. Em alguns cenários, pode não ser possível (ou desejável) para registrar todas as políticas de autorização dessa maneira. Nesses casos, você pode usar um personalizado `IAuthorizationPolicyProvider` para controlar como as políticas de autorização são fornecidas.

Exemplos de cenários onde um personalizado [IAuthorizationPolicyProvider](#) podem ser úteis incluem:

- Usando um serviço externo para fornecer a avaliação da política.
- Usando uma grande variedade de diretivas (para números de sala de diferentes ou com as idades, por exemplo), portanto, não faz sentido adicionar cada política de autorização individuais com um `AuthorizationOptions.AddPolicy` chamar.
- Criação de políticas em tempo de execução com base nas informações em uma fonte de dados externa (como um banco de dados) ou determinar os requisitos de autorização dinamicamente por meio de outro mecanismo.

[Exibir ou baixar o código de exemplo](#) do [repositório GitHub de AspNetCore](#). Baixe o arquivo ZIP do repositório `aspnet/AspNetCore`. Descompacte o arquivo. Navegue até a `src/Security/samples/CustomPolicyProvider` pasta do projeto.

## Personalizar a recuperação da política

Aplicativos ASP.NET Core usam uma implementação do `IAuthorizationPolicyProvider` interface para recuperar as políticas de autorização. Por padrão, `DefaultAuthorizationPolicyProvider` registrado e usado.

`DefaultAuthorizationPolicyProvider` Retorna as políticas do `AuthorizationOptions` fornecidas em um `IServiceCollection.AddAuthorization` chamar.

Você pode personalizar esse comportamento, registrando um diferentes `IAuthorizationPolicyProvider` implementação do aplicativo [injeção de dependência](#) contêiner.

O `IAuthorizationPolicyProvider` interface contém duas APIs:

- `GetPolicyAsync` retorna uma política de autorização para um determinado nome.
- `GetDefaultPolicyAsync` retorna a política de autorização padrão (a política usada para `[Authorize]` atributos sem uma política especificada).

Implementando essas duas APIs, você pode personalizar como as políticas de autorização são fornecidas.

## Parametrizadas autorizar o exemplo de atributo

Um cenário em que `IAuthorizationPolicyProvider` é útil é a habilitação do personalizado `[Authorize]` atributos cujos requisitos dependem de um parâmetro. Por exemplo, na [autorização baseada em política](#) documentação, uma com base em idade ("AtLeast21") política foi usada como um exemplo. Se as ações de controlador diferente em um aplicativo devem ser disponibilizadas para os usuários do *diferentes* há séculos, pode ser útil ter muitas

políticas diferentes com base em idade. Em vez de registrar todas as diferentes com base em idade políticas que o aplicativo será necessário em `AuthorizationOptions`, você pode gerar as políticas dinamicamente com um personalizado `IAuthorizationPolicyProvider`. Para tornar o uso de políticas mais fácil, você pode anotar as ações com o atributo de autorização personalizado, como `[MinimumAgeAuthorize(20)]`.

## Atributos de autorização personalizada

Políticas de autorização são identificadas por seus nomes. Personalizado `MinimumAgeAuthorizeAttribute` descrito anteriormente, precisa mapear argumentos em uma cadeia de caracteres que pode ser usada para recuperar a política de autorização correspondente. Você pode fazer isso, derivando de `AuthorizeAttribute` e fazer a `Age` quebra automática de propriedade a `AuthorizeAttribute.Policy` propriedade.

```
internal class MinimumAgeAuthorizeAttribute : AuthorizeAttribute
{
    const string POLICY_PREFIX = "MinimumAge";

    public MinimumAgeAuthorizeAttribute(int age) => Age = age;

    // Get or set the Age property by manipulating the underlying Policy property
    public int Age
    {
        get
        {
            if (int.TryParse(Policy.Substring(POLICY_PREFIX.Length), out var age))
            {
                return age;
            }
            return default(int);
        }
        set
        {
            Policy = $"{POLICY_PREFIX}{value.ToString()}";
        }
    }
}
```

Esse tipo de atributo tem um `Policy` cadeia de caracteres com base no prefixo embutido em código (`"MinimumAge"`) e um número inteiro passado por meio do construtor.

Você pode aplicá-lo às ações da mesma maneira que outras `Authorize` atributos, exceto que assume um inteiro como um parâmetro.

```
[MinimumAgeAuthorize(10)]
public IActionResult RequiresMinimumAge10()
```

## IAuthorizationPolicyProvider personalizado

Personalizado `MinimumAgeAuthorizeAttribute` facilita a políticas de autorização da solicitação para qualquer idade mínima desejada. O próximo problema a resolver é verificar se as políticas de autorização estão disponíveis para todas as idades diferentes. É aí que um `IAuthorizationPolicyProvider` é útil.

Ao usar `MinimumAgeAuthorizationAttribute`, os nomes de política de autorização seguirá o padrão `"MinimumAge" + Age`, então personalizado `IAuthorizationPolicyProvider` deve gerar diretivas de autorização por:

- A idade do nome da política de análise.
- Usando `AuthorizationPolicyBuilder` para criar um novo `AuthorizationPolicy`
- Adicionar requisitos para a política com base na idade com `AuthorizationPolicyBuilder.AddRequirements`. Em

outros cenários, você pode usar `RequireClaim`, `RequireRole`, ou `RequireUserName` em vez disso.

```
internal class MinimumAgePolicyProvider : IAuthorizationPolicyProvider
{
    const string POLICY_PREFIX = "MinimumAge";

    // Policies are looked up by string name, so expect 'parameters' (like age)
    // to be embedded in the policy names. This is abstracted away from developers
    // by the more strongly-typed attributes derived from AuthorizeAttribute
    // (like [MinimumAgeAuthorize()] in this sample)
    public Task<AuthorizationPolicy> GetPolicyAsync(string policyName)
    {
        if (policyName.StartsWith(POLICY_PREFIX, StringComparison.OrdinalIgnoreCase) &&
            int.TryParse(policyName.Substring(POLICY_PREFIX.Length), out var age))
        {
            var policy = new AuthorizationPolicyBuilder();
            policy.AddRequirements(new MinimumAgeRequirement(age));
            return Task.FromResult(policy.Build());
        }

        return Task.FromResult<AuthorizationPolicy>(null);
    }
}
```

## Vários provedores de política de autorização

Quando usar custom `IAuthorizationPolicyProvider` implementações, tenha em mente que o ASP.NET Core usa apenas uma instância de `IAuthorizationPolicyProvider`. Se um provedor personalizado não é capaz de fornecer políticas de autorização para todos os nomes de política, ele deve fazer fallback para um provedor de backup. Nomes de política podem incluir aqueles que vêm de uma política padrão para `[Authorize]` atributos sem um nome.

Por exemplo, considere que um aplicativo precisar de políticas personalizadas de idade e recuperação de política baseada em função mais tradicional. Esse aplicativo poderia usar um provedor de diretivas de autorização personalizada que:

- Tenta analisar nomes de política.
- Chamadas para um provedor de política diferente (como `DefaultAuthorizationPolicyProvider`) se o nome da política não contiver uma idade.

## Política padrão

Além de fornecer políticas de autorização nomeado, um personalizado `IAuthorizationPolicyProvider` precisa implementar `GetDefaultPolicyAsync` para fornecer uma política de autorização para `[Authorize]` atributos sem um nome de política especificado.

Em muitos casos, esse atributo de autorização requer apenas um usuário autenticado, portanto, você pode fazer a diretiva necessária com uma chamada para `RequireAuthenticatedUser`:

```
public Task<AuthorizationPolicy> GetDefaultPolicyAsync() =>
    Task.FromResult(new AuthorizationPolicyBuilder().RequireAuthenticatedUser().Build());
```

Assim como acontece com todos os aspectos de um personalizado `IAuthorizationPolicyProvider`, você pode personalizá-lo, conforme necessário. Em alguns casos:

- Políticas de autorização padrão não podem ser usadas.
- Recuperando a política padrão pode ser designado como um fallback `IAuthorizationPolicyProvider`.

## Use um `IAuthorizationPolicyProvider` personalizado

Para usar políticas personalizadas de um `IAuthorizationPolicyProvider`, você deve:

- Registrar apropriado `AuthorizationHandler` tipos de injeção de dependência (descrito na [autorização baseada em política](#)), assim como acontece com todos os cenários de autorização baseada em política.
- Registrar personalizado `IAuthorizationPolicyProvider` tipo na coleção de serviço de injeção de dependência do aplicativo (em `Startup.ConfigureServices`) para substituir o provedor de política padrão.

```
services.AddSingleton<IAuthorizationPolicyProvider, MinimumAgePolicyProvider>();
```

Um personalizado completo `IAuthorizationPolicyProvider` exemplo está disponível na [repositório do GitHub aspnet/AuthSamples](#).

# Injeção de dependência em manipuladores de requisito no ASP.NET Core

30/07/2018 • 2 minutes to read • [Edit Online](#)

Manipuladores de autorização devem ser registrados na coleção durante a configuração do serviço (usando [injeção de dependência](#)).

Suponha que você tenha um repositório de regras que você queira avaliar dentro de um manipulador de autorização e esse repositório foi registrado na coleção de serviço. A autorização será resolver e inseri-los em seu construtor.

Por exemplo, se você quiser usar o ASP.NET do log a infraestrutura que você deseja injetar `I LoggerFactory` em seu manipulador. Um manipulador desse tipo pode parecer com:

```
public class LoggingAuthorizationHandler : AuthorizationHandler<MyRequirement>
{
    ILogger _logger;

    public LoggingAuthorizationHandler(I LoggerFactory loggerFactory)
    {
        _logger = loggerFactory.CreateLogger(this.GetType().FullName);
    }

    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context, MyRequirement requirement)
    {
        _logger.LogInformation("Inside my handler");
        // Check if the requirement is fulfilled.
        return Task.CompletedTask;
    }
}
```

Você deve registrar o manipulador com `services.AddSingleton()`:

```
services.AddSingleton<IAuthorizationHandler, LoggingAuthorizationHandler>();
```

Uma instância do manipulador será criado quando o aplicativo for iniciado e será de DI injetar registrado `I LoggerFactory` em seu construtor.

## NOTE

Manipuladores que usam o Entity Framework não devem ser registrados como singletons.

# Autorização baseada em recursos no ASP.NET Core

16/11/2018 • 8 minutes to read • [Edit Online](#)

Estratégia de autorização depende do recurso que está sendo acessado. Considere um documento que tem uma propriedade de autor. Somente o autor tem permissão para atualizar o documento. Consequentemente, o documento deve ser recuperado do armazenamento de dados antes que a avaliação de autorização pode ocorrer.

Avaliação de atributo ocorre antes da vinculação de dados e antes da execução do manipulador de página ou ação que carrega o documento. Por esses motivos, a autorização declarativa com um `[Authorize]` atributo não é suficiente. Em vez disso, você pode invocar um método de autorização personalizada—um estilo conhecido como *autorização imperativa*.

[Exibir ou baixar um código de exemplo \(como baixar\).](#)

[Criar um aplicativo ASP.NET Core com os dados de usuário protegidos por autorização](#) contém um aplicativo de exemplo que usa a autorização baseada em recursos.

## Usar a autorização obrigatória

Autorização é implementada como um `IAuthorizationService` de serviço e é registrado na coleção de serviço dentro de `Startup` classe. O serviço é disponibilizado por meio [injeção de dependência](#) para manipuladores de página ou de ações.

```
public class DocumentController : Controller
{
    private readonly IAuthorizationService _authorizationService;
    private readonly IDocumentRepository _documentRepository;

    public DocumentController(IAuthorizationService authorizationService,
                             IDocumentRepository documentRepository)
    {
        _authorizationService = authorizationService;
        _documentRepository = documentRepository;
    }
}
```

`IAuthorizationService` tem dois `AuthorizeAsync` sobrecargas de método: uma aceitando o recurso e o nome da política e a outra aceitando o recurso e uma lista de requisitos para avaliar.

```
Task<AuthorizationResult> AuthorizeAsync(ClaimsPrincipal user,
                                         object resource,
                                         IEnumerable<IAuthorizationRequirement> requirements);
Task<AuthorizationResult> AuthorizeAsync(ClaimsPrincipal user,
                                         object resource,
                                         string policyName);
```

```
Task<bool> AuthorizeAsync(ClaimsPrincipal user,
                           object resource,
                           IEnumerable<IAuthorizationRequirement> requirements);
Task<bool> AuthorizeAsync(ClaimsPrincipal user,
                           object resource,
                           string policyName);
```

O exemplo a seguir, o recurso a ser protegido é carregado em um personalizado `Document` objeto. Um

`AuthorizeAsync` sobrecarga é invocada para determinar se o usuário atual tem permissão para editar o documento. Uma política de autorização personalizada "EditPolicy" é acrescentada a uma decisão. Ver [autORIZAÇÃO DE BASEADO EM POLÍTICAS PERSONALIZADAS](#) para obter mais informações sobre como criar políticas de autorização.

#### NOTE

O código a seguir exemplos pressupõem que a autenticação foi executada e o conjunto de `User` propriedade.

```
public async Task<IActionResult> OnGetAsync(Guid documentId)
{
    Document = _documentRepository.Find(documentId);

    if (Document == null)
    {
        return new NotFoundResult();
    }

    var authorizationResult = await _authorizationService
        .AuthorizeAsync(User, Document, "EditPolicy");

    if (authorizationResult.Succeeded)
    {
        return Page();
    }
    else if (User.Identity.IsAuthenticated)
    {
        return new ForbidResult();
    }
    else
    {
        return new ChallengeResult();
    }
}
```

```
[HttpGet]
public async Task<IActionResult> Edit(Guid documentId)
{
    Document document = _documentRepository.Find(documentId);

    if (document == null)
    {
        return new NotFoundResult();
    }

    if (await _authorizationService
        .AuthorizeAsync(User, document, "EditPolicy"))
    {
        return View(document);
    }
    else
    {
        return new ChallengeResult();
    }
}
```

## Escrever um Gerenciador de recursos

Escrevendo um manipulador para a autorização baseada em recursos não é muito diferente [escrevendo um manipulador de requisitos simples](#). Criar uma classe de requisito personalizado e implementar uma classe de manipulador de requisito. Para obter mais informações sobre como criar uma classe de requisito, consulte

requisitos de.

A classe de manipulador Especifica o requisito e o tipo de recurso. Por exemplo, uma utilização de manipulador uma `SameAuthorRequirement` e um `Document` recursos a seguir:

```
public class DocumentAuthorizationHandler :  
    AuthorizationHandler<SameAuthorRequirement, Document>  
{  
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,  
                                                SameAuthorRequirement requirement,  
                                                Document resource)  
    {  
        if (context.User.Identity?.Name == resource.Author)  
        {  
            context.Succeed(requirement);  
        }  
  
        return Task.CompletedTask;  
    }  
}  
  
public class SameAuthorRequirement : IAuthorizationRequirement { }
```

```
public class DocumentAuthorizationHandler :  
    AuthorizationHandler<SameAuthorRequirement, Document>  
{  
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,  
                                                SameAuthorRequirement requirement,  
                                                Document resource)  
    {  
        if (context.User.Identity?.Name == resource.Author)  
        {  
            context.Succeed(requirement);  
        }  
  
        //TODO: Use the following if targeting a version of  
        //.NET Framework older than 4.6:  
        //    return Task.FromResult(0);  
        return Task.CompletedTask;  
    }  
}  
  
public class SameAuthorRequirement : IAuthorizationRequirement { }
```

No exemplo anterior, imagine que `SameAuthorRequirement` é um caso especial de uma mais genérica `SpecificAuthorRequirement` classe. O `SpecificAuthorRequirement` classe (não mostrado) contém um `Name` propriedade que representa o nome do autor. O `Name` propriedade pode ser definida como o usuário atual.

Registrar o requisito e o manipulador no `Startup.ConfigureServices` :

```
services.AddMvc();  
  
services.AddAuthorization(options =>  
{  
    options.AddPolicy("EditPolicy", policy =>  
        policy.Requirements.Add(new SameAuthorRequirement()));  
});  
  
services.AddSingleton<IAuthorizationHandler, DocumentAuthorizationHandler>();  
services.AddSingleton<IAuthorizationHandler, DocumentAuthorizationCrudHandler>();  
services.AddScoped<IDocumentRepository, DocumentRepository>();
```

## Requisitos operacionais

Se você estiver fazendo decisões com base nos resultados de operações CRUD (Create, Read, Update, Delete), use o [OperationAuthorizationRequirement](#) classe auxiliar. Essa classe permite que você escreva um único manipulador em vez de uma classe individual para cada tipo de operação. Para usá-lo, forneça alguns nomes de operação:

```
public static class Operations
{
    public static OperationAuthorizationRequirement Create =
        new OperationAuthorizationRequirement { Name = nameof(Create) };
    public static OperationAuthorizationRequirement Read =
        new OperationAuthorizationRequirement { Name = nameof(Read) };
    public static OperationAuthorizationRequirement Update =
        new OperationAuthorizationRequirement { Name = nameof(Update) };
    public static OperationAuthorizationRequirement Delete =
        new OperationAuthorizationRequirement { Name = nameof(Delete) };
}
```

O manipulador é implementado da seguinte maneira, usando um `OperationAuthorizationRequirement` requisito e um `Document` recursos:

```
public class DocumentAuthorizationCrudHandler :
    AuthorizationHandler<OperationAuthorizationRequirement, Document>
{
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,
                                                    OperationAuthorizationRequirement requirement,
                                                    Document resource)
    {
        if (context.User.Identity?.Name == resource.Author &&
            requirement.Name == Operations.Read.Name)
        {
            context.Succeed(requirement);
        }

        return Task.CompletedTask;
    }
}
```

```
public class DocumentAuthorizationCrudHandler :
    AuthorizationHandler<OperationAuthorizationRequirement, Document>
{
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,
                                                    OperationAuthorizationRequirement requirement,
                                                    Document resource)
    {
        if (context.User.Identity?.Name == resource.Author &&
            requirement.Name == Operations.Read.Name)
        {
            context.Succeed(requirement);
        }

        //TODO: Use the following if targeting a version of
        // .NET Framework older than 4.6:
        //     return Task.FromResult(0);
        return Task.CompletedTask;
    }
}
```

O manipulador anterior valida a operação usando o recurso, a identidade do usuário e o requisito `Name` propriedade.

Para chamar um manipulador de recursos operacionais, especifique a operação ao invocar `AuthorizeAsync` em seu

manipulador de página ou ação. O exemplo a seguir determina se o usuário autenticado tem permissão para exibir o documento.

#### NOTE

O código a seguir exemplifica pressupõem que a autenticação foi executada e o conjunto de `User` propriedade.

```
public async Task<IActionResult> OnGetAsync(Guid documentId)
{
    Document = _documentRepository.Find(documentId);

    if (Document == null)
    {
        return new NotFoundResult();
    }

    var authorizationResult = await _authorizationService
        .AuthorizeAsync(User, Document, Operations.Read);

    if (authorizationResult.Succeeded)
    {
        return Page();
    }
    else if (User.Identity.IsAuthenticated)
    {
        return new ForbidResult();
    }
    else
    {
        return new ChallengeResult();
    }
}
```

Se a autorização for bem-sucedida, a página para exibir o documento é retornada. Se falhar de autorização, mas o usuário é autenticado, o retorno de `ForbidResult` informa qualquer middleware de autenticação que houve falha na autorização. Um `ChallengeResult` é retornado quando a autenticação deve ser executada. Para clientes de navegador interativo, ele pode ser apropriado redirecionar o usuário para uma página de logon.

```
[HttpGet]
public async Task<IActionResult> View(Guid documentId)
{
    Document document = _documentRepository.Find(documentId);

    if (document == null)
    {
        return new NotFoundResult();
    }

    if (await _authorizationService
        .AuthorizeAsync(User, document, Operations.Read))
    {
        return View(document);
    }
    else
    {
        return new ChallengeResult();
    }
}
```

Se a autorização for bem-sucedida, o modo de exibição para o documento é retornado. Se a autorização falhar, retornando `ChallengeResult` informa qualquer middleware de autenticação que houve falha na autorização e o

middleware pode levar a resposta apropriada. Uma resposta apropriada pode estar retornando um código de status 401 ou 403. Para clientes de navegador interativo, isso poderia significar a redirecionar o usuário para uma página de logon.

# Autorização baseada em modo de exibição no ASP.NET Core MVC

30/07/2018 • 2 minutes to read • [Edit Online](#)

Um desenvolvedor geralmente deseja mostrar, ocultar ou modificar uma interface do usuário com base na identidade do usuário atual. Você pode acessar o serviço de autorização em modos de exibição do MVC por meio [injeção de dependência](#). Para injetar o serviço de autorização em um modo de exibição do Razor, use o `@inject` diretiva:

```
@using Microsoft.AspNetCore.Authorization  
@inject IAuthorizationService AuthorizationService
```

Se você quiser que o serviço de autorização em cada exibição, coloque o `@inject` diretiva na `viewimports.cshtml` arquivo do `exibições` directory. Para obter mais informações, consulte [Injeção de dependência em exibições](#).

Usar o serviço de autorização injetado para invocar `AuthorizeAsync` exatamente da mesma forma que você poderia verificar durante [autorização baseada em recursos](#):

- [ASP.NET Core 2.x](#)
- [ASP.NET Core 1.x](#)

```
@if ((await AuthorizationService.AuthorizeAsync(User, "PolicyName")).Succeeded)  
{  
    <p>This paragraph is displayed because you fulfilled PolicyName.</p>  
}
```

Em alguns casos, o recurso será o seu modelo de exibição. Invocar `AuthorizeAsync` exatamente da mesma forma que você poderia verificar durante [autorização baseada em recursos](#):

- [ASP.NET Core 2.x](#)
- [ASP.NET Core 1.x](#)

```
@if ((await AuthorizationService.AuthorizeAsync(User, Model, Operations.Edit)).Succeeded)  
{  
    <p><a class="btn btn-default" role="button"  
        href="@Url.Action("Edit", "Document", new { id = Model.Id })">Edit</a></p>  
}
```

No código anterior, o modelo é passado como um recurso de que avaliação da política deve levar em consideração.

## WARNING

Não confie na alternância de visibilidade de elementos de interface do usuário do seu aplicativo como a verificação de autorização única. Ocultar um elemento de interface do usuário pode completamente impedir o acesso a sua ação de controlador associado. Por exemplo, considere o botão no trecho de código anterior. Um usuário pode invocar o `Edit` é de URL do método de ação, se ele ou ela sabe que o recurso relativo `/Document/Edit/1`. Por esse motivo, o `Edit` método de ação deve realizar sua própria verificação de autorização.

# Autorizar com um esquema específico no ASP.NET Core

26/10/2018 • 6 minutes to read • [Edit Online](#)

Em alguns cenários, como aplicativos de página única (SPAs), é comum usar vários métodos de autenticação. Por exemplo, o aplicativo pode usar a autenticação baseada em cookies para fazer logon e autenticação do portador do JWT para solicitações de JavaScript. Em alguns casos, o aplicativo pode ter várias instâncias de um manipulador de autenticação. Por exemplo, dois manipuladores de cookie em que um contém uma identidade básica e o outro é criado quando a autenticação multifator (MFA) foi acionada. MFA pode ser acionado porque o usuário solicitou uma operação que exige segurança extra.

- [ASP.NET Core 2.x](#)
- [ASP.NET Core 1.x](#)

Um esquema de autenticação é chamado quando o serviço de autenticação é configurado durante a autenticação. Por exemplo:

```
public void ConfigureServices(IServiceCollection services)
{
    // Code omitted for brevity

    services.AddAuthentication()
        .AddCookie(options => {
            options.LoginPath = "/Account/Unauthorized/";
            options.AccessDeniedPath = "/Account/Forbidden/";
        })
        .AddJwtBearer(options => {
            options.Audience = "http://localhost:5001/";
            options.Authority = "http://localhost:5000/";
        });
}
```

No código anterior, foram adicionados dois manipuladores de autenticação: um para cookies e outro para o portador.

## NOTE

Especificar o esquema padrão resulta no `HttpContext.User` propriedade sendo definida como essa identidade. Se esse comportamento não for desejado, desabilitá-lo, invocando o formulário sem parâmetros de `AddAuthentication`.

## Selecionar o esquema com o atributo Authorize

No ponto de autorização, o aplicativo indica o manipulador a ser usado. Selecione o manipulador com a qual o aplicativo autorizará passando uma lista delimitada por vírgulas de esquemas de autenticação para `[Authorize]`. O `[Authorize]` atributo especifica o esquema de autenticação ou esquemas de usar, independentemente de um padrão é configurado. Por exemplo:

- [ASP.NET Core 2.x](#)
- [ASP.NET Core 1.x](#)

```
[Authorize(AuthenticationSchemes = AuthSchemes)]
public class MixedController : Controller
    // Requires the following imports:
    // using Microsoft.AspNetCore.Authentication.Cookies;
    // using Microsoft.AspNetCore.Authentication.JwtBearer;
    private const string AuthSchemes =
        CookieAuthenticationDefaults.AuthenticationScheme + "," +
        JwtBearerDefaults.AuthenticationScheme;
```

No exemplo anterior, o cookie e o portador manipuladores execute em tenham a oportunidade de criar e acrescentar uma identidade para o usuário atual. Ao especificar um único esquema, o manipulador correspondente é executado.

- [ASP.NET Core 2.x](#)
- [ASP.NET Core 1.x](#)

```
[Authorize(AuthenticationSchemes =
    JwtBearerDefaults.AuthenticationScheme)]
public class MixedController : Controller
```

No código anterior, somente o manipulador com o esquema de "Bearer" é executado. Todas as identidades baseadas em cookies serão ignoradas.

## Selecionar o esquema com as políticas

Se você preferir especificar os esquemas de desejada [diretiva](#), você pode definir o `AuthenticationSchemes` coleção ao adicionar sua política:

```
services.AddAuthorization(options =>
{
    options.AddPolicy("Over18", policy =>
    {
        policy.AuthenticationSchemes.Add(JwtBearerDefaults.AuthenticationScheme);
        policy.RequireAuthenticatedUser();
        policy.Requirements.Add(new MinimumAgeRequirement());
    });
});
```

No exemplo anterior, a política de "Over18" só é executada em relação a identidade criada pelo manipulador de "Portador". Usar a política, definindo o `[Authorize]` do atributo `Policy` propriedade:

```
[Authorize(Policy = "Over18")]
public class RegistrationController : Controller
```

## Usar vários esquemas de autenticação

Alguns aplicativos talvez seja necessário dar suporte a vários tipos de autenticação. Por exemplo, seu aplicativo pode autenticar os usuários do Active Directory do Azure e de um banco de dados de usuários. Outro exemplo é um aplicativo que autentica os usuários do Active Directory Federation Services e o Azure Active Directory B2C. Nesse caso, o aplicativo deve aceitar um token de portador JWT de vários emissores.

Adicione todos os esquemas de autenticação que você gostaria de aceitar. Por exemplo, o código a seguir no `Startup.ConfigureServices` adiciona dois esquemas de autenticação de portador JWT com emissores diferentes:

```
public void ConfigureServices(IServiceCollection services)
{
    // Code omitted for brevity

    services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
        .AddJwtBearer(options =>
    {
        options.Audience = "https://localhost:5000/";
        options.Authority = "https://localhost:5000/identity/";
    })
        .AddJwtBearer("AzureAD", options =>
    {
        options.Audience = "https://localhost:5000/";
        options.Authority = "https://login.microsoftonline.com/eb971100-6f99-4bdc-8611-1bc8edd7f436/";
    });
}
```

#### NOTE

Apenas uma autenticação de portador JWT está registrada com o esquema de autenticação padrão `JwtBearerDefaults.AuthenticationScheme`. Autenticação adicional deve ser registrado com um esquema de autenticação exclusiva.

A próxima etapa é atualizar a política de autorização padrão para aceitar os dois esquemas de autenticação. Por exemplo:

```
public void ConfigureServices(IServiceCollection services)
{
    // Code omitted for brevity

    services.AddAuthorization(options =>
    {
        var defaultAuthorizationPolicyBuilder = new AuthorizationPolicyBuilder(
            JwtBearerDefaults.AuthenticationScheme,
            "AzureAD");
        defaultAuthorizationPolicyBuilder =
            defaultAuthorizationPolicyBuilder.RequireAuthenticatedUser();
        options.DefaultPolicy = defaultAuthorizationPolicyBuilder.Build();
    });
}
```

Como a política de autorização padrão é substituída, é possível usar um simples `[Authorize]` atributo em controladores. O controlador, em seguida, aceita solicitações com JWT emitido pelo emissor do primeiro ou segundo.

# Proteção de dados do ASP.NET Core

26/10/2018 • 11 minutes to read • [Edit Online](#)

Aplicativos da Web geralmente precisam armazenar dados confidenciais de segurança. Windows fornece DPAPI para aplicativos da área de trabalho, mas isso é inadequado para aplicativos da web. A pilha de proteção de dados do ASP.NET Core fornecem uma API de criptografia simples e fácil de usar um desenvolvedor pode usar para proteger dados, incluindo rotação e gerenciamento de chaves.

A pilha de proteção de dados do ASP.NET Core foi projetada para servir como a substituição de longo prazo para o `<machineKey>` elemento no ASP.NET 1.x - 4.x. Ele foi projetado para resolver muitos dos problemas da pilha de criptografia antigo, fornecendo uma solução para a maioria dos casos de uso, os aplicativos modernos têm probabilidade de encontrar.

## Declaração do problema

A declaração do problema geral pode ser declarada em uma única sentença sucintamente: eu preciso manter informações confiáveis para recuperação posterior, mas eu não confio o mecanismo de persistência. Em termos de web, isso pode ser escrito como "Eu preciso de ida e volta estado confiável por meio de um cliente não confiável."

O exemplo canônico disso é um cookie de autenticação ou portador token. O servidor gera um "Estou Groot e ter permissões de xyz" de token e entrega-o para o cliente. Em uma data futura, o cliente apresentará esse token de volta para o servidor, mas o servidor precisa de algum tipo de garantia de que o cliente não foi forjada o token. Portanto, o primeiro requisito: autenticidade (também conhecido como integridade e à prova de adulteração).

Uma vez que o estado persistente é confiável pelo servidor, estimamos que esse estado pode conter informações específicas para o ambiente operacional. Isso pode ser na forma de um caminho de arquivo, uma permissão, um identificador ou outra referência indireta, ou alguma outra parte dos dados específicos do servidor. Essas informações, em geral, não deveriam ser divulgadas para um cliente não confiável. Portanto, o segundo requisito: confidencialidade.

Por fim, como aplicativos modernos são divididos em componentes, o que vimos é componentes individuais vai querer tirar proveito desse sistema sem levar em consideração outros componentes no sistema. Por exemplo, se um componente de token de portador é usando essa pilha, ele deve operar sem interferência de um mecanismo de anti-CSRF que também pode usar a mesma pilha. Portanto, o requisito final: isolamento.

Podemos fornecer ainda mais as restrições para limitar o escopo de nossos requisitos. Vamos supor que todos os serviços que operam com o sistema de criptografia são igualmente confiáveis e que os dados não precisam ser gerado ou consumido fora os serviços sob nosso controle direto. Além disso, exigimos que as operações são tão rápidas quanto possível, pois cada solicitação ao serviço web pode percorrer o sistema criptográfico uma ou mais vezes. Isso torna a criptografia simétrica ideal para o nosso cenário, e podemos pode desconto criptografia assimétrica até que tal uma vez que ele é necessário.

## Filosofia de design

Começamos por meio da identificação de problemas com a pilha existente. Depois que tivermos, nós pesquisamos o panorama das soluções existentes e concluiu que não há solução existente teve muito os recursos que são procurados. Em seguida, projetamos uma solução baseada em vários princípios de orientação.

- O sistema deve oferecer a simplicidade da configuração. O ideal é que o sistema seria sem nenhuma configuração e os desenvolvedores poderiam parta para a ação. Em situações em que os desenvolvedores precisam configurar um aspecto específico (como o repositório de chaves), consideração deve ser fornecida a fazer essas configurações específicas simples.
- Oferecem uma API voltado ao consumidor simples. As APIs devem ser fáceis de usar corretamente e difícil de usar incorretamente.
- Os desenvolvedores não devem aprender os princípios de gerenciamento de chaves. O sistema deve lidar com a seleção de algoritmo e a vida útil da chave em nome do desenvolvedor. O ideal é que o desenvolvedor nunca mesmo deve ter acesso ao material da chave bruta.
- As chaves devem ser protegidas em repouso quando possível. O sistema deve descobrir um mecanismo de proteção padrão apropriado e aplicá-la automaticamente.

Com esses princípios em mente, desenvolvemos um simples [fácil de usar pilha da proteção de dados](#).

As APIs de proteção de dados do ASP.NET Core não destinam principalmente para persistência indefinida de cargas confidenciais. Outras tecnologias, como [DPAPI do Windows CNG](#) e [do Azure Rights Management](#) são mais adequados para o cenário de armazenamento indefinido, e eles têm recursos de gerenciamento de chave forte de forma correspondente. Dito isso, não há nada proibindo um desenvolvedor que usa as APIs de proteção de dados do ASP.NET Core para proteção de longo prazo de dados confidenciais.

## Público-alvo

O sistema de proteção de dados é dividido em cinco pacotes principais. Vários aspectos dessas APIs três principal públicos-alvo; de destino

1. O [visão geral das APIs de consumidor](#) os desenvolvedores de aplicativo e estrutura de destino.

"Quero saber mais sobre como funciona a pilha de ou sobre como ela é configurada. Simplesmente quero executar alguma operação no tão simples uma maneira possível com a alta probabilidade de usar as APIs com êxito".

2. O [APIs de configuração](#) os desenvolvedores de aplicativos e administradores de sistema de destino.

"Eu preciso informar ao sistema de proteção de dados de que meu ambiente requer configurações ou os caminhos não padrão."

3. Os desenvolvedores de destino APIs de extensibilidade responsável pela implementação de política personalizada. O uso dessas APIs seria limitado a situações raras e experientes, os desenvolvedores de reconhecimento de segurança.

"Eu preciso substituir um componente inteiro dentro do sistema porque tenho requisitos comportamentais realmente exclusivos. Desejo saber usados raramente partes da superfície de API a fim de criar um plug-in que atenda aos meus requisitos."

## Layout do pacote

A pilha da proteção de dados consiste em cinco pacotes.

- [Microsoft.AspNetCore.DataProtection.Abstractions](#) contém o [IDataProtectionProvider](#) e [IDataProtector](#) interfaces para criar serviços de proteção de dados. Ele também contém métodos de extensão úteis para trabalhar com esses tipos (por exemplo, [IDataProtector.Protect](#)). Se o sistema de proteção de dados é instanciado em outro lugar e você está consumindo a API, referência [Microsoft.AspNetCore.DataProtection.Abstractions](#).
- [Microsoft.AspNetCore.DataProtection](#) contém a implementação principal do sistema de proteção de

dados, incluindo operações criptográficas de núcleo, gerenciamento de chaves, configuração e extensibilidade. Para instanciar o sistema de proteção de dados (por exemplo, adicioná-lo para um [IServiceCollection](#)) ou fazer referência a modificar ou estender seu comportamento,

`Microsoft.AspNetCore.DataProtection`.

- [Microsoft.AspNetCore.DataProtection.Extensions](#) contém APIs adicionais que os desenvolvedores podem ser úteis, mas que não cabem no pacote principal. Por exemplo, este pacote contém métodos de fábrica para instanciar o sistema de proteção de dados para armazenar chaves em um local no sistema de arquivos sem injeção de dependência (consulte [DataProtectionProvider](#)). Ele também contém métodos de extensão para limitar o tempo de vida de cargas protegidas (consulte [ITimeLimitedDataProtector](#)).
- [Microsoft.AspNetCore.DataProtection.SystemWeb](#) pode ser instalado em um aplicativo existente do ASP.NET 4.x para redirecionar seu `<machineKey>` operações para usar a nova pilha de proteção de dados do ASP.NET Core. Para obter mais informações, consulte [Substitua o machineKey ASP.NET no núcleo do ASP.NET](#).
- [Microsoft.AspNetCore.Cryptography.KeyDerivation](#) fornece uma implementação da rotina de hash de senha de PBKDF2 e podem ser usadas por sistemas que precisam lidar com as senhas de usuário com segurança. Para obter mais informações, consulte [Hash de senhas no ASP.NET Core](#).

## Recursos adicionais

[Hospedar o ASP.NET Core em um web farm](#)

# Comece com as APIs de proteção de dados no ASP.NET Core

22/08/2018 • 3 minutes to read • [Edit Online](#)

Em seus dados mais simples, protegendo consiste as seguintes etapas:

1. Crie dados de um protetor de um provedor de proteção de dados.
2. Chamar o `Protect` método com os dados que você deseja proteger.
3. Chamar o `Unprotect` método com os dados que você deseja transformar de volta em texto sem formatação.

A maioria das estruturas e modelos de aplicativo, como o ASP.NET Core ou SignalR, já que configurar o sistema de proteção de dados e adicioná-lo a um contêiner de serviço que você acessa por meio da injeção de dependência. O exemplo a seguir demonstra Configurando um contêiner de serviço para injeção de dependência e registrando a pilha da proteção de dados, recebendo o provedor de proteção de dados por meio da DI, criando um protetor e desproteção em seguida, proteção de dados.

```

using System;
using Microsoft.AspNetCore.DataProtection;
using Microsoft.Extensions.DependencyInjection;

public class Program
{
    public static void Main(string[] args)
    {
        // add data protection services
        var serviceCollection = new ServiceCollection();
        serviceCollection.AddDataProtection();
        var services = serviceCollection.BuildServiceProvider();

        // create an instance of MyClass using the service provider
        var instance = ActivatorUtilities.CreateInstance<MyClass>(services);
        instance.RunSample();
    }

    public class MyClass
    {
        IDataProtector _protector;

        // the 'provider' parameter is provided by DI
        public MyClass(IDataProtectionProvider provider)
        {
            _protector = provider.CreateProtector("Contoso.MyClass.v1");
        }

        public void RunSample()
        {
            Console.Write("Enter input: ");
            string input = Console.ReadLine();

            // protect the payload
            string protectedPayload = _protector.Protect(input);
            Console.WriteLine($"Protect returned: {protectedPayload}");

            // unprotect the payload
            string unprotectedPayload = _protector.Unprotect(protectedPayload);
            Console.WriteLine($"Unprotect returned: {unprotectedPayload}");
        }
    }
}

/*
 * SAMPLE OUTPUT
 *
 * Enter input: Hello world!
 * Protect returned: CfDJ8ICcgQwZZh1ALTZT...OdfH66i1PnGmpCR5e441xQ
 * Unprotect returned: Hello world!
*/

```

Quando você cria um protetor deve fornecer um ou mais [cadeias de caracteres de finalidade](#). Uma cadeia de caracteres de finalidade fornece isolamento entre os consumidores. Por exemplo, um protetor criado com uma cadeia de caracteres de finalidade de "verde" seria capaz de Desproteger dados fornecidos por um protetor, com a finalidade de "roxo".

#### TIP

Instâncias do `IDataProtectionProvider` e `IDataProtector` sejam thread-safe para chamadores vários. Ele deve que depois que um componente obtém uma referência a um `IDataProtector` por meio de uma chamada para `CreateProtector`, ele usará essa referência para várias chamadas para `Protect` e `Unprotect`.

Uma chamada para `Unprotect` lançará `CryptographicException` se o conteúdo protegido não pode ser verificado ou decifrado. Alguns componentes talvez queira ignorar erros durante a Desproteger operações; um componente que lê os cookies de autenticação pode manipular esse erro e tratar a solicitação como se ele não tinha nenhum cookie em todos os em vez de falhar a solicitação imediatamente. Componentes que desejam esse comportamento devem especificamente capturar `CryptographicException` em vez de assimilação de todas as exceções.

# Visão geral APIs do consumidor do ASP.NET Core

22/06/2018 • 7 minutes to read • [Edit Online](#)

O `IDataProtectionProvider` e `IDataProtector` interfaces são as interfaces básicas por meio do qual os consumidores usam o sistema de proteção de dados. Eles estão localizados no `Microsoft.AspNetCore.DataProtection.Abstractions` pacote.

## IDataProtectionProvider

A interface de provedor representa a raiz do sistema de proteção de dados. Ele não pode ser usado diretamente para proteger ou desproteger dados. Em vez disso, o consumidor deve obter uma referência a um `IDataProtector` chamando `IDataProtectionProvider.CreateProtector(purpose)`, onde o objetivo é uma cadeia de caracteres que descreve o caso de uso pretendido do consumidor. Consulte [cadeias de caracteres de finalidade](#) para obter mais informações sobre a intenção deste parâmetro e como escolher um valor apropriado.

## IDataProtector

A interface de protetor é retornada por uma chamada para `CreateProtector` e essa interface que os consumidores podem usar para executar proteger e Desproteger as operações.

Para proteger uma parte dos dados, transmitir os dados para o `Protect` método. A interface básica define um método que byte converte `[] -> byte []`, mas há também uma sobrecarga (fornecida como um método de extensão), que converte a cadeia de caracteres `->` a cadeia de caracteres. A segurança oferecida pelos dois métodos é idêntica; o desenvolvedor deve escolher qualquer sobrecarga é mais conveniente para seu caso de uso. Independentemente da sobrecarga escolhida, o valor retornado pelo `Protect` método agora está protegido (enciphered e à prova de adulteração) e o aplicativo pode enviá-lo para um cliente não confiável.

Para desproteger uma parte anteriormente protegido dos dados, passar os dados protegidos para o `Unprotect` método. (Há `byte []`-sobrecargas com base e baseada em cadeia de caracteres para conveniência do desenvolvedor.) Se a carga protegida foi gerada por uma chamada anterior para `Protect` esse mesmo `IDataProtector`, o `Unprotect` método retornará a carga desprotegida original. Se a carga protegida tenha sido violada ou foi produzida por outro `IDataProtector`, o `Unprotect` método lançará `CryptographicException`.

O conceito de mesmo versus diferentes `IDataProtector` vínculos de volta para o conceito de finalidade. Se dois `IDataProtector` instâncias foram geradas a partir da mesma raiz `IDataProtectionProvider` mas por meio de cadeias de caracteres de finalidade diferente na chamada para `IDataProtectionProvider.CreateProtector`, em seguida, elas são consideradas [protetores diferentes](#), e um não será possível desproteger cargas geradas por outros.

## Essas interfaces de consumo

Para um componente com reconhecimento de injeção de dependência, o uso pretendido é que o componente levar uma `IDataProtectionProvider` parâmetro em seu construtor e que o sistema DI fornece automaticamente esse serviço quando o componente é instanciado.

#### **NOTE**

Alguns aplicativos (como aplicativos de console ou aplicativos do ASP.NET 4. x) podem não ser DI com reconhecimento de forma não é possível usar o mecanismo descrito aqui. Para esses cenários consulte o [cenários com reconhecimento de DI não](#) documento para obter mais informações sobre a obtenção de uma instância de um `IDataProtection` provedor sem passar por DI.

O exemplo a seguir demonstra três conceitos:

1. [Adicione o sistema de proteção de dados](#) ao contêiner de serviço,
2. Usando DI para receber uma instância de um `IDataProtectionProvider`, e
3. Criando um `IDataProtector` de um `IDataProtectionProvider` e usá-lo para proteger e Desproteger dados.

```

using System;
using Microsoft.AspNetCore.DataProtection;
using Microsoft.Extensions.DependencyInjection;

public class Program
{
    public static void Main(string[] args)
    {
        // add data protection services
        var serviceCollection = new ServiceCollection();
        serviceCollection.AddDataProtection();
        var services = serviceCollection.BuildServiceProvider();

        // create an instance of MyClass using the service provider
        var instance = ActivatorUtilities.CreateInstance<MyClass>(services);
        instance.RunSample();
    }

    public class MyClass
    {
        IDataProtector _protector;

        // the 'provider' parameter is provided by DI
        public MyClass(IDataProtectionProvider provider)
        {
            _protector = provider.CreateProtector("Contoso.MyClass.v1");
        }

        public void RunSample()
        {
            Console.Write("Enter input: ");
            string input = Console.ReadLine();

            // protect the payload
            string protectedPayload = _protector.Protect(input);
            Console.WriteLine($"Protect returned: {protectedPayload}");

            // unprotect the payload
            string unprotectedPayload = _protector.Unprotect(protectedPayload);
            Console.WriteLine($"Unprotect returned: {unprotectedPayload}");
        }
    }
}

/*
 * SAMPLE OUTPUT
 *
 * Enter input: Hello world!
 * Protect returned: CfDJ8ICcgQwZZhlALTZT...OdfH66i1PnGmpCR5e441xQ
 * Unprotect returned: Hello world!
*/

```

O pacote Microsoft.AspNetCore.DataProtection.Abstractions contém um método de extensão

`IServiceProvider.GetDataProtector` como uma conveniência de desenvolvedor. Ele encapsula como uma única operação ambos recuperando uma `IDataProtectionProvider` do provedor de serviços e chamar `IDataProtectionProvider.CreateProtector`. O exemplo a seguir demonstra o uso.

```

using System;
using Microsoft.AspNetCore.DataProtection;
using Microsoft.Extensions.DependencyInjection;

public class Program
{
    public static void Main(string[] args)
    {
        // add data protection services
        var serviceCollection = new ServiceCollection();
        serviceCollection.AddDataProtection();
        var services = serviceCollection.BuildServiceProvider();

        // get an IDataProtector from the IServiceProvider
        var protector = services.GetDataProtector("Contoso.Example.v2");
        Console.WriteLine("Enter input: ");
        string input = Console.ReadLine();

        // protect the payload
        string protectedPayload = protector.Protect(input);
        Console.WriteLine($"Protect returned: {protectedPayload}");

        // unprotect the payload
        string unprotectedPayload = protector.Unprotect(protectedPayload);
        Console.WriteLine($"Unprotect returned: {unprotectedPayload}");
    }
}

```

#### TIP

Instâncias do `IDataProtectionProvider` e `IDataProtector` são thread-safe para chamadores vários. Ele foi desenvolvido que depois que um componente obtém uma referência a um `IDataProtector` por meio de uma chamada para `CreateProtector`, ele usará essa referência para várias chamadas para `Protect` e `Unprotect`. Uma chamada para `Unprotect` lançará `CryptographicException` se a carga protegida não pode ser verificada ou decifrada. Alguns componentes poderão ignorar erros durante Desproteger operações; um componente que lê os cookies de autenticação pode manipular esse erro e tratar a solicitação, como não se tivesse nenhum cookie de em vez de falhar a solicitação. Componentes que deseja que esse comportamento especificamente devem capturar `CryptographicException` em vez de assimilação todas as exceções.

# Cadeias de caracteres de finalidade no núcleo do ASP.NET

22/06/2018 • 6 minutes to read • [Edit Online](#)

Componentes que consomem `IDataProtectionProvider` deve passar em uma única *fins* parâmetro para o `CreateProtector` método. A finalidade *parâmetro* é inerente à segurança do sistema de proteção de dados, como ele fornece isolamento entre consumidores de criptografia, mesmo se as chaves de criptografia de raiz são as mesmas.

Quando um consumidor Especifica a finalidade, a cadeia de caracteres de finalidade é usada junto com as chaves de criptografia de raiz para derivar criptográficas subchaves exclusivas para que o consumidor. Isso isola o consumidor de todos os outros consumidores criptográficos do aplicativo: nenhum outro componente pode ler suas cargas de e não pode ler cargas do qualquer outro componente. Esse isolamento também processa inviável categorias inteiras de ataque em relação ao componente.



No diagrama acima, `IDataProtector` instâncias A e B **não é possível** ler umas das outras cargas, somente seus próprios.

A cadeia de caracteres de fim não precisa ser segredo. Ele simplesmente deve ser exclusivo no sentido de que nenhum outro componente com bom comportamento já fornecem a mesma cadeia de caracteres de finalidade.

## TIP

Usando o nome de namespace e o tipo do componente consumindo as APIs de proteção de dados é uma boa regra geral, como prática que essas informações nunca entrarão em conflito.

Um componente de autoria do Contoso que é responsável por minting tokens de portador pode usar `Contoso.Security.BearerToken` como cadeia de caracteres sua finalidade. Ou - ainda melhor - ele pode usar `Contoso.Security.BearerToken.v1` como cadeia de caracteres sua finalidade. Anexar o número de versão permite que uma versão futura usar `Contoso.Security.BearerToken.v2` como sua finalidade e as versões diferentes seria completamente isoladas uma da outra quanto cargas ir.

Desde o parâmetro fins `CreateProtector` é uma matriz de cadeia de caracteres, acima poderiam ter sido em vez disso, especificadas como `[ "Contoso.Security.BearerToken", "v1" ]`. Isso permite o estabelecimento de uma hierarquia de propósitos e abre a possibilidade de cenários de multilocação com o sistema de proteção de dados.

## WARNING

Componentes não devem permitir que a entrada do usuário não confiável ser a única origem de entrada para a cadeia de fins.

Por exemplo, considere um componente Contoso.Messaging.SecureMessage que é responsável por armazenar mensagens seguras. Se o componente de mensagens seguro chamar `CreateProtector([ username ])`, em seguida, um usuário mal-intencionado pode criar uma conta com o nome de usuário "Contoso.Security.BearerToken" em uma tentativa de obter o componente para chamar `CreateProtector([ "Contoso.Security.BearerToken" ])`, inadvertidamente provocando mensagens seguras sistema cargas Menta que pode ser considerada como tokens de autenticação.

Uma cadeia de fins melhor para o componente de mensagens seria

`CreateProtector([ "Contoso.Messaging.SecureMessage", "User: username" ])`, que fornece isolamento apropriado.

O isolamento fornecido pelos e comportamentos de `IDataProtectionProvider`, `IDataProtector`, e fins é o seguinte:

- Para um determinado `IDataProtectionProvider` objeto, o `CreateProtector` método criará uma `IDataProtector` objeto vinculado exclusivamente para ambos os `IDataProtectionProvider` objeto que criou e o parâmetro de fins que foi passado para o método.
- O parâmetro de fim não deve ser nulo. (Se fins é especificado como uma matriz, isso significa que a matriz não deve ser de comprimento zero e todos os elementos da matriz devem ser não-nulo.) Uma finalidade de cadeia de caracteres vazia é tecnicamente permitida, mas não é recomendada.
- Argumentos de duas finalidades são equivalentes se e somente se eles contêm as mesmas cadeias de caracteres (usando um comparador ordinal) na mesma ordem. Um argumento de finalidade única é equivalente à matriz de elemento único fins correspondente.
- Dois `IDataProtector` objetos são equivalentes e apenas se eles são criados a partir equivalente `IDataProtectionProvider` objetos com parâmetros de fins equivalente.
- Para um determinado `IDataProtector` objeto, uma chamada para `Unprotect(protectedData)` retornará original `unprotectedData` se e somente se `protectedData := Protect(unprotectedData)` para um equivalente `IDataProtector` objeto.

## NOTE

Não estamos considerando o caso onde algum componente intencionalmente escolhe uma cadeia de caracteres de finalidade que é conhecida em conflito com outro componente. Esse componente essencialmente seria considerado mal-intencionados e este sistema não se destina a fornecer garantias de segurança que um código mal-intencionado já está em execução dentro do processo de trabalho.

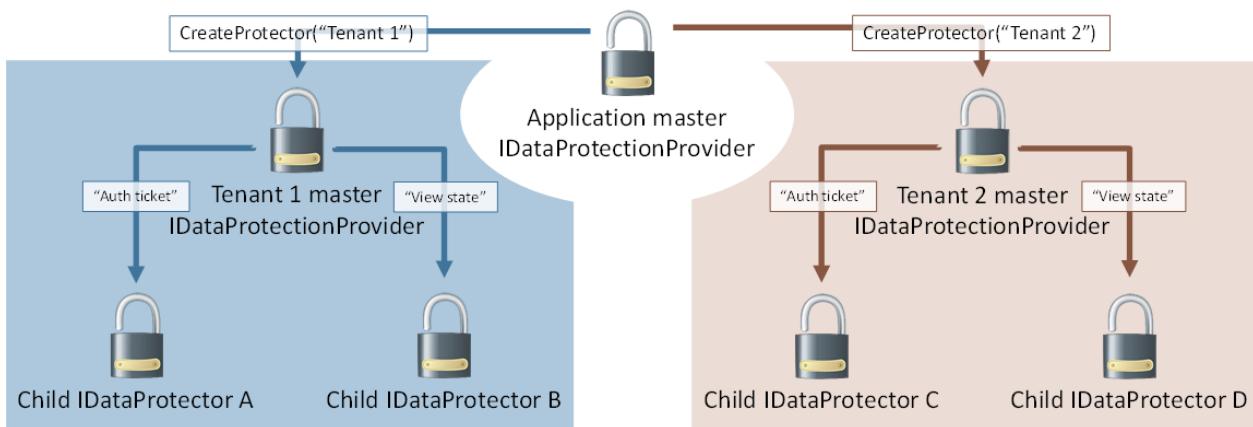
# Hierarquia de finalidade e a multilocação no ASP.NET Core

23/08/2018 • 3 minutes to read • [Edit Online](#)

Uma vez que um `IDataProtector` também é implicitamente um `IDataProtectionProvider`, fins podem ser encadeada juntos. Nesse sentido `provider.CreateProtector(["purpose1", "purpose2"])` é equivalente a `provider.CreateProtector("purpose1").CreateProtector("purpose2")`.

Isso permite que algumas relações hierárquicas interessantes através do sistema de proteção de dados. No exemplo anterior de `Contoso.Messaging.SecureMessage`, o componente `SecureMessage` pode chamar `provider.CreateProtector("Contoso.Messaging.SecureMessage")` iniciais de uma vez e armazenar em cache o resultado em uma particular `_myProvider` campo. Protetores de futuros, em seguida, podem ser criadas por meio de chamadas para `_myProvider.CreateProtector("User: username")`, e os protetores seriam usados para proteger as mensagens individuais.

Isso também pode ser invertido. Considere um único aplicativo lógico que hospeda vários locatários (um CMS parece razoável) e cada locatário podem ser configurado com seu próprio sistema de gerenciamento de autenticação e o estado. O aplicativo guarda-chuva tem um único provedor mestre e, em seguida, chama `provider.CreateProtector("Tenant 1")` e `provider.CreateProtector("Tenant 2")` para fornecer seu próprio fatia isolada do sistema de proteção de dados de cada locatário. Os locatários, em seguida, pode derivar seus próprios protetores individuais com base em suas necessidades, mas, independentemente de como eles tentam eles não é possível criar protetores que entrem em conflito com qualquer outro locatário no sistema. Graficamente, isso é representado como abaixo.



## WARNING

Isso pressupõe que a cobertura do controles de aplicativo, quais APIs estão disponíveis para locatários individuais e que os locatários não podem executar código arbitrário no servidor. Se um locatário pode executar código arbitrário, eles poderiam executar a reflexão privada para quebrar as garantias de isolamento, ou pode apenas ler o material de chave mestra diretamente e derivar qualquer subchaves que desejarem.

Na verdade, o sistema de proteção de dados usa uma espécie de multilocação em sua configuração de out-of-the-box do padrão. Por padrão, o material de chave mestra é armazenado na pasta de perfil do usuário da conta de processo do operador (ou o registro, para identidades do pool de aplicativos IIS). Mas é, na verdade, bastante comum usar uma única conta para executar vários aplicativos e, portanto, todos esses aplicativos acabam compartilhando a material da chave mestra. Para resolver isso, o sistema de proteção de dados insere

automaticamente um identificador exclusivo por aplicativo como o primeiro elemento na cadeia de finalidade geral. Que serve para essa finalidade implícita [isolar aplicativos individuais](#) uns dos outros, tratando efetivamente cada aplicativo como um locatário exclusivo no sistema e o processo de criação de protetor parece idêntico na imagem acima.

# Hash de senhas no ASP.NET Core

18/09/2018 • 2 minutes to read • [Edit Online](#)

A base de código do proteção de dados inclui um pacote `Microsoft.AspNetCore.Cryptography.KeyDerivation` que contém funções de derivação de chave de criptografia. Esse pacote é um componente autônomo e não tem nenhuma dependência no restante do sistema de proteção de dados. Ele pode ser usado independentemente completamente. A origem existe junto com o código de proteção de dados base como uma conveniência.

O pacote no momento, oferece um método `KeyDerivation.Pbkdf2` que permite o hash de uma senha usando o [PBKDF2 algoritmo](#). Essa API é muito semelhante à existente do .NET Framework [Rfc2898DeriveBytes tipo](#), mas há três diferenças importantes:

1. O `KeyDerivation.Pbkdf2` método dá suporte ao consumo PRFs vários (atualmente `HMACSHA1`, `HMACSHA256`, e `HMACSHA512`), enquanto o `Rfc2898DeriveBytes` digite dá suporte apenas à `HMACSHA1`.
2. O `KeyDerivation.Pbkdf2` método detecta o sistema operacional atual e tenta escolher a implementação mais otimizada de rotina, fornecendo um desempenho muito melhor em alguns casos. (No Windows 8, ele oferece cerca de 10 vezes a taxa de transferência `Rfc2898DeriveBytes`.)
3. O `KeyDerivation.Pbkdf2` método requer que o chamador especificar todos os parâmetros (salt, PRF e contagem de iteração). O `Rfc2898DeriveBytes` tipo fornece valores padrão para eles.

```
using System;
using System.Security.Cryptography;
using Microsoft.AspNetCore.Cryptography.KeyDerivation;

public class Program
{
    public static void Main(string[] args)
    {
        Console.Write("Enter a password: ");
        string password = Console.ReadLine();

        // generate a 128-bit salt using a secure PRNG
        byte[] salt = new byte[128 / 8];
        using (var rng = RandomNumberGenerator.Create())
        {
            rng.GetBytes(salt);
        }
        Console.WriteLine($"Salt: {Convert.ToBase64String(salt)}");

        // derive a 256-bit subkey (use HMACSHA1 with 10,000 iterations)
        string hashed = Convert.ToBase64String(KeyDerivation.Pbkdf2(
            password: password,
            salt: salt,
            prf: KeyDerivationPrf.HMACSHA1,
            iterationCount: 10000,
            numBytesRequested: 256 / 8));
        Console.WriteLine($"Hashed: {hashed}");
    }
}

/*
 * SAMPLE OUTPUT
 *
 * Enter a password: Xtw9NMgx
 * Salt: NZsP6NnmfBuYeJrrAKNuVQ==
 * Hashed: /Ooo0er10+tGwTRDTrQSoeCxVTFr6dtYly7d0cPxIak=
 */
```

Consulte a [código-fonte](#) para o ASP.NET Core Identity `PasswordHasher` caso de uso de tipo para um mundo real.

# Limite o tempo de vida das cargas protegidos no núcleo do ASP.NET

22/06/2018 • 5 minutes to read • [Edit Online](#)

Há cenários onde quer que o desenvolvedor do aplicativo para criar uma carga protegida que expira após um período de tempo definido. Por exemplo, a carga protegida pode representar um token de redefinição de senha que deve ser válido somente por uma hora. É certamente possível para o desenvolvedor criar seu próprio formato de carga que contém uma data de expiração incorporados e os desenvolvedores avançados talvez queira fazer isso mesmo assim, mas para a maioria dos desenvolvedores gerenciar esses expirações pode crescer entediante.

Para facilitar isso para nossos público de desenvolvedor, o pacote [Microsoft.AspNetCore.DataProtection.Extensions](#) contém APIs do utilitário para criar cargas expirarem automaticamente após um período de tempo definido. Essas APIs de suspensão do `ITimeLimitedDataProtector` tipo.

## Uso de API

O `ITimeLimitedDataProtector` interface é a interface principal para proteger e ao desproteger cargas de tempo limitado / expiração automática. Para criar uma instância de um `ITimeLimitedDataProtector`, você precisará de uma instância de uma expressão `IDataProtector` construído com uma finalidade específica. Uma vez o `IDataProtector` instância estiver disponível, chame o `IDataProtector.ToTimeLimitedDataProtector` método de extensão para retornar um protetor com recursos internos de expiração.

`ITimeLimitedDataProtector` apresenta os seguintes métodos de extensão e de superfície de API:

- `CreateProtector` (objetivo de cadeia de caracteres): `ITimeLimitedDataProtector` - esta API é semelhante à existente `IDataProtectionProvider.CreateProtector` em que ele pode ser usado para criar **finalidade cadeias** de um protetor de tempo limite de raiz.
- `Proteger` (`byte []` texto sem formatação, expiração de `DateTimeOffset`): `byte[]`
- `Proteger` (`texto sem formatação do byte []`, tempo de vida de `TimeSpan`): `byte[]`
- `Proteger` (`byte []` texto sem formatação): `byte[]`
- `Proteger` (`cadeia de caracteres em texto sem formatação, expiração de DateTimeOffset`): `cadeia de caracteres`
- `Proteger` (`texto sem formatação de cadeia de caracteres, tempo de vida de TimeSpan`): `cadeia de caracteres`
- `Proteger` (`cadeia de caracteres em texto sem formatação`): `cadeia de caracteres`

Além das principais `Protect` métodos que levam apenas o texto não criptografado, não há novas sobrecargas que permitem especificar a data de validade da carga. A data de validade pode ser especificada como uma data absoluta (por meio de um `DateTimeOffset`) ou como uma hora relativa (do sistema atual de tempo, por meio de um `TimeSpan`). Se uma sobrecarga que não tem uma expiração é chamada, a carga será considerada nunca para expirar.

- `Desproteger` (`byte []` `protectedData`, limite de expiração de `DateTimeOffset`): `byte[]`
- `Desproteger` (`byte []` `protectedData`): `byte[]`
- `Desproteger` (`protectedData` de `cadeia de caracteres`, limite de expiração de `DateTimeOffset`): `cadeia de caracteres`

- Desproteger (cadeia de caracteres protectedData): cadeia de caracteres

O `Unprotect` métodos retornam os dados desprotegidos originais. Se a carga ainda não tiver expirado, a expiração absoluta é retornada como um parâmetro junto com os dados desprotegidos originais ou opcional. Se a carga tiver expirada, todas as sobrecargas do método Desproteger lançará `CryptographicException`.

#### WARNING

Ele não tem recomendável usar essas APIs para proteger cargas que requerem a persistência de longo prazo ou indefinida. "Pode suportar para as cargas protegidas sejam irrecuperáveis permanentemente após um mês?" pode servir como uma boa regra prática; Se a resposta for nenhum desenvolvedores, em seguida, considere APIs alternativas.

O exemplo abaixo usa o [caminhos de código não DI](#) para instanciar o sistema de proteção de dados. Para executar este exemplo, certifique-se de que você adicionou uma referência ao pacote `Microsoft.AspNetCore.DataProtection.Extensions` primeiro.

```
using System;
using System.IO;
using System.Threading;
using Microsoft.AspNetCore.DataProtection;

public class Program
{
    public static void Main(string[] args)
    {
        // create a protector for my application

        var provider = DataProtectionProvider.Create(new DirectoryInfo(@"c:\myapp-keys\"));  

        var baseProtector = provider.CreateProtector("Contoso.TimeLimitedSample");

        // convert the normal protector into a time-limited protector
        var timeLimitedProtector = baseProtector.ToTimeLimitedDataProtector();

        // get some input and protect it for five seconds
        Console.Write("Enter input: ");
        string input = Console.ReadLine();
        string protectedData = timeLimitedProtector.Protect(input, lifetime: TimeSpan.FromSeconds(5));
        Console.WriteLine($"Protected data: {protectedData}");

        // unprotect it to demonstrate that round-tripping works properly
        string roundtripped = timeLimitedProtector.Unprotect(protectedData);
        Console.WriteLine($"Round-tripped data: {roundtripped}");

        // wait 6 seconds and perform another unprotect, demonstrating that the payload self-expires
        Console.WriteLine("Waiting 6 seconds...");
        Thread.Sleep(6000);
        timeLimitedProtector.Unprotect(protectedData);
    }
}

/*
 * SAMPLE OUTPUT
 *
 * Enter input: Hello!
 * Protected data: CfDJ8Hu5z0zwxn...nLk70k
 * Round-tripped data: Hello!
 * Waiting 6 seconds...
 * <<throws CryptographicException with message 'The payload expired at ...'>>
 */
```

# Desproteger cargas cujas chaves foram revogadas no ASP.NET Core

26/10/2018 • 5 minutes to read • [Edit Online](#)

As APIs de proteção de dados do ASP.NET Core não destinam principalmente para persistência indefinida de cargas confidenciais. Outras tecnologias, como [DPAPI do Windows CNG e do Azure Rights Management](#) são mais adequados para o cenário de armazenamento indefinido, e eles têm recursos de gerenciamento de chave forte de forma correspondente. Dito isso, não há nada proibindo um desenvolvedor que usa as APIs de proteção de dados do ASP.NET Core para proteção de longo prazo de dados confidenciais. As chaves nunca são removidas do anel de chave, portanto, `IDataProtector.Unprotect` sempre pode recuperar conteúdos existentes desde que as chaves estão disponíveis e válido.

No entanto, um problema surge quando o desenvolvedor tenta desproteger os dados que foi protegidos com uma chave revogada, como `IDataProtector.Unprotect` lançará uma exceção, nesse caso. Isso pode ser adequado para cargas de curta duração ou transitórias (como tokens de autenticação), pois esses tipos de cargas podem ser facilmente recriados pelo sistema e, na pior das hipóteses, o visitante do site pode ser necessário fazer logon novamente. Mas para cargas persistentes, tendo `Unprotect` throw pode levar à perda de dados inaceitável.

## IPersistedDataProtector

Para suportar o cenário de permitir que os conteúdos a serem desprotegidos mesmo diante de chaves revogadas, o sistema de proteção de dados contém um `IPersistedDataProtector` tipo. Para obter uma instância de `IPersistedDataProtector`, simplesmente obtém uma instância do `IDataProtector` no modo normal e tente conversão a `IDataProtector` para `IPersistedDataProtector`.

### NOTE

Nem todos os `IDataProtector` instâncias podem ser convertidas em `IPersistedDataProtector`. Os desenvolvedores devem usar o C# operador ou semelhante ou evitar exceções de tempo de execução causado por conversões inválidas, e eles devem estar preparados para tratar o caso de falha de forma adequada.

`IPersistedDataProtector` expõe a superfície de API a seguir:

```
DangerousUnprotect(byte[] protectedData, bool ignoreRevocationErrors,
    out bool requiresMigration, out bool wasRevoked) : byte[]
```

Essa API usa o conteúdo protegido (como uma matriz de bytes) e retorna a carga desprotegida. Não há nenhuma sobrecarga baseada em cadeia de caracteres. Os dois parâmetros de saída são da seguinte maneira.

- `requiresMigration` : será definido como true se a chave usada para proteger esse conteúdo não é mais a chave padrão do Active Directory, por exemplo, a chave usada para proteger essa carga é antiga e uma chave sem interrupção operação tem desde ocorrido. O chamador poderá considerar o proteger novamente a carga, dependendo de suas necessidades de negócios.
- `wasRevoked` : será definido como true se a chave usada para proteger este conteúdo foi revogada.

## WARNING

Tenha bastante cuidado ao passar `ignoreRevocationErrors: true` para o `DangerousUnprotect` método. Se, depois de chamar esse método o `wasRevoked` valor for true, em seguida, a chave usada para proteger este conteúdo foi revogada e autenticidade do conteúdo deve ser tratada como suspeito. Nesse caso, apenas continue a operar na carga de desprotegidos se você tiver alguma garantia separada que é autêntico, por exemplo, se ele é proveniente de um banco de dados seguro em vez de que está sendo enviada por um cliente da web não confiável.

```
using System;
using System.IO;
using System.Text;
using Microsoft.AspNetCore.DataProtection;
using Microsoft.AspNetCore.DataProtection.KeyManagement;
using Microsoft.Extensions.DependencyInjection;

public class Program
{
    public static void Main(string[] args)
    {
        var serviceCollection = new ServiceCollection();
        serviceCollection.AddDataProtection()
            // point at a specific folder and use DPAPI to encrypt keys
            .PersistKeysToFileSystem(new DirectoryInfo(@"c:\temp-keys"))
            .ProtectKeysWithDpapi();
        var services = serviceCollection.BuildServiceProvider();

        // get a protector and perform a protect operation
        var protector = services.GetDataProtector("Sample.DangerousUnprotect");
        Console.WriteLine("Input: ");
        byte[] input = Encoding.UTF8.GetBytes(Console.ReadLine());
        var protectedData = protector.Protect(input);
        Console.WriteLine($"Protected payload: {Convert.ToBase64String(protectedData)}");

        // demonstrate that the payload round-trips properly
        var roundTripped = protector.Unprotect(protectedData);
        Console.WriteLine($"Round-tripped payload: {Encoding.UTF8.GetString(roundTripped)}");

        // get a reference to the key manager and revoke all keys in the key ring
        var keyManager = services.GetService<IKeyManager>();
        Console.WriteLine("Revoking all keys in the key ring...");
        keyManager.RevokeAllKeys(DateTimeOffset.Now, "Sample revocation.");

        // try calling Protect - this should throw
        Console.WriteLine("Calling Unprotect...");
        try
        {
            var unprotectedPayload = protector.Unprotect(protectedData);
            Console.WriteLine($"Unprotected payload: {Encoding.UTF8.GetString(unprotectedPayload)}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"{ex.GetType().Name}: {ex.Message}");
        }

        // try calling DangerousUnprotect
        Console.WriteLine("Calling DangerousUnprotect...");
        try
        {
            IPersistedDataProtector persistedProtector = protector as IPersistedDataProtector;
            if (persistedProtector == null)
            {
                throw new Exception("Can't call DangerousUnprotect.");
            }
        }

        bool requiresMigration, wasRevoked;
```

```
        var unprotectedPayload = persistedProtector.DangerousUnprotect(
            protectedData: protectedData,
            ignoreRevocationErrors: true,
            requiresMigration: out requiresMigration,
            wasRevoked: out wasRevoked);
        Console.WriteLine($"Unprotected payload: {Encoding.UTF8.GetString(unprotectedPayload)}");
        Console.WriteLine($"Requires migration = {requiresMigration}, was revoked = {wasRevoked}");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"{ex.GetType().Name}: {ex.Message}");
    }
}

/*
 * SAMPLE OUTPUT
 *
 * Input: Hello!
 * Protected payload: CfDJ8LH1zUCX1ZVBn2BZ...
 * Round-tripped payload: Hello!
 * Revoking all keys in the key ring...
 * Calling Unprotect...
 * CryptographicException: The key {...} has been revoked.
 * Calling DangerousUnprotect...
 * Unprotected payload: Hello!
 * Requires migration = True, was revoked = True
*/

```

# Configuração da Proteção de Dados no ASP.NET Core

21/06/2018 • 2 minutes to read • [Edit Online](#)

Consulte estes tópicos para obter informações sobre a configuração da Proteção de Dados no ASP.NET Core:

- [Configurar a proteção de dados do ASP.NET Core](#)  
Uma visão geral sobre como configurar a Proteção de Dados do ASP.NET Core.
- [Tempo de vida e gerenciamento de chaves da Proteção de Dados](#)  
Informações sobre tempo de vida e gerenciamento de chaves da Proteção de Dados.
- [Política de suporte da Proteção de Dados em todo o computador](#)  
Detalhes de como definir uma política padrão em todo o computador para todos os aplicativos que usam a Proteção de Dados.
- [Cenários sem reconhecimento de DI para a Proteção de Dados no ASP.NET Core](#)  
Como usar o tipo concreto `DataProtectionProvider` para usar a Proteção de Dados sem passar por caminhos de código específicos de DI.

# Configurar a proteção de dados do ASP.NET Core

22/01/2019 • 19 minutes to read • [Edit Online](#)

Quando o sistema de proteção de dados é inicializado, ele se aplica [as configurações padrão](#) com base no ambiente operacional. Essas configurações são geralmente apropriadas para aplicativos em execução em um único computador. Há casos em que um desenvolvedor talvez queira alterar as configurações padrão:

- O aplicativo é distribuído entre várias máquinas.
- Por motivos de conformidade.

Para esses cenários, o sistema de proteção de dados oferece uma API de configuração avançada.

## WARNING

Semelhante aos arquivos de configuração, o anel de chave de proteção de dados devem ser protegido usando as permissões apropriadas. Você pode optar por criptografar as chaves em repouso, mas isso não impede que os invasores desde a criação de novas chaves. Consequentemente, a segurança desse aplicativo é afetada. O local de armazenamento configurado com proteção de dados deve ter seu acesso limitado ao próprio aplicativo, semelhante à forma como você protegerá os arquivos de configuração. Por exemplo, se você optar por armazenar seu token de autenticação no disco, use as permissões do sistema de arquivos. Certifique-se apenas a identidade sob a qual seu aplicativo web é executado tem ler, gravar e criar acesso a esse diretório. Se você usar o armazenamento de tabelas do Azure, apenas o aplicativo web deve ter a capacidade de ler, gravar ou criar novas entradas no repositório de tabela, etc.

O método de extensão `AddDataProtection` retorna um `IDataProtectionBuilder`. `IDataProtectionBuilder` expõe métodos de extensão que você pode encadear para configurar a proteção de dados de opções.

## ProtectKeysWithAzureKeyVault

Para armazenar as chaves na [Azure Key Vault](#), configure o sistema com `ProtectKeysWithAzureKeyVault` no `Startup` classe:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDataProtection()
        .PersistKeysToAzureBlobStorage(new Uri("<blobUriWithSasToken>"))
        .ProtectKeysWithAzureKeyVault("<keyIdentifier>", "<clientId>", "<clientSecret>");
}
```

Defina o local de armazenamento do anel de chave (por exemplo, `PersistKeysToAzureBlobStorage`). O local deve ser definido porque chamando `ProtectKeysWithAzureKeyVault` implementa uma `IXmlEncryptor` que desabilita as configurações de proteção automática de dados, incluindo o local de armazenamento do anel de chave. O exemplo anterior usa o armazenamento de BLOBs do Azure para persistir o anel de chave. Para obter mais informações, consulte [principais provedores de armazenamento: O Azure e Redis](#). Também é possível persistir o token de autenticação localmente com `PersistKeysToFileSystem`.

O `keyIdentifier` é o identificador de chave de Cofre de chaves usado para criptografia de chave (por exemplo, <https://contosokeyvault.vault.azure.net/keys/dataprotection/>).

`ProtectKeysWithAzureKeyVault` sobrecargas:

- [ProtectKeysWithAzureKeyVault](#) ([IDataProtectionBuilder](#), [KeyVaultClient](#), [String](#)) permite o uso de um [KeyVaultClient](#) para permitir que o sistema de proteção de dados usar o Cofre de chaves.
- [ProtectKeysWithAzureKeyVault](#) ([IDataProtectionBuilder](#), cadeia de caracteres, cadeia de caracteres, [X509Certificate2](#)) permite o uso de um [ClientId](#) e [X509Certificate](#) para permitir que o sistema de proteção de dados usar o Cofre de chaves.
- [ProtectKeysWithAzureKeyVault](#) ([IDataProtectionBuilder](#), [String](#), [String](#), [String](#)) permite o uso de um [ClientId](#) e [ClientSecret](#) para permitir que o sistema de proteção de dados usar o Cofre de chaves.

## PersistKeysToFileSystem

Para armazenar chaves em um compartilhamento UNC, em vez da a % LOCALAPPDATA % local padrão, configure o sistema com [PersistKeysToFileSystem](#):

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDataProtection()
        .PersistKeysToFileSystem(new DirectoryInfo(@"\\server\share\directory\""));
}
```

### WARNING

Se você alterar o local de persistência de chave, o sistema não criptografa automaticamente os chaves em repouso, já que ele não sabe se o DPAPI é um mecanismo de criptografia apropriado.

## ProtectKeysWith\*

Você pode configurar o sistema para proteger as chaves em repouso chamando o [ProtectKeysWith\\*](#) APIs de configuração. Considere o exemplo a seguir, que armazena as chaves em um compartilhamento UNC e criptografa essas chaves em repouso com um certificado x. 509 específico:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDataProtection()
        .PersistKeysToFileSystem(new DirectoryInfo(@"\\server\share\directory\""))
        .ProtectKeysWithCertificate("thumbprint");
}
```

No ASP.NET Core 2.1 ou posterior, você pode fornecer um [X509Certificate2](#) à [ProtectKeysWithCertificate](#), tais como carregar um certificado de um arquivo:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDataProtection()
        .PersistKeysToFileSystem(new DirectoryInfo(@"\\server\share\directory\""))
        .ProtectKeysWithCertificate(
            new X509Certificate2("certificate.pfx", "password"));
}
```

Ver [chave de criptografia em repouso](#) para obter mais exemplos e uma discussão sobre os mecanismos de criptografia de chave interna.

## UnprotectKeysWithAnyCertificate

No ASP.NET Core 2.1 ou posterior, você pode girar certificados e descriptografar chaves em repouso

usando uma matriz de [X509Certificate2](#) certificados com [UnprotectKeysWithAnyCertificate](#):

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDataProtection()
        .PersistKeysToFileSystem(new DirectoryInfo(@"\\server\share\directory\""))
        .ProtectKeysWithCertificate(
            new X509Certificate2("certificate.pfx", "password"));
    .UnprotectKeysWithAnyCertificate(
        new X509Certificate2("certificate_old_1.pfx", "password_1"),
        new X509Certificate2("certificate_old_2.pfx", "password_2"));
}
```

## SetDefaultKeyLifetime

Para configurar o sistema para usar uma vida útil de 14 dias, em vez do padrão 90 dias, use [SetDefaultKeyLifetime](#):

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDataProtection()
        .SetDefaultKeyLifetime(TimeSpan.FromDays(14));
}
```

## SetApplicationName

Por padrão, o sistema de proteção de dados isola os aplicativos uns dos outros com base em seus caminhos de conteúdo raiz, mesmo que compartilham o mesmo repositório de chave físico. Isso impede que os aplicativos Noções básicas sobre cargas protegidas uns dos outros.

Compartilhar protegido cargas entre aplicativos:

- Configurar [SetApplicationName](#) em cada aplicativo com o mesmo valor.
- Use a mesma versão da pilha de API de proteção de dados entre os aplicativos. Realizar **qualquer** das seguintes opções em arquivos de projeto de aplicativos:
  - A mesma versão da estrutura compartilhada por meio de referência a [metapacote Microsoft](#).
  - Referência à mesma [pacote de proteção de dados](#) versão.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDataProtection()
        .SetApplicationName("shared app name");
}
```

## DisableAutomaticKeyGeneration

Você pode ter um cenário onde você não deseja que um aplicativo para reverter automaticamente chaves (criar novas chaves), como eles abordam a expiração. Um exemplo disso pode ser configurados em uma relação primária/secundária, em que apenas o aplicativo principal é responsável por questões de gerenciamento de chaves e aplicativos secundários simplesmente tem uma exibição somente leitura do anel de chave de aplicativos. Os aplicativos secundários podem ser configurados para tratar o anel de chave como somente leitura ao configurar o sistema com [DisableAutomaticKeyGeneration](#):

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDataProtection()
        .DisableAutomaticKeyGeneration();
}
```

## Isolamento por aplicativo

Quando o sistema de proteção de dados é fornecido por um host do ASP.NET Core, ele automaticamente isola os aplicativos uns dos outros, mesmo que esses aplicativos estão em execução na mesma conta de processo de trabalho e estiver usando o mesmo material de chave mestra. Isso é um pouco semelhante ao modificador `IsolateApps` partir do System. Web `<machineKey>` elemento.

O mecanismo de isolamento funciona considerando cada aplicativo no computador local como um locatário exclusivo, portanto, o [IDataProtector](#) com raiz para qualquer aplicativo em particular automaticamente inclui a ID do aplicativo como um discriminador. A ID do aplicativo exclusivo vem de um dos dois locais:

1. Se o aplicativo estiver hospedado no IIS, o identificador exclusivo é o caminho de configuração do aplicativo. Se um aplicativo é implantado em um ambiente de farm da web, esse valor deve ser estável, supondo que os ambientes de IIS são configurados da mesma forma em todas as máquinas no web farm.
2. Se o aplicativo não está hospedado no IIS, o identificador exclusivo é o caminho físico do aplicativo.

O identificador exclusivo é projetado para sobreviver a reinicializações — do aplicativo individual e a própria máquina.

Esse mecanismo de isolamento pressupõe que os aplicativos não são mal-intencionados. Um aplicativo mal-intencionado sempre pode afetar qualquer outro aplicativo em execução sob a mesma conta de processo de trabalho. Em um ambiente de hospedagem compartilhado em que os aplicativos são mutuamente não confiáveis, o provedor de hospedagem deve tomar medidas para garantir o isolamento do nível de sistema operacional entre aplicativos, incluindo a separação de repositórios de chave de base de aplicativos.

Se o sistema de proteção de dados não é fornecido por um host do ASP.NET Core (por exemplo, se você instanciá-la por meio de `DataProtectionProvider` tipo concreto) isolamento de aplicativo está desabilitado por padrão. Quando o isolamento de aplicativo está desabilitado, todos os aplicativos de apoio com o mesmo material de chaveamento podem compartilhar cargas desde que eles fornecem apropriado [fins](#). Para fornecer isolamento de aplicativo nesse ambiente, chame o [SetApplicationName](#) método na configuração do objeto e forneça um nome exclusivo para cada aplicativo.

## Alterando os algoritmos com `UseCryptographicAlgorithms`

A pilha da proteção de dados permite que você altere o algoritmo padrão usado por chaves recentemente geradas. A maneira mais simples de fazer isso é chamar [UseCryptographicAlgorithms](#) do retorno de chamada de configuração:

```
services.AddDataProtection()
    .UseCryptographicAlgorithms(
        new AuthenticatedEncryptorConfiguration()
    {
        EncryptionAlgorithm = EncryptionAlgorithm.AES_256_CBC,
        ValidationAlgorithm = ValidationAlgorithm.HMACSHA256
    });
}
```

```
services.AddDataProtection()
    .UseCryptographicAlgorithms(
        new AuthenticatedEncryptionSettings()
    {
        EncryptionAlgorithm = EncryptionAlgorithm.AES_256_CBC,
        ValidationAlgorithm = ValidationAlgorithm.HMACSHA256
    });
}
```

O padrão `EncryptionAlgorithm` é AES-256-CBC e o padrão `ValidationAlgorithm` é HMACSHA256. A política padrão pode ser definida por um administrador do sistema por meio de um [política de todo o computador](#), mas uma chamada explícita para `UseCryptographicAlgorithms` substitui a política padrão.

Chamar `UseCryptographicAlgorithms` permite que você especifique o algoritmo desejado em uma lista predefinida de interna. Você não precisa se preocupar sobre a implementação do algoritmo. No cenário acima, o sistema de proteção de dados tenta usar a implementação de CNG do AES se em execução no Windows. Caso contrário, ele reverterá para o gerenciado `System.Security.Cryptography.Aes` classe.

Você pode especificar manualmente uma implementação por meio de uma chamada para `UseCustomCryptographicAlgorithms`.

#### TIP

Algoritmos a alteração não afeta as chaves existentes do anel de chave. Ela afeta apenas as chaves recentemente geradas.

## Especificando algoritmos gerenciados personalizados

Para especificar algoritmos gerenciados personalizados, crie uma `ManagedAuthenticatedEncryptorConfiguration` instância que aponta para os tipos de implementação:

```
serviceCollection.AddDataProtection()
    .UseCustomCryptographicAlgorithms(
        new ManagedAuthenticatedEncryptorConfiguration()
    {
        // A type that subclasses SymmetricAlgorithm
        EncryptionAlgorithmType = typeof(Aes),

        // Specified in bits
        EncryptionAlgorithmKeySize = 256,

        // A type that subclasses KeyedHashAlgorithm
        ValidationAlgorithmType = typeof(HMACSHA256)
    });
}
```

Para especificar algoritmos gerenciados personalizados, crie uma `ManagedAuthenticatedEncryptionSettings` instância que aponta para os tipos de implementação:

```

serviceCollection.AddDataProtection()
    .UseCustomCryptographicAlgorithms(
        new ManagedAuthenticatedEncryptionSettings()
    {
        // A type that subclasses SymmetricAlgorithm
        EncryptionAlgorithmType = typeof(Aes),

        // Specified in bits
        EncryptionAlgorithmKeySize = 256,

        // A type that subclasses KeyedHashAlgorithm
        ValidationAlgorithmType = typeof(HMACSHA256)
    });

```

Geralmente, as propriedades de tipo devem apontar para concreto, implementações que pode ser instanciadas (por meio de um construtor sem parâmetros público) de [SymmetricAlgorithm](#) e [KeyedHashAlgorithm](#), embora o sistema classifica como caso especial, como alguns valores `typeof(Aes)` para sua conveniência.

#### **NOTE**

O [SymmetricAlgorithm](#) deve ter um comprimento de chave de 128 bits  $\geq$  e um tamanho de bloco de 64 bits do  $\geq$ , e ele deve oferecer suporte à criptografia de modo CBC com preenchimento PKCS #7. O [KeyedHashAlgorithm](#) deve ter um tamanho de resumo de  $>= 128$  bits, e ele deve oferecer suporte a chaves de comprimento igual ao comprimento do resumo do algoritmo de hash. O [KeyedHashAlgorithm](#) não é estritamente necessária para ser HMAC.

## **Especificando algoritmos personalizados da CNG do Windows**

Para especificar um algoritmo personalizado do CNG do Windows usando a criptografia de modo CBC com validação do HMAC, crie uma [CngCbcAuthenticatedEncryptorConfiguration](#) instância que contém as informações de algoritmos:

```

services.AddDataProtection()
    .UseCustomCryptographicAlgorithms(
        new CngCbcAuthenticatedEncryptorConfiguration()
    {
        // Passed to BCryptOpenAlgorithmProvider
        EncryptionAlgorithm = "AES",
        EncryptionAlgorithmProvider = null,

        // Specified in bits
        EncryptionAlgorithmKeySize = 256,

        // Passed to BCryptOpenAlgorithmProvider
        HashAlgorithm = "SHA256",
        HashAlgorithmProvider = null
    });

```

Para especificar um algoritmo personalizado do CNG do Windows usando a criptografia de modo CBC com validação do HMAC, crie uma [CngCbcAuthenticatedEncryptionSettings](#) instância que contém as informações de algoritmos:

```

services.AddDataProtection()
    .UseCustomCryptographicAlgorithms(
        new CngCbcAuthenticatedEncryptionSettings()
    {
        // Passed to BCryptOpenAlgorithmProvider
        EncryptionAlgorithm = "AES",
        EncryptionAlgorithmProvider = null,

        // Specified in bits
        EncryptionAlgorithmKeySize = 256,

        // Passed to BCryptOpenAlgorithmProvider
        HashAlgorithm = "SHA256",
        HashAlgorithmProvider = null
    });

```

#### **NOTE**

O algoritmo de criptografia de bloco simétricas deve ter um comprimento de chave de  $\geq 128$  bits, um tamanho de bloco de  $\geq 64$  bits, e ele deve oferecer suporte à criptografia de modo CBC com preenchimento PKCS #7. O algoritmo de hash deve ter um tamanho de resumo de  $\geq 128$  bits e deve oferecer suporte a que está sendo aberto com o BCRYPT\_ALG\_MANIPULAR\_HMAC\_sinalizador de sinalizador. O \*propriedades do provedor podem ser definidas como nulas para usar o provedor padrão para o algoritmo especificado. Consulte a [BCryptOpenAlgorithmProvider](#) documentação para obter mais informações.

Para especificar um algoritmo personalizado do CNG do Windows usando a criptografia de modo Galois/contador com a validação, crie uma [CngGcmAuthenticatedEncryptorConfiguration](#) instância que contém as informações de algoritmos:

```

services.AddDataProtection()
    .UseCustomCryptographicAlgorithms(
        new CngGcmAuthenticatedEncryptorConfiguration()
    {
        // Passed to BCryptOpenAlgorithmProvider
        EncryptionAlgorithm = "AES",
        EncryptionAlgorithmProvider = null,

        // Specified in bits
        EncryptionAlgorithmKeySize = 256
    });

```

Para especificar um algoritmo personalizado do CNG do Windows usando a criptografia de modo Galois/contador com a validação, crie uma [CngGcmAuthenticatedEncryptionSettings](#) instância que contém as informações de algoritmos:

```

services.AddDataProtection()
    .UseCustomCryptographicAlgorithms(
        new CngGcmAuthenticatedEncryptionSettings()
    {
        // Passed to BCryptOpenAlgorithmProvider
        EncryptionAlgorithm = "AES",
        EncryptionAlgorithmProvider = null,

        // Specified in bits
        EncryptionAlgorithmKeySize = 256
    });

```

#### **NOTE**

O algoritmo de criptografia de bloco simétricas deve ter um comprimento de chave de  $> = 128$  bits, um tamanho de bloco de exatamente de 128 bits, e ele deve oferecer suporte a criptografia do GCM. Você pode definir as [EncryptionAlgorithmProvider](#) propriedade como nulo para usar o provedor padrão para o algoritmo especificado. Consulte a [BCryptOpenAlgorithmProvider](#) documentação para obter mais informações.

#### **Especificar outros algoritmos personalizados**

Embora não é exposta como uma API de primeira classe, o sistema de proteção de dados é extensível o suficiente para permitir a especificação de praticamente qualquer tipo de algoritmo. Por exemplo, é possível manter todas as chaves contidas dentro do módulo de segurança um Hardware (HSM) e fornecer uma implementação personalizada do núcleo rotinas de criptografia e descriptografia. Ver [IAuthenticatedEncryptor](#) na [principais de extensibilidade da criptografia](#) para obter mais informações.

## **Chaves persistentes ao hospedar em um contêiner do Docker**

Ao hospedar em um [Docker](#) contêiner, as chaves devem ser mantidas em qualquer um:

- Uma pasta que é um volume do Docker que persiste além do tempo de vida do contêiner, como um volume compartilhado ou um volume montado pelo host.
- Um provedor externo, como [Azure Key Vault](#) ou [Redis](#).

## **Recursos adicionais**

- [Cenários de reconhecimento não DI para proteção de dados no ASP.NET Core](#)
- [Suporte a política de máquina de proteção de dados no ASP.NET Core](#)
- [Hospedar o ASP.NET Core em um web farm](#)

# Gerenciamento de chaves de proteção de dados e o tempo de vida no ASP.NET Core

18/07/2018 • 5 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

## Gerenciamento de chaves

O aplicativo tenta detectar seu ambiente operacional e lidar com a configuração de chave por conta própria.

1. Se o aplicativo estiver hospedado em [aplicativos do Azure](#), as chaves são persistidas para o `%HOME%\ASP.NET\DataProtection-Keys` pasta. Essa pasta é apoiada pelo repositório de rede e é sincronizada em todos os computadores que hospedam o aplicativo.
  - As chaves não são protegidas em repouso.
  - O *DataProtection* chaves pasta fornece o anel de chave a todas as instâncias de um aplicativo em um único slot de implantação.
  - Slots de implantação separados, como de preparo e produção, não compartilham um anel de chave. Quando você alternar entre os slots de implantação, por exemplo, trocar o preparo e produção ou usando um teste a / B, qualquer aplicativo usando a proteção de dados não será capaz de descriptografar dados armazenados usando o anel de chave dentro do slot anterior. Isso leva a usuários que estão sendo conectados fora de um aplicativo que usa a autenticação de cookie padrão do ASP.NET Core, pois ele usa a proteção de dados para proteger seus cookies. Se desejar que os anéis de chave independentes de slot, use um provedor de anel de chave externo, como o armazenamento de BLOBs Azure, Azure Key Vault, um repositório SQL ou do cache Redis.
2. Se o perfil do usuário estiver disponível, as chaves são persistidas para o `%LOCALAPPDATA%\ASP.NET\DataProtection-Keys` pasta. Se o sistema operacional for Windows, as chaves são criptografadas em repouso usando a DPAPI.
3. Se o aplicativo estiver hospedado no IIS, as chaves são mantidas no registro HKLM em uma chave de registro especial que tem a ACL acessível apenas para a conta de processo de trabalho. As chaves são criptografadas em repouso usando a DPAPI.
4. Se nenhuma dessas condições corresponderem, as chaves não são persistidas fora do processo atual. Quando o processo é encerrado, todas geradas chaves forem perdidas.

O desenvolvedor está sempre no controle total e pode substituir como e onde as chaves são armazenadas. As três primeiras opções acima devem fornecer bons padrões para a maioria dos aplicativos de forma semelhante a como o ASP.NET `<machineKey>` rotinas de geração automática funcionavam no passado. A opção de fallback final é o único cenário que exige que o desenvolvedor especifique [configuração](#) antecipado se quiserem persistência de chave, mas essa fallback só ocorre em situações raras.

Ao hospedar em um contêiner do Docker, as chaves devem ser persistentes em uma pasta que é um volume do Docker (um volume compartilhado ou um volume montado de host que persiste além do tempo de vida do contêiner) ou em um provedor externo, como [Azure Key Vault](#) ou [Redis](#). Um provedor externo também é útil em cenários de web farm se os aplicativos não podem acessar um volume compartilhado de rede (consulte [PersistKeysToFileSystem](#) para obter mais informações).

#### **WARNING**

Se o desenvolvedor substitui as regras descritas acima e aponta o sistema de proteção de dados em um repositório de chave específico, a criptografia automática de chaves em repouso está desabilitada. Proteção em repouso pode ser reabilitada por meio [configuração](#).

## Vida útil da chave

Por padrão, as chaves têm um tempo de vida de 90 dias. Quando uma chave expira, o aplicativo gera uma nova chave automaticamente e define a nova chave como a chave ativa. Chaves obsoletos permanecerão no sistema, desde que seu aplicativo pode descriptografar todos os dados protegidos com eles. Ver [gerenciamento de chaves](#) para obter mais informações.

## Algoritmos padrão

O algoritmo de proteção de conteúdo padrão usado é AES-256-CBC para confidencialidade e HMACSHA256 autenticidade. Uma chave mestra de 512 bits, alterada a cada 90 dias, é usada para derivar as duas chaves de subpropriedades usadas para esses algoritmos em uma base por carga. Ver [subchave derivação](#) para obter mais informações.

## Recursos adicionais

- [Extensibilidade de gerenciamento de chaves no ASP.NET Core](#)
- [Hospedar o ASP.NET Core em um web farm](#)

# Suporte a política de máquina de proteção de dados no ASP.NET Core

22/06/2018 • 7 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

Quando em execução no Windows, o sistema de proteção de dados tem suporte limitado para definir uma política de todo o computador padrão para todos os aplicativos que consomem a proteção de dados do ASP.NET Core. A ideia geral é que um administrador pode querer alterar uma configuração padrão, como os algoritmos usados ou a vida útil da chave, sem a necessidade de atualizar manualmente cada aplicativo no computador.

## WARNING

O administrador do sistema pode definir a política padrão, mas eles não é possível impor a ele. O desenvolvedor do aplicativo sempre pode substituir qualquer valor com um dos seus próprios escolhendo. A política padrão afeta somente os aplicativos em que o desenvolvedor não foi especificado um valor explícito para uma configuração.

## Configuração de política padrão

Para definir a política padrão, um administrador pode definir os valores conhecidos no registro do sistema na seguinte chave do registro:

**HKLM\SOFTWARE\Microsoft\DotNetPackages\Microsoft.AspNetCore.DataProtection**

Se você estiver em um sistema operacional de 64 bits e quiser afetar o comportamento de aplicativos de 32 bits, lembre-se de configurar o equivalente Wow6432Node a chave acima.

Os valores com suporte são mostrados abaixo.

VALOR	TIPO	DESCRIÇÃO
EncryptionType	cadeia de caracteres	Especifica os algoritmos que devem ser usados para proteção de dados. O valor deve ser CBC CNG, GCM CNG ou gerenciado e é descrito com mais detalhes abaixo.
DefaultKeyLifetime	DWORD	Especifica o tempo de vida para chaves geradas recentemente. O valor é especificado em dias e deve ser > = 7.
KeyEscrowSinks	cadeia de caracteres	Especifica os tipos que são usados para caução de chaves. O valor é uma lista separada por ponto-e-vírgula de Coletores de caução de chaves, onde cada elemento na lista é o nome qualificado do assembly de um tipo que implementa <a href="#">IKeyEscrowSink</a> .

## Tipos de criptografia

Se EncryptionType é CBC CNG, o sistema está configurado para usar uma codificação de bloco simétrica de

modo CBC para confidencialidade e HMAC autenticidade com serviços fornecidos pelo Windows CNG (consulte [especificando algoritmos personalizados de Windows CNG](#) para mais detalhes). Os seguintes valores adicionais são suportados, cada uma correspondendo a uma propriedade do tipo CngCbcAuthenticatedEncryptionSettings.

VALOR	TIPO	DESCRIÇÃO
EncryptionAlgorithm	cadeia de caracteres	O nome de um algoritmo de criptografia simétrica bloco entendido pelo CNG. Esse algoritmo é aberto no modo CBC.
EncryptionAlgorithmProvider	cadeia de caracteres	O nome da implementação do provedor CNG que pode produzir o algoritmo EncryptionAlgorithm.
EncryptionAlgorithmKeySize	DWORD	O comprimento (em bits) da chave derivada para o algoritmo de criptografia simétrica de bloco.
HashAlgorithm	cadeia de caracteres	O nome de um algoritmo de hash entendido pelo CNG. Esse algoritmo é aberto no modo HMAC.
HashAlgorithmProvider	cadeia de caracteres	O nome da implementação do provedor CNG que pode produzir o algoritmo HashAlgorithm.

Se EncryptionType é GCM CNG, o sistema está configurado para usar uma codificação de bloco simétrica de modo Galois/contador para autenticidade e confidencialidade com serviços fornecidos pelo Windows CNG (consulte [especificando algoritmos personalizados de Windows CNG](#) Para obter mais detalhes). Os seguintes valores adicionais são suportados, cada uma correspondendo a uma propriedade do tipo CngGcmAuthenticatedEncryptionSettings.

VALOR	TIPO	DESCRIÇÃO
EncryptionAlgorithm	cadeia de caracteres	O nome de um algoritmo de criptografia simétrica bloco entendido pelo CNG. Esse algoritmo é aberto no modo de Galois/contador.
EncryptionAlgorithmProvider	cadeia de caracteres	O nome da implementação do provedor CNG que pode produzir o algoritmo EncryptionAlgorithm.
EncryptionAlgorithmKeySize	DWORD	O comprimento (em bits) da chave derivada para o algoritmo de criptografia simétrica de bloco.

Se EncryptionType for gerenciado, o sistema está configurado para usar um SymmetricAlgorithm gerenciado para confidencialidade e KeyedHashAlgorithm autenticidade (consulte [especificando personalizado gerenciado algoritmos](#) para obter mais detalhes). Os seguintes valores adicionais são suportados, cada uma correspondendo a uma propriedade do tipo ManagedAuthenticatedEncryptionSettings.

VALOR	TIPO	DESCRIÇÃO

VALOR	TIPO	DESCRIÇÃO
EncryptionAlgorithmType	cadeia de caracteres	O nome qualificado do assembly de um tipo que implementa SymmetricAlgorithm.
EncryptionAlgorithmKeySize	DWORD	O comprimento (em bits) da chave para derivar o algoritmo de criptografia simétrica.
ValidationAlgorithmType	cadeia de caracteres	O nome qualificado do assembly de um tipo que implementa KeyedHashAlgorithm.

Se EncryptionType tiver qualquer valor diferente de nulo ou vazio, o sistema de proteção de dados gera uma exceção durante a inicialização.

#### **WARNING**

Ao configurar uma configuração de política padrão que envolve os nomes de tipo (EncryptionAlgorithmType, ValidationAlgorithmType, KeyEscrowSinks), os tipos devem estar disponíveis para o aplicativo. Isso significa que para aplicativos em execução no CLR de área de trabalho, os assemblies que contêm esses tipos devem estar presentes no Cache de Assembly Global (GAC). Para aplicativos do ASP.NET Core em execução no .NET Core, os pacotes que contêm esses tipos devem ser instalados.

# Cenários de reconhecimento não DI para proteção de dados no ASP.NET Core

22/06/2018 • 4 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

O sistema de proteção de dados do ASP.NET Core é normalmente [adicionada a um contêiner de serviço](#) e consumido por componentes dependentes por meio de injeção de dependência (DI). No entanto, há casos em que isso não é possível ou desejado, especialmente ao importar o sistema para um aplicativo existente.

Para dar suporte a esses cenários, o [Microsoft.AspNetCore.DataProtection.Extensions](#) pacote fornece um tipo concreto, [DataProtectionProvider](#), que oferece uma maneira simples de usar a proteção de dados sem depender de injeção de dependência. O `DataProtectionProvider` tipo implementa [IDataProtectionProvider](#). Construindo `DataProtectionProvider` requer apenas fornecer um  [DirectoryInfo](#) instância para indicar onde as chaves de criptografia do provedor devem ser armazenadas, como mostrado no exemplo de código a seguir:

```

using System;
using System.IO;
using Microsoft.AspNetCore.DataProtection;

public class Program
{
    public static void Main(string[] args)
    {
        // Get the path to %LOCALAPPDATA%\myapp-keys
        var destFolder = Path.Combine(
            System.Environment.GetEnvironmentVariable("LOCALAPPDATA"),
            "myapp-keys");

        // Instantiate the data protection system at this folder
        var dataProtectionProvider = DataProtectionProvider.Create(
            new DirectoryInfo(destFolder));

        var protector = dataProtectionProvider.CreateProtector("Program.No-DI");
        Console.WriteLine("Enter input: ");
        var input = Console.ReadLine();

        // Protect the payload
        var protectedPayload = protector.Protect(input);
        Console.WriteLine($"Protect returned: {protectedPayload}");

        // Unprotect the payload
        var unprotectedPayload = protector.Unprotect(protectedPayload);
        Console.WriteLine($"Unprotect returned: {unprotectedPayload}");

        Console.WriteLine();
        Console.WriteLine("Press any key...");
        Console.ReadKey();
    }
}

/*
 * SAMPLE OUTPUT
 *
 * Enter input: Hello world!
 * Protect returned: CfDJ8FWbAn6...ch3hAPm1NJA
 * Unprotect returned: Hello world!
 *
 * Press any key...
*/

```

Por padrão, o `DataProtectionProvider` tipo concreto não criptografa a chave primas persisti-los antes do sistema de arquivos. Isso é para dar suporte a cenários onde os pontos de desenvolvedor para um compartilhamento de rede e o sistema de proteção de dados não é possível deduzir automaticamente um mecanismo de criptografia de chave apropriado em repouso.

Além disso, o `DataProtectionProvider` tipo concreto não **isolar os aplicativos** por padrão. Todos os aplicativos usando o mesmo diretório principal podem compartilhar cargas, contanto que seus **finalidade parâmetros** corresponder.

O `DataProtectionProvider` construtor aceita um retorno de chamada de configuração opcional que pode ser usado para ajustar os comportamentos do sistema. O exemplo a seguir demonstra o isolamento de restauração com uma chamada explícita para `SetApplicationName`. O exemplo também demonstra como configurar o sistema para criptografar automaticamente chaves persistentes usando Windows DPAPI. Se o diretório aponta para um compartilhamento UNC, você poderá distribuir um certificado compartilhado por todos os computadores relevantes e para configurar o sistema para usar a criptografia baseada em certificado com uma chamada para `ProtectKeysWithCertificate`.

```

using System;
using System.IO;
using Microsoft.AspNetCore.DataProtection;

public class Program
{
    public static void Main(string[] args)
    {
        // Get the path to %LOCALAPPDATA%\myapp-keys
        var destFolder = Path.Combine(
            System.Environment.GetEnvironmentVariable("LOCALAPPDATA"),
            "myapp-keys");

        // Instantiate the data protection system at this folder
        var dataProtectionProvider = DataProtectionProvider.Create(
            new DirectoryInfo(destFolder),
            configuration =>
            {
                configuration.SetApplicationName("my app name");
                configuration.ProtectKeysWithDpapi();
            });

        var protector = dataProtectionProvider.CreateProtector("Program.No-DI");
        Console.Write("Enter input: ");
        var input = Console.ReadLine();

        // Protect the payload
        var protectedPayload = protector.Protect(input);
        Console.WriteLine($"Protect returned: {protectedPayload}");

        // Unprotect the payload
        var unprotectedPayload = protector.Unprotect(protectedPayload);
        Console.WriteLine($"Unprotect returned: {unprotectedPayload}");

        Console.WriteLine();
        Console.WriteLine("Press any key...");
        Console.ReadKey();
    }
}

```

#### TIP

Instâncias de `DataProtectionProvider` tipo concreto são caros de criar. Se um aplicativo mantém várias instâncias desse tipo e se eles estão usando o mesmo diretório de armazenamento de chaves, pode afetar o desempenho do aplicativo. Se você usar o `DataProtectionProvider` tipo, recomendamos que você crie esse tipo de uma vez e reutilizá-la tanto quanto possível. O `DataProtectionProvider` tipo e todos os `IDataProtector` instâncias criadas a partir dela são thread-safe para chamadores vários.

# APIs de extensibilidade de proteção de dados do ASP.NET Core

21/06/2018 • 2 minutes to read • [Edit Online](#)

- [Extensibilidade da criptografia básica](#)
- [Extensibilidade de gerenciamento de chaves](#)
- [APIs diversas](#)

# Extensibilidade da criptografia de núcleo no núcleo do ASP.NET

22/06/2018 • 11 minutes to read • [Edit Online](#)

## WARNING

Tipos que implementam qualquer uma das seguintes interfaces devem ser thread-safe para chamadores vários.

## IAuthenticatedEncryptor

O **IAuthenticatedEncryptor** interface é o bloco de construção básico do subsistema de criptografia. Em geral, há um IAuthenticatedEncryptor por chave e a instância IAuthenticatedEncryptor encapsula todas as material de chave de criptografia e algoritmos informações necessárias para executar operações criptográficas.

Como o nome sugere, o tipo é responsável por fornecer serviços de criptografia e descriptografia autenticados. Expõe as APIs a seguir.

- Descriptografar (ArraySegment texto cifrado, ArraySegment additionalAuthenticatedData): byte]
- Criptografar (ArraySegment texto sem formatação, ArraySegment additionalAuthenticatedData): byte]

O método Encrypt retorna um blob que inclui o texto não criptografado enciphered e uma marca de autenticação. A marca de autenticação deve abranger os dados adicionais autenticados (AAD), embora o AAD em si não precisa ser recuperável da carga final. O método Decrypt valida a marca de autenticação e retorna a carga deciphered.

Todas as falhas (exceto ArgumentNullException e semelhante) devem ser homogenized para CryptographicException.

## NOTE

A própria instância IAuthenticatedEncryptor, na verdade, não precisa conter o material da chave. Por exemplo, a implementação pudesse ser delegado a um HSM para todas as operações.

## Como criar um IAuthenticatedEncryptor

- [ASP.NET Core 2.x](#)
- [ASP.NET Core 1.x](#)

O **IAuthenticatedEncryptorFactory** interface representa um tipo que sabe como criar um IAuthenticatedEncryptor instância. Sua API é o seguinte.

- CreateEncryptorInstance (chave IKey): IAuthenticatedEncryptor

Para qualquer instância específica de IKey os qualquer intermediária autenticada criada por seu método CreateEncryptorInstance deve ser consideradas equivalentes, como o exemplo de código abaixo.

```

// we have an IAuthenticatedEncryptorFactory instance and an IKey instance
IAuthenticatedEncryptorFactory factory = ...;
IKey key = ...;

// get an encryptor instance and perform an authenticated encryption operation
ArraySegment<byte> plaintext = new ArraySegment<byte>(Encoding.UTF8.GetBytes("plaintext"));
ArraySegment<byte> aad = new ArraySegment<byte>(Encoding.UTF8.GetBytes("AAD"));
var encryptor1 = factory.CreateEncryptorInstance(key);
byte[] ciphertext = encryptor1.Encrypt(plaintext, aad);

// get another encryptor instance and perform an authenticated decryption operation
var encryptor2 = factory.CreateEncryptorInstance(key);
byte[] roundTripped = encryptor2.Decrypt(new ArraySegment<byte>(ciphertext), aad);

// the 'roundTripped' and 'plaintext' buffers should be equivalent

```

## IAuthenticatedEncryptorDescriptor (ASP.NET Core 2. x somente)

- [ASP.NET Core 2.x](#)
- [ASP.NET Core 1.x](#)

O **IAuthenticatedEncryptorDescriptor** interface representa um tipo que sabe como exportar em si para XML. Sua API é o seguinte.

- ExportToXml(): XmlSerializedDescriptorInfo

## Serialização XML

A principal diferença entre IAuthenticatedEncryptor e IAuthenticatedEncryptorDescriptor é que o descritor sabe como criar o Criptografador e fornecê-lo com os argumentos válidos. Considere um IAuthenticatedEncryptor cuja implementação depende do SymmetricAlgorithm e KeyedHashAlgorithm. Trabalho do Criptografador é consumir esses tipos, mas não necessariamente souber onde esses tipos de origem, para que ele realmente não é possível gravar uma descrição adequada do como recriar a mesmo se o aplicativo for reiniciado. O descritor de atua como um nível superior sobre isso. Desde que o descritor sabe como criar a instância do Criptografador (por exemplo, sabe como criar os algoritmos necessários), ele pode serializar esse conhecimento em formato XML para que a instância do Criptografador pode ser recriada após a redefinição de um aplicativo.

O descritor de pode ser serializado por meio de sua rotina de ExportToXml. Esta rotina retorna um XmlSerializedDescriptorInfo que contém duas propriedades: a representação de XElement do descritor e o tipo que representa um **IAuthenticatedEncryptorDescriptorDeserializer** que pode ser usado para lembrar Esse descritor fornecido o XElement correspondente.

O descritor de serializado pode conter informações confidenciais, como o material de chave de criptografia. O sistema de proteção de dados tem suporte interno para criptografar informações antes de ele tem persistidos para armazenamento. Para tirar proveito disso, o descritor deve marcar o elemento que contém informações confidenciais com o nome do atributo "requiresEncryption" (xmlns "<http://schemas.asp.net/2015/03/dataProtection>"), valor "true".

### TIP

Há um auxiliar de API para a configuração deste atributo. Chame o método de extensão que XElement.markasrequiresencryption() localizado no namespace Microsoft.AspNetCore.DataProtection.AuthenticatedEncryption.ConfigurationModel.

Também pode haver casos onde o descritor serializado não contêm informações confidenciais. Considere

novamente o caso de uma chave de criptografia armazenada em um HSM. Não é possível gravar o descritor de material de chave ao serializar em si, pois o HSM não expõe o material em forma de texto sem formatação. Em vez disso, o descritor pode gravar a versão de chave de linha de chave (se o HSM permite exportar dessa forma) ou o identificador exclusivo do HSM para a chave.

## IAuthenticatedEncryptorDescriptorDeserializer

O **IAuthenticatedEncryptorDescriptorDeserializer** interface representa um tipo que sabe como desserializar uma instância IAuthenticatedEncryptorDescriptor de um XElement. Ela expõe um único método:

- ImportFromXml (elemento de XElement): IAuthenticatedEncryptorDescriptor

O método ImportFromXml usa o XElement foi retornado por [IAuthenticatedEncryptorDescriptor.ExportToXml](#) e cria um equivalente a IAuthenticatedEncryptorDescriptor original.

Tipos que implementam IAuthenticatedEncryptorDescriptorDeserializer devem ter um dos dois construtores públicos a seguir:

- .ctor(IServiceProvider)
- .ctor()

### NOTE

O IServiceProvider transmitido ao construtor pode ser nulo.

## A fábrica de nível superior

- [ASP.NET Core 2.x](#)
- [ASP.NET Core 1.x](#)

O **AlgorithmConfiguration** classe representa um tipo que sabe como criar IAuthenticatedEncryptorDescriptor instâncias. Ela expõe uma única API.

- CreateNewDescriptor() : IAuthenticatedEncryptorDescriptor

Pense AlgorithmConfiguration como a fábrica de nível superior. A configuração funciona como um modelo. Ela inclui informações de algoritmos (por exemplo, essa configuração produz descritores com uma chave mestra de AES-128-GCM), mas ela ainda não está associada com uma chave específica.

Quando CreateNewDescriptor é chamado, novo material de chave é criada somente para essa chamada e um novo IAuthenticatedEncryptorDescriptor é produzido que encapsula o material de chave e as informações de algoritmos necessária para consumir o material. O material da chave pode ser criado no software (e mantido na memória), ele pode ser criado e mantido em um HSM e assim por diante. O ponto fundamental é que as duas chamadas para CreateNewDescriptor nunca devem criar instâncias de IAuthenticatedEncryptorDescriptor equivalentes.

O tipo de AlgorithmConfiguration serve como ponto de entrada para rotinas de criação da chave como [chave automática](#). Para alterar a implementação para todas as chaves futuras, defina a propriedade de AuthenticatedEncryptorConfiguration em KeyManagementOptions.

# Extensibilidade de gerenciamento de chaves no ASP.NET Core

12/12/2018 • 14 minutes to read • [Edit Online](#)

## TIP

Leia as [gerenciamento de chaves](#) seção antes de ler esta seção, pois ele explica alguns dos conceitos fundamentais dessas APIs.

## WARNING

Tipos que implementam qualquer uma das seguintes interfaces devem ser thread-safe para chamadores vários.

## Chave

O `IKey` interface é a representação básica de uma chave no sistema de criptografia. A chave do termo é usada aqui no sentido abstrato, não no sentido literal "criptográfico do material da chave". Uma chave tem as seguintes propriedades:

- Datas de expiração, criação e ativação
- Status de revogação
- Identificador de chave (um GUID)

Além disso, `IKey` expõe uma `CreateEncryptor` método que pode ser usado para criar um `IAuthenticatedEncryptor` instância vinculado a essa chave.

Além disso, `IKey` expõe uma `CreateEncryptorInstance` método que pode ser usado para criar um `IAuthenticatedEncryptor` instância vinculado a essa chave.

## NOTE

Há uma API para recuperar o material criptográfico bruto de um `IKey` instância.

## IKeyManager

O `IKeyManager` interface representa um objeto responsável pelo armazenamento de chaves geral, a recuperação e a manipulação. Ele expõe três operações de alto nível:

- Criar uma nova chave e mantê-lo no armazenamento.
- Obter todas as chaves de armazenamento.
- Revogar uma ou mais chaves e persistir as informações de revogação para o armazenamento.

## WARNING

Escrevendo um `IKeyManager` é uma tarefa muito avançada e a maioria dos desenvolvedores não deve tentá-lo. Em vez disso, a maioria dos desenvolvedores deve tirar vantagem dos recursos oferecidos pelo `XmlKeyManager` classe.

## XmlKeyManager

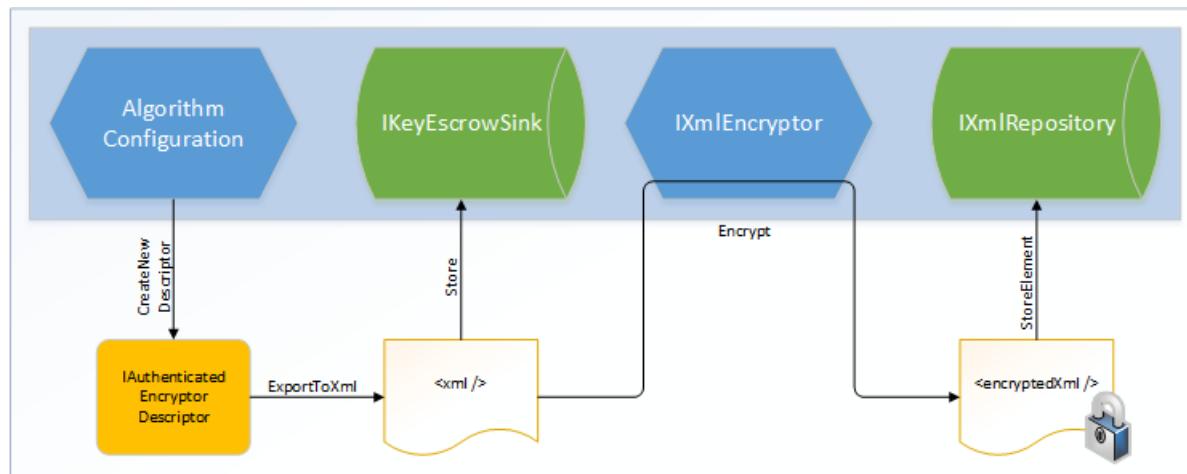
O `XmlKeyManager` tipo é a implementação concreta da caixa de entrada da `IKeyManager`. Ele fornece vários recursos úteis, incluindo caução de chaves e criptografia de chaves em repouso. Chaves nesse sistema são representadas como elementos XML (especificamente, `XElement`).

`XmlKeyManager` depende de vários outros componentes no decorrer de cumprir suas tarefas:

- `AlgorithmConfiguration`, que determina os algoritmos usados por novas chaves.
- `IRepository`, que controla onde as chaves são mantidas no armazenamento.
- `IEncryptor` [opcional], que permite criptografar as chaves em repouso.
- `IKeyEscrowSink` [opcional], que fornece serviços de caução de chaves.
- `IRepository`, que controla onde as chaves são mantidas no armazenamento.
- `IEncryptor` [opcional], que permite criptografar as chaves em repouso.
- `IKeyEscrowSink` [opcional], que fornece serviços de caução de chaves.

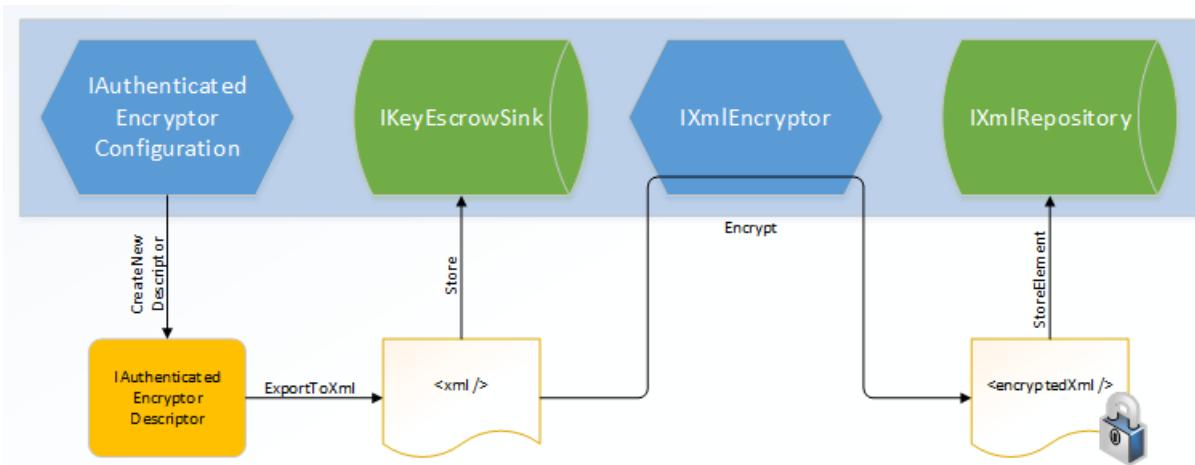
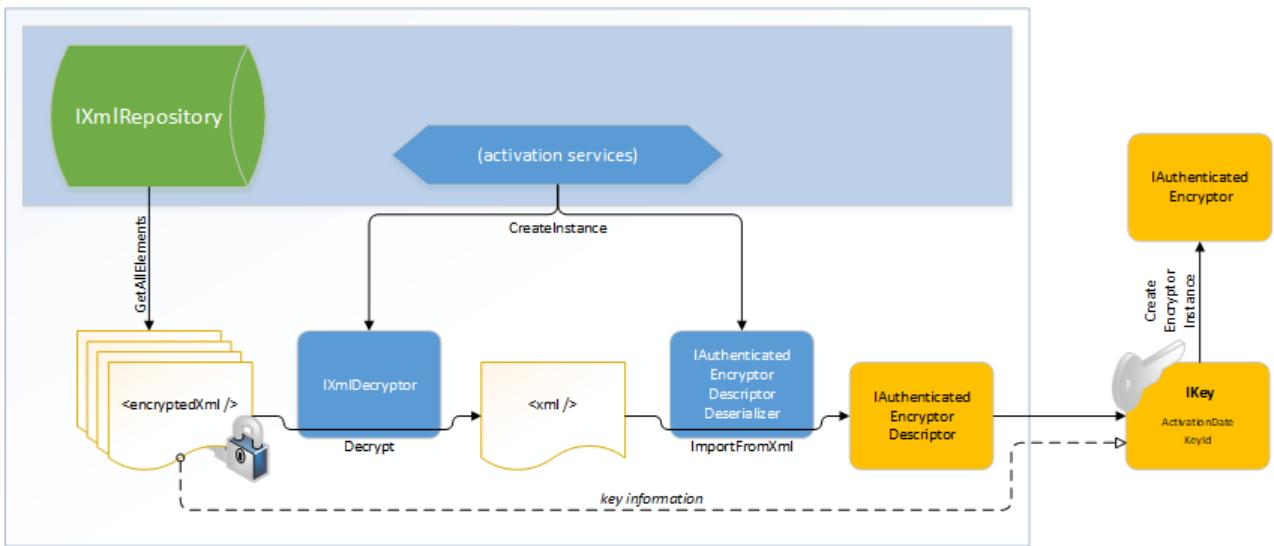
Abaixo estão os diagramas de alto nível que indicam como esses componentes são vinculados dentro

`XmlKeyManager`.



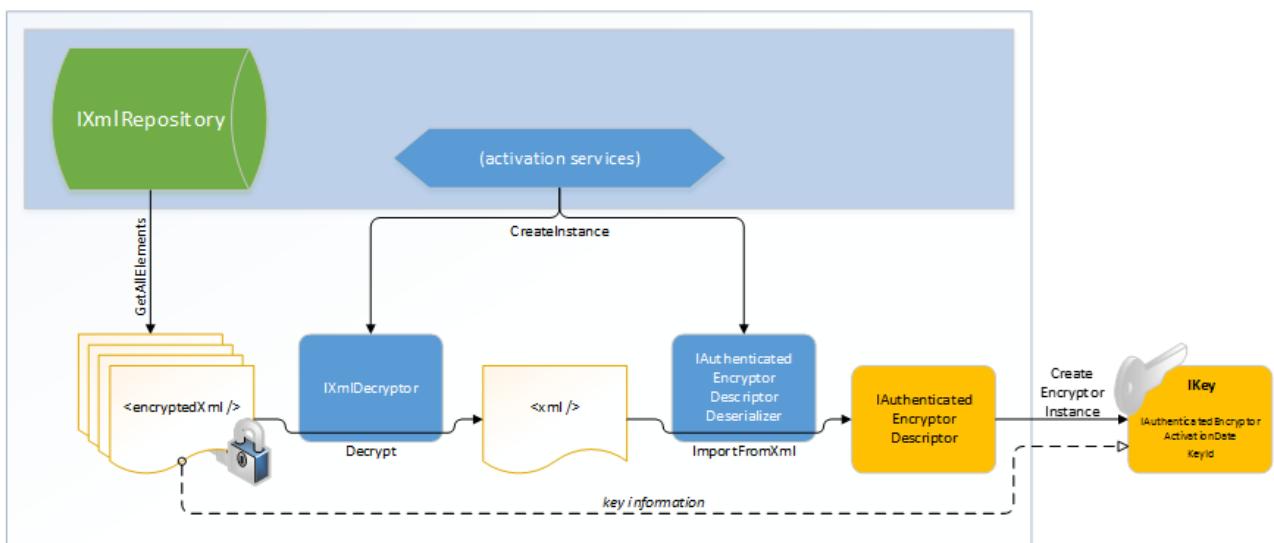
### Criação de chave / CreateNewKey

Na implementação de `CreateNewKey`, o `AlgorithmConfiguration` componente é usado para criar um único `IAuthenticatedEncryptorDescriptor`, que é então serializado como XML. Se um coletor de caução de chave estiver presente, o XML bruto de (não criptografado) é fornecido para o coletor para armazenamento de longo prazo. O XML não criptografado é executado por meio de um `IEncryptor` (se necessário) para gerar o documento XML criptografado. Este documento criptografado é persistido no armazenamento de longo prazo via o `IRepository`. (Se nenhum `IEncryptor` é configurado, o documento não criptografado é persistido no `IRepository`.)



### Criação de chave / CreateNewKey

Na implementação de `CreateNewKey`, o `IAuthenticatedEncryptorConfiguration` componente é usado para criar um único `IAuthenticatedEncryptorDescriptor`, que é então serializado como XML. Se um coletor de caução de chave estiver presente, o XML bruto de (não criptografado) é fornecido para o coletor para armazenamento de longo prazo. O XML não criptografado é executado por meio de um `IXmlEncryptor` (se necessário) para gerar o documento XML criptografado. Este documento criptografado é persistido no armazenamento de longo prazo via o `IXmlRepository`. (Se nenhum `IXmlEncryptor` é configurado, o documento não criptografado é persistido no `IXmlRepository`.)



### Recuperação de chave / GetAllKeys

Na implementação de `GetAllKeys`, o XML documenta as chaves que representa e revogações são lidas do subjacente `IXmlRepository`. Se esses documentos são criptografados, o sistema automaticamente descriptografá-los. `XmlKeyManager` cria o apropriada `IAuthenticatedEncryptorDescriptorDeserializer` instâncias para desserializar os documentos de volta para o `IAuthenticatedEncryptorDescriptor` instâncias, que, em seguida, são encapsuladas em indivíduo `IKey` instâncias. Esta coleção de `IKey` instâncias é retornado ao chamador.

Obter mais informações sobre os elementos XML específicos podem ser encontradas na [documentos de formato de armazenamento de chaves](#).

## IXmlAttributeRepository

O `IXmlAttributeRepository` interface representa um tipo que pode persistir o XML para e recuperar o XML de um armazenamento de backup. Ele expõe duas APIs:

- `GetAllElements` : `IReadOnlyCollection< XElement >`
- `StoreElement(XElement element, string friendlyName)`

Implementações de `IXmlAttributeRepository` não precisam analisar o XML passar através deles. Eles devem tratar os documentos XML como opaco e permitir que camadas superiores se preocupar sobre como gerar e analisar os documentos.

Há quatro tipos concretos internos que implementam `IXmlAttributeRepository`:

- [FileSystemXmlRepository](#)
  - [RegistryXmlRepository](#)
  - [AzureStorage.AzureBlobXmlRepository](#)
  - [RedisXmlRepository](#)
- 
- [FileSystemXmlRepository](#)
  - [RegistryXmlRepository](#)
  - [AzureStorage.AzureBlobXmlRepository](#)
  - [RedisXmlRepository](#)

Consulte a [documentos de provedores de armazenamento de chaves](#) para obter mais informações.

Registrando um personalizado `IXmlAttributeRepository` é apropriado ao usar um armazenamento de backup diferente (por exemplo, armazenamento de tabela do Azure).

Para alterar o repositório padrão todo o aplicativo, registre um personalizado `IXmlAttributeRepository` instância:

```
services.Configure<KeyManagementOptions>(options => options.XmlRepository = new MyCustomXmlRepository());
```

```
services.AddSingleton<IXmlAttributeRepository>(new MyCustomXmlRepository());
```

## IXmlAttributeEncryptor

O `IXmlAttributeEncryptor` interface representa um tipo que pode criptografar um elemento XML de texto sem formatação. Ela apresenta uma única API:

- `Encrypt(XElement plaintextElement): EncryptedXmlInfo`

Se um serializado `IAuthenticatedEncryptorDescriptor` contém quaisquer elementos marcados como "requer criptografia", em seguida, `XmlKeyManager` executará esses elementos por meio do configurado `IXmlAttributeEncryptor` do

`Encrypt` método e ele serão mantido o elemento adulterem em vez de elemento de texto sem formatação para o `IXmlRepository`. A saída a `Encrypt` método é um `EncryptedXmlInfo` objeto. Esse objeto é um wrapper que contém os dois o resultante adulterem `XElement` e o tipo que representa um `IXmlDecryptor` que pode ser usado para decifrar o elemento correspondente.

Há quatro tipos concretos internos que implementam `IXmlEncryptor`:

- `CertificateXmlEncryptor`
- `DpapiNGXmlEncryptor`
- `DpapiXmlEncryptor`
- `NullXmlEncryptor`

Consulte a [criptografia de chave em documentos do rest](#) para obter mais informações.

Para alterar o mecanismo padrão de chave criptografia em repouso todo o aplicativo, registre um personalizado `IXmlEncryptor` instância:

```
services.Configure<KeyManagementOptions>(options => options.XmlEncryptor = new MyCustomXmlEncryptor());
```

```
services.AddSingleton<IXmlEncryptor>(new MyCustomXmlEncryptor());
```

## IXmlDecryptor

O `IXmlDecryptor` interface representa um tipo que sabe como descriptografar um `XElement` que foi adulterem por meio de um `IXmlEncryptor`. Ela apresenta uma única API:

- Descriptografe (encryptedElement XElement): XElement

O `Decrypt` método desfaz a criptografia executada pelo `IXmlEncryptor.Encrypt`. Em geral, cada concreto `IXmlEncryptor` implementação terá um concreto correspondente `IXmlDecryptor` implementação.

Tipos que implementam `IXmlDecryptor` deve ter um dos dois construtores públicos a seguir:

- `.ctor(IServiceProvider)`
- `.ctor()`

### NOTE

O `IServiceProvider` passado para o construtor pode ser nulo.

## IKeyEscrowSink

O `IKeyEscrowSink` interface representa um tipo que pode executar a caução de informações confidenciais.

Lembre-se de que descritores serializados podem conter informações confidenciais (por exemplo, o material criptográfico), e esse é o que levou à introdução do `IXmlEncryptor` digite em primeiro lugar. No entanto, acidentes acontecem e anéis de chave podem ser excluídos ou corrompidos.

A interface de caução fornece uma hachura de escape de emergência, permitindo o acesso ao XML bruto serializado antes que ele é transformado por qualquer configurados `IXmlEncryptor`. A interface expõe uma única API:

- Store (keyId Guid, o elemento de XElement)

Cabe ao `IKeyEscrowSink` implementação para lidar com o elemento fornecido de maneira segura consistente com

a política de negócios. Uma possível implementação pode ser para o coletor de caução criptografar o elemento XML usando um certificado x.509 corporativo reconhecido onde chave privada do certificado tenha sido mantida em garantia; o `ICertificateXmlEncryptor` tipo pode ajudar com isso. O `IKeyEscrowSink` implementação também é responsável por manter o elemento fornecido adequadamente.

Por padrão nenhum mecanismo de caução está habilitado, embora os administradores de servidor podem [configurar isso globalmente](#). Ele também pode ser configurado por meio de programação por meio de `IDataProtectionBuilder.AddKeyEscrowSink` método conforme mostrado no exemplo a seguir. O `AddKeyEscrowSink` espelho de sobrecargas do método de `IServiceCollection.AddSingleton` e `IServiceCollection.AddInstance` sobrecargas, como `IKeyEscrowSink` instâncias devem ser singlettons. Se vários `IKeyEscrowSink` instâncias registradas, cada um deles será chamado durante a geração de chave, para que as chaves podem ser mantida em garantia para diversos mecanismos simultaneamente.

Há uma API para ler o material de um `IKeyEscrowSink` instância. Isso é consistente com a teoria de design do mecanismo de caução: ele deve disponibilizar o material da chave para uma autoridade confiável, e uma vez que o aplicativo em si não é uma autoridade confiável, ele não deve ter acesso ao seu próprio caucionada material.

O código de exemplo a seguir demonstra como criar e registrar um `IKeyEscrowSink` onde as chaves são mantida em garantia, de modo que somente os membros do grupo "administradores de CONTOSODomain" poderá recuperá-los.

#### NOTE

Para executar este exemplo, você deve estar em um domínio do Windows 8 / máquina Windows Server 2012 e o controlador de domínio devem ser Windows Server 2012 ou posterior.

```
using System;
using System.IO;
using System.Xml.Linq;
using Microsoft.AspNetCore.DataProtection;
using Microsoft.AspNetCore.DataProtection.KeyManagement;
using Microsoft.AspNetCore.DataProtection.XmlEncryption;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;

public class Program
{
    public static void Main(string[] args)
    {
        var serviceCollection = new ServiceCollection();
        serviceCollection.AddDataProtection()
            .PersistKeysToFileSystem(new DirectoryInfo(@"c:\temp-keys"))
            .ProtectKeysWithDpapi()
            .AddKeyEscrowSink(sp => new MyKeyEscrowSink(sp));
        var services = serviceCollection.BuildServiceProvider();

        // get a reference to the key manager and force a new key to be generated
        Console.WriteLine("Generating new key...");
        var keyManager = services.GetService<IKeyManager>();
        keyManager.CreateNewKey(
            activationDate: DateTimeOffset.Now,
            expirationDate: DateTimeOffset.Now.AddDays(7));
    }

    // A key escrow sink where keys are escrowed such that they
    // can be read by members of the CONTOSO\Domain Admins group.
    private class MyKeyEscrowSink : IKeyEscrowSink
    {
        private readonly IXmlEncryptor _escrowEncryptor;

        public MyKeyEscrowSink(IServiceProvider services)
```

```

    {
        // Assuming I'm on a machine that's a member of the CONTOSO
        // domain, I can use the Domain Admins SID to generate an
        // encrypted payload that only they can read. Sample SID from
        // https://technet.microsoft.com/library/cc778824(v=ws.10).aspx.
        _escrowEncryptor = new DpapiNGXmlEncryptor(
            "SID=S-1-5-21-1004336348-1177238915-682003330-512",
            DpapiNGProtectionDescriptorFlags.None,
            new LoggerFactory());
    }

    public void Store(Guid keyId, XElement element)
    {
        // Encrypt the key element to the escrow encryptor.
        var encryptedXmlInfo = _escrowEncryptor.Encrypt(element);

        // A real implementation would save the escrowed key to a
        // write-only file share or some other stable storage, but
        // in this sample we'll just write it out to the console.
        Console.WriteLine($"Escrowing key {keyId}");
        Console.WriteLine(encryptedXmlInfo.EncryptedElement);

        // Note: We cannot read the escrowed key material ourselves.
        // We need to get a member of CONTOSO\Domain Admins to read
        // it for us in the event we need to recover it.
    }
}

/*
 * SAMPLE OUTPUT
 *
 * Generating new key...
 * Escrowing key 38e74534-c1b8-4b43-aea1-79e856a822e5
 * <encryptedKey>
 *   <!-- This key is encrypted with Windows DPAPI-NG. -->
 *   <!-- Rule: SID=S-1-5-21-1004336348-1177238915-682003330-512 -->
 *   <value>MIIIfAYJKoZIhvcNAQcDoIIbTCCCGkCAQ...T5rA4g==</value>
 * </encryptedKey>
 */

```

# APIs de proteção de dados de diversos principais do ASP.NET

22/06/2018 • 2 minutes to read • [Edit Online](#)

## WARNING

Tipos que implementam qualquer uma das seguintes interfaces devem ser thread-safe para chamadores vários.

## ISecret

O `ISecret` interface representa um valor de segredo, como o material de chave de criptografia. Ele contém a superfície de API a seguir:

- `Length` : `int`
- `Dispose()` : `void`
- `WriteSecretIntoBuffer(ArraySegment<byte> buffer)` : `void`

O `WriteSecretIntoBuffer` método preenche o buffer fornecido com o valor bruto de segredo. O motivo pelo qual essa API usa o buffer como um parâmetro em vez de retornar um `byte[]` diretamente é que isso permite que o chamador fixar o objeto de buffer, limitar a exposição de segredo para o coletor de lixo gerenciada.

O `Secret` tipo é uma implementação concreta de `ISecret` onde o valor de segredo é armazenado na memória no processo. Em plataformas Windows, o valor do segredo é criptografado por meio de [CryptProtectMemory](#).

# Implementação da proteção de dados do ASP.NET Core

21/06/2018 • 2 minutes to read • [Edit Online](#)

- Detalhes de criptografia autenticada
- Derivação de subchaves e criptografia autenticada
- Cabeçalhos de contexto
- Gerenciamento de chaves
- Provedores de armazenamento de chaves
- Criptografia de chave em repouso
- Imutabilidade de chave e configurações
- Formato do armazenamento de chaves
- Provedores de proteção de dados efêmeros

# Detalhes de criptografia autenticada no núcleo do ASP.NET

22/06/2018 • 4 minutes to read • [Edit Online](#)

Chamadas para `IDataProtector.Protect` são as operações de criptografia autenticada. O método `Protect` oferece confidencialidade e a autenticidade e ela é vinculada à cadeia finalidade que foi usada para essa instância específica do `IDataProtector` derivam da sua raiz `IDataProtectionProvider`.

`IDataProtector.Protect` usa um parâmetro de texto sem formatação do byte [] e produz uma byte [] protegido carga, cujo formato é descrito abaixo. (Também há uma sobrecarga de método de extensão que usa um parâmetro de texto sem formatação da cadeia de caracteres e retorna uma carga protegido de cadeia de caracteres. Se essa API é usada ainda terá o formato de carga protegido o abaixo de estrutura, mas será [codificado base64url](#).)

## Formato de conteúdo protegido

O formato de carga protegido consiste em três componentes principais:

- Um cabeçalho mágico de 32 bits que identifica a versão do sistema de proteção de dados.
- Uma id de 128 bits chave que identifica a chave usada para proteger essa carga específica.
- O restante da carga protegido é [específico para o Criptografador encapsulado por esta chave](#). No exemplo a seguir a chave representa um AES-256-CBC + Criptografador HMACSHA256 e a carga é mais subdividida da seguinte maneira: \* modificador chave A 128 bits. \* Um vetor de inicialização de 128 bits. \* 48 bytes de saída de AES-256-CBC. \* Uma marca de autenticação HMACSHA256.

Uma carga protegido exemplo ilustrada abaixo.

```
09 F0 C9 F0 80 9C 81 0C 19 66 19 40 95 36 53 F8
AA FF EE 57 57 2F 40 4C 3F 7F CC 9D CC D9 32 3E
84 17 99 16 EC BA 1F 4A A1 18 45 1F 2D 13 7A 28
79 6B 86 9C F8 B7 84 F9 26 31 FC B1 86 0A F1 56
61 CF 14 58 D3 51 6F CF 36 50 85 82 08 2D 3F 73
5F B0 AD 9E 1A B2 AE 13 57 90 C8 F5 7C 95 4E 6A
8A AA 06 EF 43 CA 19 62 84 7C 11 B2 C8 71 9D AA
52 19 2E 5B 4C 1E 54 F0 55 BE 88 92 12 C1 4B 5E
52 C9 74 A0
```

O formato de carga acima os primeiros 32 bits ou 4 bytes são o cabeçalho magic identifica a versão (09 F0 C9 F0)

O próximos 128 bits ou 16 bytes é o identificador de chave (80 9 81 de C 0C 19 66 19 40 95 36 53 F8 AA FF EE 57)

O restante contém a carga e é específico para o formato usado.

**WARNING**

Todas as cargas protegidas para uma determinada chave começa com o mesmo cabeçalho de 20 bytes (valor mágico, id de chave). Os administradores podem usar esse fato para fins de diagnóstico para aproximar quando uma carga foi gerada. Por exemplo, a carga acima corresponde à chave {0c819c80-6619-4019-9536-53f8aaffee57}. Se depois de verificar se o repositório de chave achar que a data de ativação dessa chave específica era 2015-01-01 e data de expiração foi 2015-03-01, é razoável pressupor que a carga (se não violada) foi gerada dentro dessa janela, dê ou levar um pequeno fator em ambos os lados.

# Derivação subchave e criptografia autenticada no núcleo do ASP.NET

22/06/2018 • 8 minutes to read • [Edit Online](#)

A maioria das chaves do anel de chave contém alguma forma de entropia e terá informações algorítmicas informando "criptografia de modo CBC + validação HMAC" ou "criptografia GCM + validação". Nesses casos, nos referimos a entropia inserida como o material de chave mestre (ou KM) para essa chave e podemos executar uma função de derivação de chave para derivar as chaves que serão usadas para operações de criptografia reais.

## NOTE

As chaves são abstratas e uma implementação personalizada pode não se comportar como abaixo. Se a chave fornece sua própria implementação do `IAuthenticatedEncryptor` em vez de usar uma das nossas fábricas internas, o mecanismo descrito nesta seção não se aplica.

## Dados autenticados adicionais e subchave derivação

O `IAuthenticatedEncryptor` interface serve como a interface principal para todas as operações de criptografia autenticada. Seu `Encrypt` método usa dois buffers: texto sem formatação e `additionalAuthenticatedData` (AAD). O fluxo de conteúdo de texto sem formatação inalterado a chamada para `IDataProtector.Protect`, mas o AAD é gerado pelo sistema e consiste em três componentes:

1. 32 bits magic cabeçalho 09 F0 C9 F0 que identifica esta versão do sistema de proteção de dados.
2. A id de chave de 128 bits.
3. Uma cadeia de caracteres de comprimento variável formada da cadeia de finalidade que criou o `IDataProtector` que está executando esta operação.

Como o AAD é exclusivo para a tupla de todos os três componentes, podemos usá-lo derivar novas chaves KM em vez de usar KM em si em todos os nossos operações criptográficas. Para todas as chamadas para `IAuthenticatedEncryptor.Encrypt`, ocorre o processo de derivação de chaves a seguir:

$(K_E, K_H) = SP800\_108\_CTR\_HMACSHA512(\text{contextHeader } K_M, \text{AAD}, || \text{keyModifier})$

Aqui, estamos ligando para o NIST SP800-108 KDF no modo de contador (consulte [NIST SP800-108](#), SEC 5.1) com os seguintes parâmetros:

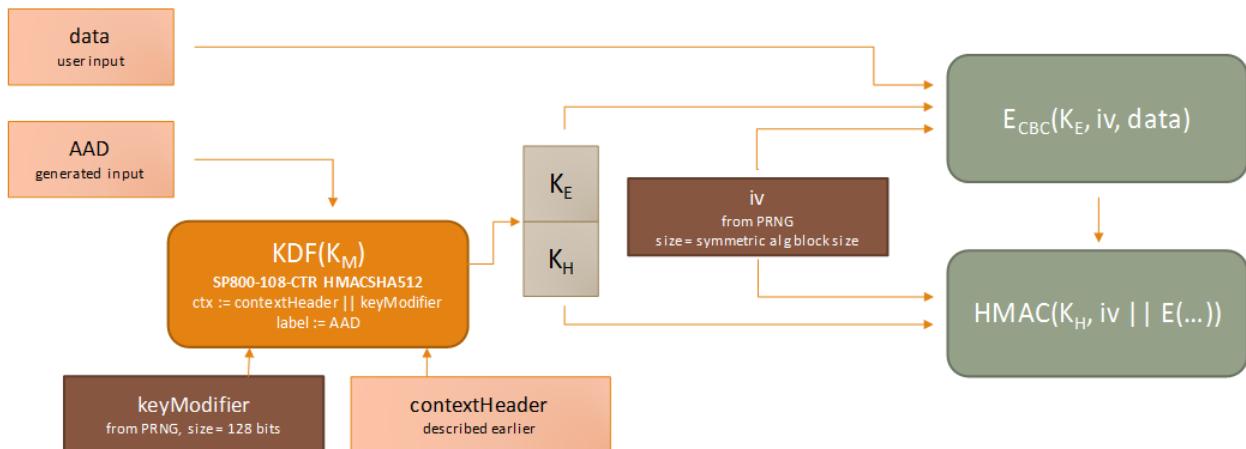
- Chave de derivação de chave (KDK) =  $K_M$
- PRF = HMACSHA512
- rótulo = `additionalAuthenticatedData`
- contexto = `contextHeader || keyModifier`

O cabeçalho de contexto é de comprimento variável e essencialmente serve como uma impressão digital dos algoritmos para a qual estamos estiver derivando  $K_E$  e  $K_H$ . O modificador de chave é uma cadeia de caracteres de 128 bits gerada aleatoriamente para cada chamada `Encrypt` e serve para garantir com grandes probabilidade que KE e KH são exclusivas para essa operação de criptografia de autenticação específico, mesmo se todas as outras entradas para o KDF é constante.

Para criptografia de modo CBC + operações de validação de HMAC,  $| K_E |$  é o comprimento da chave de criptografia simétrica bloco, e  $| K_H |$  é o tamanho do resumo da rotina de HMAC. Para criptografia GCM + operações de validação,  $| K_H | = 0$ .

## Criptografia de modo CBC + validação HMAC

Depois de  $K_E$  é gerado por meio do mecanismo acima, podemos gerar um vetor de inicialização aleatória e executar o algoritmo de criptografia simétrica de bloco para codificar o texto não criptografado. O vetor de inicialização e o texto cifrado são, em seguida, executar a rotina HMAC inicializada com a chave  $K_H$  para produzir Mac. Esse processo e o valor de retorno é representado graficamente abaixo.



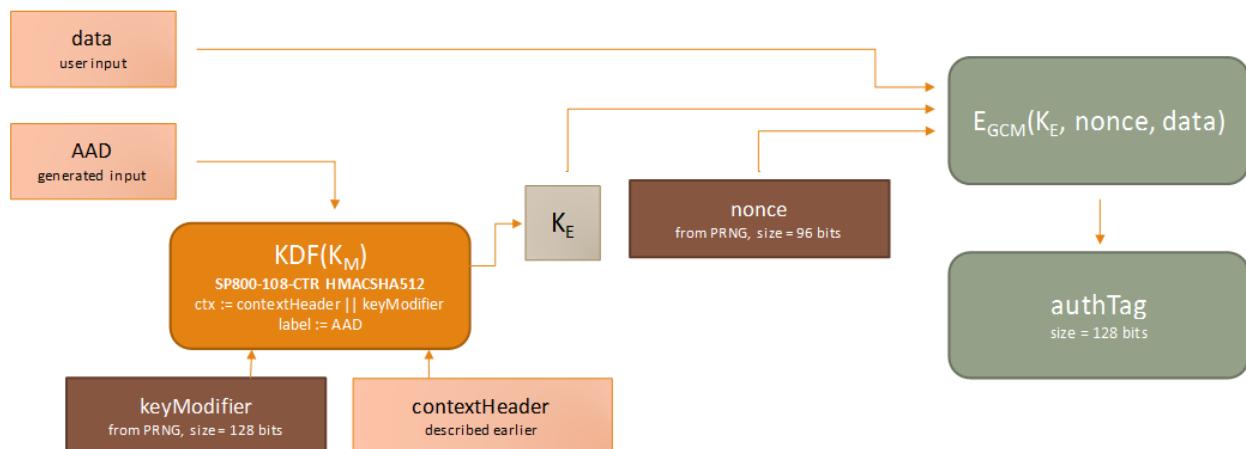
*saída: = keyModifier || IV || E\_cbc (dados K\_E, iv,) || HMAC (K\_H, iv || E\_cbc (dados K\_E, iv,))*

### NOTE

O `IDataProtector.Protect` implementação será precedida o cabeçalho **magic** e a **id de chave** a saída antes de retorná-lo ao chamador. Porque o cabeçalho **magic** e a **id de chave** são implicitamente parte do **AAD**, e porque o modificador de chave é passado como entrada para o KDF, isso significa que cada byte único da carga final retornada é autenticado pelo Mac.

## Criptografia de modo Galois/contador + validação

Quando  $K_E$  é gerado por meio do mecanismo acima, podemos gerar um nonce de 96 bits aleatório e executar o algoritmo de criptografia simétrica de bloco para codificar o texto não criptografado e produzir a marca de autenticação de 128 bits.



*saída: = keyModifier || nonce || E\_gcm (dados de uso único, K\_E,) || authTag*

#### **NOTE**

Embora GCM nativamente dê suporte ao conceito de AAD, podemos estiver ainda de alimentação AAD somente para o KDF original, aceitar para passar uma cadeia de caracteres vazia para GCM para seu parâmetro AAD. A razão para isso é dupla. Primeiro, [para dar suporte a agilidade](#) nunca queremos usar K\_M diretamente como a chave de criptografia. Além disso, GCM impõe requisitos de exclusividade muito estrita em suas entradas. Define a probabilidade de que a rotina de criptografia do GCM é sempre invocado em dois ou mais distintos de dados de entrada com o mesmo (chave, nonce) par não deve exceder  $2^{-32}$ . Se corrigir K\_E não foi possível realizar mais de  $2^{-32}$  operações de criptografia antes de podemos executar afoul da  $2^{-32}$  limitar. Isso pode parecer um grande número de operações, mas um servidor web de alto tráfego pode passar por solicitações de 4 bilhões em alguns dias, dentro do tempo de vida normal para essas chaves. Para manter a conformidade de  $2^{-32}$  limite de probabilidade-32, podemos continuar a usar um modificador de chave de 128 bits e nonce de 96 bits, o que aumenta a contagem de operação utilizável para qualquer K\_M determinado. Para manter a simplicidade de design compartilha o caminho do código KDF entre as operações CBC e GCM e como AAD já é considerado o KDF não é necessário encaminhá-lo para a rotina do GCM.

# Cabeçalhos de contexto no núcleo do ASP.NET

22/06/2018 • 16 minutes to read • [Edit Online](#)

## Plano de fundo e a teoria

No sistema de proteção de dados, uma "chave" significa que um objeto que pode fornecer serviços de criptografia de autenticação. Cada chave é identificado por uma id exclusiva (uma GUID) e transporta informações algorítmicos e material entropic. Ele destina-se que cada chave realizar entropia exclusiva, mas o sistema não pode impor que precisamos para desenvolvedores que podem alterar manualmente o anel de chave, modificando as informações de algoritmos de uma chave existente do anel de chave de conta. Para alcançar nossos requisitos de segurança fornecidos nesses casos, o sistema de proteção de dados tem um conceito de **agilidade criptográfica**, que permite a com segurança usando um único valor entropic entre vários algoritmos de criptografia.

A maioria dos sistemas que oferecem suporte a agilidade criptográfica de fazer isso, incluindo algumas informações de identificação sobre o algoritmo de carga. OID do algoritmo geralmente é uma boa candidata para isso. No entanto, um problema que tivemos é que há várias maneiras de especificar o mesmo algoritmo: "AES" (CNG) e o gerenciado Aes, AesManaged, AesCryptoServiceProvider, AesCng e RijndaelManaged (fornecida parâmetros específicos) classes são, na verdade, todos os mesmos coisa e é necessário manter um mapeamento de tudo isso para a identificação de objeto correta. Se um desenvolvedor deseja fornecer um algoritmo personalizado (ou até mesmo outra implementação do AES!), eles precisam Conte-nos sua OID. Essa etapa de registro extra facilita a configuração do sistema particularmente penoso.

Recuando, decidimos que podemos forçando o problema de direção errada. Um OID informa o que é o algoritmo, mas, na verdade, não importa sobre isso. Se for necessário usar um único valor entropic com segurança em dois algoritmos diferentes, não é necessário para que possamos saber o que os algoritmos realmente são. O que é realmente importante é como eles se comportam. Qualquer algoritmo de codificação de bloco simétrica razoável também é uma forte permutação pseudoaleatórios (PRP): corrija as entradas (chave, cadeia de texto sem formatação de modo, o IV,) e a saída de texto cifrado com grandes probabilidade serão diferente dos outros qualquer codificação de bloco simétrica algoritmo considerando as entradas do mesmo. Da mesma forma, qualquer função de hash com chave razoável também é uma função pseudoaleatórios forte (PRF) e fornecido um conjunto fixo de entrada sua saída predominantemente serão diferente de qualquer outra função de hash com chave.

Podemos usar esse conceito de alta segurança PRPs e PRFs para criar um cabeçalho de contexto. Esse cabeçalho de contexto atua essencialmente como uma impressão digital estável sobre os algoritmos em uso para qualquer operação especificada e fornece a agilidade criptográfica necessária ao sistema de proteção de dados. Esse cabeçalho pode ser reproduzido e é usado posteriormente como parte do **processo de derivação de subchave**. Há duas maneiras diferentes para criar o cabeçalho de contexto dependendo os modos de operação dos algoritmos subjacentes.

## Criptografia de modo CBC + autenticação HMAC

O cabeçalho de contexto consiste dos seguintes componentes:

- [16 bits] O valor 00 00, o que é um marcador que significa "criptografia CBC + autenticação HMAC".
- [32 bits] O comprimento da chave (em bytes, big-endian) do algoritmo de criptografia simétrica de bloco.
- [32 bits] O tamanho de bloco (em bytes, big-endian) do algoritmo de criptografia simétrica de bloco.
- [32 bits] O comprimento da chave (em bytes, big-endian) do algoritmo HMAC. (Atualmente o tamanho da chave sempre coincide com o tamanho do resumo.)

- [32 bits] O tamanho (em bytes, big-endian) resumo do algoritmo HMAC.
- EncCBC ( $K_E$ , IV, ""), que é a saída do algoritmo de criptografia simétrica bloco recebe uma entrada de cadeia de caracteres vazia e onde o IV é um vetor de zeros. A construção de  $K_E$  é descrita abaixo.
- MAC ( $K_H$ , ""), que é a saída do algoritmo HMAC recebe uma entrada de cadeia de caracteres vazia. A construção de  $K_H$  é descrita abaixo.

Idealmente, poderíamos passar vetores de zeros para  $K_E$  e  $K_H$ . No entanto, queremos evitar a situação em que o algoritmo subjacente verifica a existência de baixa segurança chaves antes de executar quaisquer operações (especialmente DES e 3DES), que impede usando um padrão simple ou repeatable como um vetor de zeros.

Em vez disso, usamos o NIST SP800-108 KDF no modo de contador (consulte [NIST SP800-108](#), SEC 5.1) com uma chave de comprimento zero, o rótulo e o contexto e a HMACSHA512 como o PRF subjacente. Podemos derivar  $|K_E| + |K_H|$  bytes de saída, em seguida, decompor o resultado em  $K_E$  e  $K_H$  próprios.

Matematicamente, isso é representado como a seguir.

$$(K_E \parallel K_H) = \text{SP800\_108\_CTR}(\text{prf} = \text{HMACSHA512}, \text{key} = "", \text{label} = "", \text{context} = "")$$

### **Exemplo: AES de 192 de CBC + HMACSHA256**

Por exemplo, considere o caso em que o algoritmo de criptografia simétrica de bloco é AES de 192 de CBC e o algoritmo de validação é HMACSHA256. O sistema poderia gerar o cabeçalho de contexto usando as etapas a seguir.

Primeiro, permitem  $(K_E \parallel K_H) = \text{SP800\_108\_CTR}(\text{prf} = \text{HMACSHA512}, \text{chave} = "", \text{rótulo} = "", \text{contexto} = "")$ , onde  $|K_E| = 192$  bits e  $|K_H| = 256$  bits por algoritmos especificados. Isso leva a  $K_E = 5BB6.21DD$  e  $K_H = A04A.00A9$  no exemplo a seguir:

```
5B B6 C9 83 13 78 22 1D 8E 10 73 CA CF 65 8E B0
61 62 42 71 CB 83 21 DD A0 4A 05 00 5B AB C0 A2
49 6F A5 61 E3 E2 49 87 AA 63 55 CD 74 0A DA C4
B7 92 3D BF 59 90 00 A9
```

Em seguida, calcular Enc\_CBC ( $K_E$ , IV, "") para AES-192-CBC fornecido IV = 0 \* e  $K_E$  como acima.

result := F474B1872B3B53E4721DE19C0841DB6F

Em seguida, o MAC de computação ( $K_H$ , "") para HMACSHA256 determinado  $K_H$  como acima.

result := D4791184B996092EE1202F36E8608FA8FBD98ABDFF5402F264B1D7211536220C

Isso gera o cabeçalho de contexto completo abaixo:

```
00 00 00 00 00 18 00 00 00 10 00 00 00 20 00 00
00 20 F4 74 B1 87 2B 3B 53 E4 72 1D E1 9C 08 41
DB 6F D4 79 11 84 B9 96 09 2E E1 20 2F 36 E8 60
8F A8 FB D9 8A BD FF 54 02 F2 64 B1 D7 21 15 36
22 0C
```

Esse cabeçalho de contexto é a impressão digital do par de algoritmo de criptografia autenticada (criptografia AES de 192 de CBC + HMACSHA256 validação). Os componentes, conforme descrito [acima](#) são:

- o marcador (00 00)
- o comprimento da chave de codificação de bloco (00 00 00 18)
- o tamanho de bloco de codificação de bloco (00 00 00 10)
- o comprimento da chave HMAC (00 00 00 20)

- o tamanho do resumo HMAC (00 00 00 20)
- a codificação de bloco saída PRP (F4 74 - DB 6F) e
- a saída PRF HMAC (D4 79 - end).

**NOTE**

A criptografia de modo CBC + HMAC cabeçalho de contexto de autenticação baseia-se da mesma maneira, independentemente de se as implementações de algoritmos são fornecidas pelo Windows CNG ou por tipos SymmetricAlgorithm e KeyedHashAlgorithm gerenciados. Isso permite que os aplicativos executados em diferentes sistemas operacionais confiável produzem o mesmo cabeçalho de contexto, embora as implementações de algoritmos diferem entre sistemas operacionais. (Na prática, o KeyedHashAlgorithm não precisa ser um HMAC adequado. Ele pode ser qualquer tipo de algoritmo de hash com chave.)

**Exemplo: 3DES-192-CBC + HMACSHA1**

Primeiro, permitem ( $K_E \parallel K_H$ ) = SP800\_108\_CTR (prf = HMACSHA512, chave = "", rótulo = "", contexto = ""), onde  $|K_E| = 192$  bits e  $|K_H| = 160$  bits por algoritmos especificados. Isso leva a  $K_E = A219.E2BB$  e  $K_H = DC4A.B464$  no exemplo a seguir:

```
A2 19 60 2F 83 A9 13 EA B0 61 3A 39 B8 A6 7E 22
61 D9 F8 6C 10 51 E2 BB DC 4A 00 D7 03 A2 48 3E
D1 F7 5A 34 EB 28 3E D7 D4 67 B4 64
```

Em seguida, calcular Enc\_CBC ( $K_E$ , IV, "") para 3DES-192-CBC fornecido IV = 0 \* e  $K_E$  como acima.

resultado: = ABB100F81E53E10E

Em seguida, o MAC de computação ( $K_H$ , "") para HMACSHA1 determinado  $K_H$  como acima.

result := 76EB189B35CF03461DDF877CD9F4B1B4D63A7555

Isso gera o cabeçalho de contexto completo que é uma impressão digital do autenticado par de algoritmo do encryption (criptografia 3DES-192-CBC + HMACSHA1 validação), mostrado abaixo:

```
00 00 00 00 00 18 00 00 00 08 00 00 00 14 00 00
00 14 AB B1 00 F8 1E 53 E1 0E 76 EB 18 9B 35 CF
03 46 1D DF 87 7C D9 F4 B1 B4 D6 3A 75 55
```

Os componentes são divididos da seguinte maneira:

- o marcador (00 00)
- o comprimento da chave de codificação de bloco (00 00 00 18)
- o tamanho de bloco de codificação de bloco (00 00 00 08)
- o comprimento da chave HMAC (00 00 00 14)
- o tamanho do resumo HMAC (00 00 00 14)
- a codificação de bloco saída PRP (B1 AB - E1 0E) e
- a saída PRF HMAC (76 EB - end).

## Criptografia de modo Galois/contador + autenticação

O cabeçalho de contexto consiste dos seguintes componentes:

- [16 bits] O valor 00 01, que é um marcador que significa "criptografia GCM + autenticação".
- [32 bits] O comprimento da chave (em bytes, big-endian) do algoritmo de criptografia simétrica de bloco.
- [32 bits] O tamanho nonce (em bytes, big-endian) usado durante as operações de criptografia autenticada. (Para nosso sistema, isso é fixo em tamanho nonce = 96 bits.)
- [32 bits] O tamanho de bloco (em bytes, big-endian) do algoritmo de criptografia simétrica de bloco. (Para GCM, isso é fixo em tamanho de bloco = 128 bits.)
- [32 bits] O autenticação marca tamanho (em bytes, big-endian) produzido pela função criptografia autenticada. (Para nosso sistema, isso é fixo em tamanho de marca = 128 bits.)
- [128 bits] A marca de Enc\_GCM ( $K_E$ , nonce, ""), que é a saída do algoritmo de criptografia simétrica bloco recebe uma entrada de cadeia de caracteres vazia e onde nonce é um vetor de zeros de 96 bits.

$K_E$  é obtida usando o mesmo mecanismo como a criptografia CBC + o cenário de autenticação de HMAC. No entanto, já que não há nenhum  $K_H$  em jogo aqui, essencialmente temos  $|K_H| = 0$ , e o algoritmo recolhe para o formulário abaixo.

$K_E = SP800\_108\_CTR$  (prf = HMACSHA512, chave = "", rótulo = "", contexto = "")

#### **Exemplo: AES-256-GCM**

Primeiramente, vamos  $K_E = SP800\_108\_CTR$  (prf = HMACSHA512, chave = "", rótulo = "", contexto = ""), onde  $|K_E| = 256$  bits.

$K_E := 22BC6F1B171C08C4AE2F27444AF8FC8B3087A90006CAEA91FDCFB47C1B8733B8$

Em seguida, calcular a marca de autenticação de Enc\_GCM ( $K_E$ , nonce, "") para AES-256-GCM fornecido nonce = 096 e  $K_E$  como acima.

result := E7DCCE66DF855A323A6BB7BD7A59BE45

Isso gera o cabeçalho de contexto completo abaixo:

```
00 01 00 00 00 20 00 00 00 0C 00 00 00 10 00 00
00 10 E7 DC CE 66 DF 85 5A 32 3A 6B B7 BD 7A 59
BE 45
```

Os componentes são divididos da seguinte maneira:

- o marcador (00 01)
- o comprimento da chave de codificação de bloco (00 00 00 20)
- o tamanho de nonce (00 00 00 0 C)
- o tamanho de bloco de codificação de bloco (00 00 00 10)
- o tamanho de marca de autenticação (00 00 00 10) e
- a marca de autenticação da execução de codificação de bloco (controlador de domínio E7 - end).

# Gerenciamento de chaves no ASP.NET Core

24/07/2018 • 12 minutes to read • [Edit Online](#)

O sistema de proteção de dados gerencia automaticamente o tempo de vida das chaves mestras usado para proteger e Desproteger cargas. Cada chave pode existir em um dos quatro estágios:

- Criada - a chave existe no anel de chave, mas ainda não foi ativada. A chave não deve ser usada para novas operações de proteção até que haja tempo suficiente tenha decorrido a chave tem tido a oportunidade de se propagar para todos os computadores que estão consumindo esse anel de chave.
- Ativo - a chave existe no anel de chave e deve ser usado para todas as operações de proteção de novo.
- Expirou - a chave executou seu tempo de vida natural e não deve mais ser usada para novas operações de proteção.
- Revogado - a chave for comprometida e não deve ser usada para novas operações de proteção.

Todos criadas, ativas e vencidas chaves podem ser usadas para desproteger cargas de entrada. Chaves revogadas por padrão não podem ser usadas para desproteger cargas, mas o desenvolvedor de aplicativos pode [substituir esse comportamento](#) se necessário.

## WARNING

O desenvolvedor pode ser tentado a excluir uma chave do anel de chave (por exemplo, excluindo o arquivo correspondente do sistema de arquivos). Nesse ponto, todos os dados protegidos pela chave é indecifráveis permanentemente e não há nenhuma substituição de emergência como ocorre com chaves revogadas. Excluir uma chave é verdadeiramente destrutivo comportamento e, consequentemente, o sistema de proteção de dados não expõe nenhuma API de primeira classe para realizar esta operação.

## Seleção de chave padrão

Quando o sistema de proteção de dados lê o anel de chave do repositório de backup, ele tenta localizar uma chave de "padrão" do anel de chave. A chave padrão é usada para novas operações de proteção.

A heurística geral é que o sistema de proteção de dados escolhe a chave com a data mais recente de ativação como a chave padrão. (Há um pequeno fator para permitir o relógio do servidor para servidor distorção.) Se a chave expirou ou foi revogada, e se o aplicativo não tiver desabilitado automática da chave de geração, uma nova chave será gerada com a ativação imediata pela [expiração e sem interrupção de chave](#) política abaixo.

O motivo pelo qual o sistema de proteção de dados gera uma nova chave imediatamente em vez de fazer fallback para uma chave diferente é que a nova geração de chave deve ser tratada como uma expiração implícita de todas as chaves que foram ativados antes da nova chave. A ideia geral é que novas chaves podem ter sido configuradas com algoritmos diferentes ou mecanismos de criptografia em repouso que chaves antigos, e o sistema deve preferir a configuração atual em vez de fazer fallback.

Há uma exceção. Se o desenvolvedor do aplicativo tiver [desabilitada a geração automática de chaves](#), em seguida, o sistema de proteção de dados deve escolher algo como a chave padrão. Nesse cenário de fallback, o sistema escolherá a chave não revogado com a data de ativação mais recente, com preferência para chaves que tem tido tempo para ser propagada para outros computadores no cluster. O sistema de fallback pode acabar de escolher uma chave expirada padrão como resultado. O sistema de fallback nunca escolherá uma chave revogada como a chave padrão e se o anel de chave estiver vazio ou todas as chaves foi revogada, em seguida, o sistema produzirá um erro na inicialização.

## Expiração da chave e sem interrupção

Quando uma chave é criada, ele automaticamente tenha dado uma data de ativação de {agora + 2 dias} e uma data de expiração de {agora + 90 dias}. O atraso de 2 dias antes da ativação fornece o tempo-chave para se propagar através do sistema. Ou seja, ele permite que outros aplicativos que aponta para o repositório de backup observar a chave em seu próximo período de atualização automática, além de aumentar as chances de que quando a chave de anel ativo faz tornam-se que ele ser propagado para todos os aplicativos que talvez precise usá-lo.

Se a chave padrão irá expirar dentro de 2 dias e o anel de chave ainda não tiver uma chave que estará ativa após a expiração da chave padrão, o sistema de proteção de dados persistirá automaticamente uma nova chave para o anel de chave. Essa nova chave tem uma data de ativação de {data de validade da chave padrão} e uma data de expiração de {agora + 90 dias}. Isso permite que o sistema automaticamente reverte chaves regularmente sem interrupção do serviço.

Pode haver circunstâncias em que uma chave será criada com a ativação imediata. Um exemplo seria quando o aplicativo não foi executada por um tempo e todas as chaves do anel de chave estão expiradas. Quando isso acontece, a chave é dada uma data de ativação de {agora} sem o atraso de ativação de 2 dias normal.

O tempo de vida de chave padrão é de 90 dias, embora isso seja configurável, como no exemplo a seguir.

```
services.AddDataProtection()
    // use 14-day lifetime instead of 90-day lifetime
    .SetDefaultKeyLifetime(TimeSpan.FromDays(14));
```

Um administrador também pode alterar o padrão para todo o sistema, embora uma chamada explícita para `SetDefaultKeyLifetime` substituirá qualquer política de todo o sistema. O tempo de vida de chave padrão não pode ser menor do que 7 dias.

## Atualização automática de anel de chave

Quando o sistema de proteção de dados é inicializado, ele lê o anel de chave do repositório subjacente e armazena em cache na memória. Esse cache permite proteger e Desproteger operações continuar sem atingir o repositório de backup. O sistema verificará o repositório de backup para que as alterações automaticamente aproximadamente a cada 24 horas ou quando a chave padrão atual expirar, o que vier primeiro.

### WARNING

Os desenvolvedores devem muito raramente (se utilizadas) precisará usar as APIs de gerenciamento de chave diretamente. O sistema de proteção de dados executará gerenciamento automático de chaves, conforme descrito acima.

O sistema de proteção de dados expõe uma interface `IKeyManager` que pode ser usado para inspecionar e fazer alterações para o anel de chave. O sistema de injeção de dependência que forneceu a instância do `IDataProtectionProvider` também pode fornecer uma instância de `IKeyManager` para seu consumo. Como alternativa, você pode extrair os `IKeyManager` diretamente do `IServiceProvider` como no exemplo a seguir.

Qualquer operação que modifica o anel de chave (Criando uma nova chave explicitamente ou executar uma revogação) invalida o cache na memória. A próxima chamada para `Protect` ou `Unprotect` fará com que o sistema de proteção de dados releia o anel de chave e recriar o cache.

O exemplo a seguir demonstra como usar o `IKeyManager` interface para inspecionar e manipular o anel de chave, incluindo Revogando existente de chaves e gerar uma nova chave manualmente.

```
using System;
using System.IO;
```

```

-----,
using System.Threading;
using Microsoft.AspNetCore.DataProtection;
using Microsoft.AspNetCore.DataProtection.KeyManagement;
using Microsoft.Extensions.DependencyInjection;

public class Program
{
    public static void Main(string[] args)
    {
        var serviceCollection = new ServiceCollection();
        serviceCollection.AddDataProtection()
            // point at a specific folder and use DPAPI to encrypt keys
            .PersistKeysToFileSystem(new DirectoryInfo(@"c:\temp-keys"))
            .ProtectKeysWithDpapi();
        var services = serviceCollection.BuildServiceProvider();

        // perform a protect operation to force the system to put at least
        // one key in the key ring
        services.GetDataProtector("Sample.KeyManager.v1").Protect("payload");
        Console.WriteLine("Performed a protect operation.");
        Thread.Sleep(2000);

        // get a reference to the key manager
        var keyManager = services.GetService<IKeyManager>();

        // list all keys in the key ring
        var allKeys = keyManager.GetAllKeys();
        Console.WriteLine($"The key ring contains {allKeys.Count} key(s.).");
        foreach (var key in allKeys)
        {
            Console.WriteLine($"Key {key.KeyId:B}: Created = {key.CreationDate:u}, IsRevoked = {key.IsRevoked}");
        }

        // revoke all keys in the key ring
        keyManager.RevokeAllKeys(DateTimeOffset.Now, reason: "Revocation reason here.");
        Console.WriteLine("Revoked all existing keys.");

        // add a new key to the key ring with immediate activation and a 1-month expiration
        keyManager.CreateNewKey(
            activationDate: DateTimeOffset.Now,
            expirationDate: DateTimeOffset.Now.AddMonths(1));
        Console.WriteLine("Added a new key.");

        // list all keys in the key ring
        allKeys = keyManager.GetAllKeys();
        Console.WriteLine($"The key ring contains {allKeys.Count} key(s.).");
        foreach (var key in allKeys)
        {
            Console.WriteLine($"Key {key.KeyId:B}: Created = {key.CreationDate:u}, IsRevoked = {key.IsRevoked}");
        }
    }
}

/*
 * SAMPLE OUTPUT
 *
 * Performed a protect operation.
 * The key ring contains 1 key(s.).
 * Key {1b948618-be1f-440b-b204-64ff5a152552}: Created = 2015-03-18 22:20:49Z, IsRevoked = False
 * Revoked all existing keys.
 * Added a new key.
 * The key ring contains 2 key(s.).
 * Key {1b948618-be1f-440b-b204-64ff5a152552}: Created = 2015-03-18 22:20:49Z, IsRevoked = True
 * Key {2266fc40-e2fb-48c6-8ce2-5fde6b1493f7}: Created = 2015-03-18 22:20:51Z, IsRevoked = False
*/

```

## Armazenamento de chaves

O sistema de proteção de dados tem uma heurística, no qual ele tenta deduzir um mecanismo de criptografia em repouso e o local de armazenamento de chave apropriado automaticamente. O mecanismo de persistência de chave também é configurável pelo desenvolvedor do aplicativo. Os documentos a seguir discutem as implementações de caixa de entrada desses mecanismos:

- [Provedores de armazenamento de chaves no ASP.NET Core](#)
- [Chave de criptografia em repouso no ASP.NET Core](#)

# Provedores de armazenamento de chaves no ASP.NET Core

27/12/2018 • 6 minutes to read • [Edit Online](#)

O sistema de proteção de dados [emprega um mecanismo de descoberta por padrão](#) para determinar onde as chaves de criptografia devem ser persistente. O desenvolvedor pode substituir o mecanismo de descoberta padrão e especificar manualmente o local.

## WARNING

Se você especificar um local de persistência de chave explícita, o sistema de proteção de dados cancela o registro a criptografia de chave padrão no mecanismo de rest, portanto, as chaves não são criptografadas em repouso. É recomendável que você adicionalmente [especificar um mecanismo de criptografia de chave explícita](#) para implantações de produção.

## Sistema de arquivos

Para configurar um repositório chave baseadas no sistema de arquivos, chame o [PersistKeysToFileSystem](#) rotina de configuração, conforme mostrado abaixo. Fornecer um  [DirectoryInfo](#) apontando para o repositório em que as chaves devem ser armazenadas:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDataProtection()
        .PersistKeysToFileSystem(new DirectoryInfo(@"c:\temp-keys\"));
}
```

O `DirectoryInfo` pode apontar para um diretório no computador local, ou ele pode apontar para uma pasta em um compartilhamento de rede. Se apontando para um diretório no computador local (e o cenário é que apenas os aplicativos no computador local exigem acesso para usar esse repositório), considere o uso de [Windows DPAPI](#) (no Windows) para criptografar as chaves em repouso. Caso contrário, considere o uso de um [certificado x.509](#) para criptografar as chaves em repouso.

## O Azure e Redis

O [Microsoft.AspNetCore.DataProtection.AzureStorage](#) e [Microsoft.AspNetCore.DataProtection.StackExchangeRedis](#) pacotes permitem armazenar chaves de proteção de dados no armazenamento do Azure ou de um Redis cache. As chaves podem ser compartilhadas entre várias instâncias de um aplicativo web. Aplicativos podem compartilhar cookies de autenticação ou proteção CSRF em vários servidores.

O [Microsoft.AspNetCore.DataProtection.AzureStorage](#) e [Microsoft.AspNetCore.DataProtection.Redis](#) pacotes permitem armazenar chaves de proteção de dados no armazenamento do Azure ou um cache Redis. As chaves podem ser compartilhadas entre várias instâncias de um aplicativo web. Aplicativos podem compartilhar cookies de autenticação ou proteção CSRF em vários servidores.

Para configurar o provedor de armazenamento de BLOBs do Azure, chame um dos [PersistKeysToAzureBlobStorage](#) sobrecargas:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDataProtection()
        .PersistKeysToAzureBlobStorage(new Uri("<blob URI including SAS token>"));
}
```

Para configurar o Redis, chame um dos [PersistKeysToStackExchangeRedis](#) sobrecargas:

```
public void ConfigureServices(IServiceCollection services)
{
    var redis = ConnectionMultiplexer.Connect("<URI>");
    services.AddDataProtection()
        .PersistKeysToStackExchangeRedis(redis, "DataProtection-Keys");
}
```

Para configurar o Redis, chame um dos [PersistKeysToRedis](#) sobrecargas:

```
public void ConfigureServices(IServiceCollection services)
{
    var redis = ConnectionMultiplexer.Connect("<URI>");
    services.AddDataProtection()
        .PersistKeysToRedis(redis, "DataProtection-Keys");
}
```

Para mais informações, consulte os seguintes tópicos:

- [ConnectionMultiplexer stackexchange.Redis](#)
- [Cache Redis do Azure](#)
- [exemplos de ASP.NET/DataProtection](#)

## Registro

**Só se aplica às implantações do Windows.**

Às vezes, o aplicativo pode não ter acesso de gravação para o sistema de arquivos. Considere um cenário em que um aplicativo é executado como uma conta de serviço virtual (como `w3wp.exe` da identidade do pool de aplicativo). Nesses casos, o administrador pode provisionar uma chave do registro que seja acessível pela identidade da conta de serviço. Chame o [PersistKeysToRegistry](#) método de extensão, conforme mostrado abaixo. Fornecer um [RegistryKey](#) apontando para o local onde as chaves de criptografia devem ser armazenadas:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDataProtection()
        .PersistKeysToRegistry(Registry.CurrentUser.OpenSubKey(@"SOFTWARE\Sample\keys"));
}
```

### IMPORTANT

É recomendável usar [Windows DPAPI](#) para criptografar as chaves em repouso.

## Entity Framework Core

O [Microsoft.AspNetCore.DataProtection.EntityFrameworkCore](#) pacote fornece um mecanismo para armazenar

chaves de proteção de dados para um banco de dados usando o Entity Framework Core. O `Microsoft.AspNetCore.DataProtection.EntityFrameworkCore` pacote do NuGet deve ser adicionado ao arquivo de projeto, ele não é parte do [metapacote Microsoft](#).

Com esse pacote, as chaves podem ser compartilhadas entre várias instâncias de um aplicativo web.

Para configurar o provedor do EF Core, chame o `PersistKeysToDbContext<TContext>` método:

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));

    // using Microsoft.AspNetCore.DataProtection;
    services.AddDataProtection()
        .PersistKeysToDbContext<MyKeysContext>();

    services.AddDefaultIdentity<IdentityUser>()
        .AddDefaultUI(UIFramework.Bootstrap4)
        .AddEntityFrameworkStores<ApplicationDbContext>();
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
}
```

O parâmetro genérico, `TContext`, deve herdar de `DbContext` e `IDataProtectionKeyContext`:

```
using Microsoft.AspNetCore.DataProtection.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using WebApp1.Data;

namespace WebApp1
{
    class MyKeysContext : DbContext, IDataProtectionKeyContext
    {
        // A recommended constructor overload when using EF Core
        // with dependency injection.
        public MyKeysContext(DbContextOptions<ApplicationDbContext> options)
            : base(options) { }

        // This maps to the table that stores keys.
        public DbSet<DataProtectionKey> DataProtectionKeys { get; set; }
    }
}
```

Criar o `DataProtectionKeys` tabela.

- [Visual Studio](#)
- [CLI do .NET Core](#)

Execute os seguintes comandos na **Package Manager Console** janela (PMC):

```
Add-Migration AddDataProtectionKeys -Context MyKeysContext
Update-Database -Context MyKeysContext
```

`MyKeysContext` é o `DbContext` definido no exemplo de código anterior. Se você estiver usando um `DbContext`

com um nome diferente, substitua seu `DbContext` nome do `MyKeysContext`.

O `DataProtectionKeys` classe/entidade adota a estrutura mostrada na tabela a seguir.

CAMPO/PROPRIEDADE	TIPO CLR	TIPO SQL
<code>Id</code>	<code>int</code>	<code>int</code> , PK, não nulo
<code>FriendlyName</code>	<code>string</code>	<code>nvarchar(MAX)</code> , nulo
<code>Xml</code>	<code>string</code>	<code>nvarchar(MAX)</code> , nulo

## Repositório de chave personalizado

Se os mecanismos de caixa de entrada não forem apropriados, o desenvolvedor pode especificar seu próprio mecanismo de persistência de chave, fornecendo um personalizado [IXmlRepository](#).

# Chave de criptografia em repouso no ASP.NET Core

24/07/2018 • 6 minutes to read • [Edit Online](#)

O sistema de proteção de dados [emprega um mecanismo de descoberta por padrão](#) para determinar as chaves de criptografia como devem ser criptografados em repouso. O desenvolvedor pode substituir o mecanismo de descoberta e especificar manualmente como chaves devem ser criptografadas em repouso.

## WARNING

Se você especificar um explícito [local de persistência da chave](#), o sistema de proteção de dados cancela o registro a criptografia de chave padrão no mecanismo de rest. Consequentemente, as chaves não são criptografadas em repouso. É recomendável que você [especificar um mecanismo de criptografia de chave explícita](#) para implantações de produção. As opções de mecanismo de criptografia em repouso são descritas neste tópico.

## Azure Key Vault

Para armazenar as chaves na [Azure Key Vault](#), configure o sistema com [ProtectKeysWithAzureKeyVault](#) no `Startup` classe:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDataProtection()
        .PersistKeysToAzureBlobStorage(new Uri("<blobUriWithSasToken>"))
        .ProtectKeysWithAzureKeyVault("<keyIdentifier>", "<clientId>", "<clientSecret>");
}
```

Para obter mais informações, consulte [configurar a proteção de dados do ASP.NET Core: ProtectKeysWithAzureKeyVault](#).

## Windows DPAPI

**Só se aplica às implantações do Windows.**

Quando o Windows DPAPI é usado, o material da chave é criptografada com [CryptProtectData](#) antes que estão sendo mantidos no armazenamento. A DPAPI é um mecanismo de criptografia apropriado para os dados que nunca é lida de fora do computador atual (no entanto é possível fazer essas chaves até do Active Directory; Consulte [DPAPI e perfis móveis](#)). Para configurar a criptografia de chave em repouso DPAPI, chame um dos [ProtectKeysWithDpapi](#) métodos de extensão:

```
public void ConfigureServices(IServiceCollection services)
{
    // Only the local user account can decrypt the keys
    services.AddDataProtection()
        .ProtectKeysWithDpapi();
}
```

Se `ProtectKeysWithDpapi` é chamado sem parâmetros, somente a conta de usuário atual do Windows poderá decifrar o anel de chave persistente. Opcionalmente, você pode especificar que qualquer conta de usuário no computador (não apenas a conta de usuário atual) poderá decifrar o anel de chave:

```
public void ConfigureServices(IServiceCollection services)
{
    // All user accounts on the machine can decrypt the keys
    services.AddDataProtection()
        .ProtectKeysWithDpapi(protectToLocalMachine: true);
}
```

## Certificado X.509

Se o aplicativo é distribuído entre vários computadores, pode ser conveniente distribuir um certificado x.509 compartilhado entre as máquinas e configure os aplicativos hospedados para usar o certificado de criptografia de chaves em repouso:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDataProtection()
        .ProtectKeysWithCertificate("3BCE558E2AD3E0E34A7743EAB5AEA2A9BD2575A0");
}
```

Devido às limitações do .NET Framework, somente os certificados com chaves privadas de CAPI têm suporte. Consulte o conteúdo abaixo para obter possíveis soluções para essas limitações.

## Windows DPAPI-NG

**Esse mecanismo está disponível apenas no Windows 8/Windows Server 2012 ou posterior.**

Começando com o Windows 8, o sistema operacional do Windows oferece suporte a DPAPI-NG (também chamado de CNG DPAPI). Para obter mais informações, consulte [sobre o DPAPI CNG](#).

A entidade de segurança é codificada como uma regra de proteção do descritor. No exemplo a seguir que chama [ProtectKeysWithDpapiNG](#), somente o usuário de domínio com o SID especificado pode descriptografar o anel de chave:

```
public void ConfigureServices(IServiceCollection services)
{
    // Uses the descriptor rule "SID=S-1-5-21-..."
    services.AddDataProtection()
        .ProtectKeysWithDpapiNG("SID=S-1-5-21-...",
            flags: DpapiNGProtectionDescriptorFlags.None);
}
```

Também há uma sobrecarga sem parâmetros de [ProtectKeysWithDpapiNG](#). Use esse método de conveniência para especificar a regra "SID = {CURRENT\_ACCOUNT\_SID}", onde *CURRENT\_ACCOUNT\_SID* é o SID da conta de usuário atual do Windows:

```
public void ConfigureServices(IServiceCollection services)
{
    // Use the descriptor rule "SID={current account SID}"
    services.AddDataProtection()
        .ProtectKeysWithDpapiNG();
}
```

Nesse cenário, o controlador de domínio do AD é responsável por distribuir as chaves de criptografia usadas pelas operações NG DPAPI. O usuário de destino poderá decifrar a carga criptografada de qualquer computador ingressado no domínio (desde que o processo é executado sob sua identidade).

## Criptografia baseada em certificado com o Windows DPAPI-NG

Se o aplicativo está em execução no Windows 8.1 / Windows Server 2012 R2 ou posterior, você pode usar o Windows DPAPI-NG para executar a criptografia baseada em certificado. Use a cadeia de caracteres do descritor de regra "certificado = HashId:THUMBPRINT", onde *impressão digital* é a impressão digital codificação hexadecimal SHA1 do certificado:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDataProtection()
        .ProtectKeysWithDpapi("CERTIFICATE=HashId:3BCE558E2...B5AEA2A9BD2575A0",
            flags: DpapiNGProtectionDescriptorFlags.None);
}
```

Qualquer aplicativo apontado para esse repositório deve estar executando no Windows 8.1 / Windows Server 2012 R2 ou posterior para decifrar as chaves.

## Criptografia de chave personalizada

Se os mecanismos de caixa de entrada não forem apropriados, o desenvolvedor pode especificar seu próprio mecanismo de criptografia de chave, fornecendo um personalizado [IXmlEncryptor](#).

# Imutabilidade de chave e as configurações de chave no ASP.NET Core

24/07/2018 • 2 minutes to read • [Edit Online](#)

Depois que um objeto é persistido para o repositório de backup, sua representação para sempre é fixo. Novos dados podem ser adicionados ao repositório de backup, mas os dados existentes nunca podem ser modificados. A principal finalidade desse comportamento é evitar dados corrompidos.

Uma consequência deste comportamento é que, quando uma chave é gravada para o repositório de backup, é imutável. As datas de criação, ativação e expiração nunca podem ser alteradas, embora ele pode ser revogada usando `IKeyManager`. Além disso, suas informações de algorítmicos subjacentes, material de chave mestre e a criptografia em Propriedades do rest também são imutáveis.

Se o desenvolvedor altera qualquer configuração que afeta a persistência de chave, essas alterações não entrarão em vigor até a próxima vez que uma chave é gerada, por meio de uma chamada explícita para `IKeyManager.CreateNewKey` ou por meio do sistema proteção de dados próprio [chave automática geração](#) comportamento. As configurações que afetam a persistência de chave são da seguinte maneira:

- [O tempo de vida de chave padrão](#)
- [A criptografia de chave no mecanismo de rest](#)
- [As informações de algorítmicos contidas na chave do](#)

Se você precisar dessas configurações arranque anteriores à próxima chave automática sem interrupção de tempo, considere fazer uma chamada explícita para `IKeyManager.CreateNewKey` para forçar a criação de uma nova chave. Lembre-se de fornecer uma data de ativação explícita ({agora + 2 dias} é uma boa regra prática para dar tempo para a alteração seja propagada) e a data de expiração na chamada.

## TIP

Tocar o repositório de todos os aplicativos devem especificar as mesmas configurações com o `IDataProtectionBuilder` métodos de extensão. Caso contrário, as propriedades da chave persistente será dependentes do aplicativo específico que invocou as rotinas de geração de chave.

# Formato de armazenamento de chaves no ASP.NET Core

24/07/2018 • 5 minutes to read • [Edit Online](#)

Objetos são armazenados em repouso na representação XML. O diretório padrão para o armazenamento de chaves é % LOCALAPPDATA%\ASP.NET\DataProtection-Keys.

## O <key> elemento

Chaves existem como objetos de nível superior no repositório de chave. Por convenção, as chaves têm o nome do arquivo **chave-{guid}.XML**, onde {guid} é a id da chave. Cada arquivo contém uma única chave. O formato do arquivo é da seguinte maneira.

```
<?xml version="1.0" encoding="utf-8"?>
<key id="80732141-ec8f-4b80-af9c-c4d2d1ff8901" version="1">
  <creationDate>2015-03-19T23:32:02.3949887Z</creationDate>
  <activationDate>2015-03-19T23:32:02.3839429Z</activationDate>
  <expirationDate>2015-06-17T23:32:02.3839429Z</expirationDate>
  <descriptor deserializerType="{deserializerType}">
    <descriptor>
      <encryption algorithm="AES_256_CBC" />
      <validation algorithm="HMACSHA256" />
      <enc:encryptedSecret decryptorType="{decryptorType}" xmlns:enc="...">
        <encryptedKey>
          <!-- This key is encrypted with Windows DPAPI. -->
          <value>AQAAANCM...8/zeP8lcwAg==</value>
        </encryptedKey>
      </enc:encryptedSecret>
    </descriptor>
  </descriptor>
</key>
```

O <key> elemento contém os seguintes atributos e elementos filho:

- A id da chave. Esse valor é tratado como autoritativo; o nome do arquivo é simplesmente uma sutileza por humanos.
- A versão do <key> elemento, no momento é fixado em 1.
- Datas de criação, ativação e expiração da chave.
- Um <descriptor> elemento, que contém informações sobre a implementação de criptografia autenticada contida dentro dessa chave.

No exemplo acima, id da chave é {80732141-ec8f-4b80-af9c-c4d2d1ff8901}, ele foi criado e ativado no dia 19 de março de 2015, e ele tem um tempo de vida de 90 dias. (Ocasionalmente, a data de ativação pode ser um pouco antes da data de criação, como neste exemplo. Isso ocorre devido a um bug em como as APIs funcionam e é inofensivas na prática.)

## O <descriptor> elemento

Externo <descriptor> elemento contém um deserializerType de atributo, que é o nome qualificado pelo assembly de um tipo que implementa IAuthenticatedEncryptorDescriptorDeserializer. Esse tipo é responsável por ler interno <descriptor> elemento e para analisar as informações contidas neles.

O formato específico do <descritor> elemento depende da implementação do Criptografador autenticado encapsulada pela chave e cada tipo de desserializador espera um formato ligeiramente diferente para isso. Em geral, no entanto, esse elemento conterá informações algorítmicas (nomes, tipos, OIDs, ou semelhante) e o material de chave secreta. No exemplo acima, o descritor especifica que essa chave encapsula a criptografia AES-256-CBC + HMACSHA256 validação.

## O <encryptedSecret> elemento

Uma <encryptedSecret> elemento que contém o formulário criptografado do material de chave secreto pode estar presente se [criptografia de segredos em repouso está ativada](#). O atributo `decryptorType` é o nome qualificado pelo assembly de um tipo que implementa `IXmlDecryptor`. Esse tipo é responsável por ler interna <encryptedKey> elemento e descriptografá-los para recuperar o texto sem formatação original.

Assim como acontece com <descritor>, o formato específico do elemento depende do mecanismo de criptografia em repouso em uso. No exemplo acima, a chave mestra é criptografada usando a DPAPI do Windows por comentário.

## O <revogação> elemento

Revogações existem como objetos de nível superior no repositório de chave. Por convenção revogações têm o nome do arquivo **revogação-{timestamp}. XML** (para revogar todas as chaves antes de uma data específica) ou **revogação-{guid}. XML** (para a revogação de uma chave específica). Cada arquivo contém um único <revogação> elemento.

Para revogações de chaves individuais, o conteúdo do arquivo será conforme mostrado abaixo.

```
<?xml version="1.0" encoding="utf-8"?>
<revocation version="1">
  <revocationDate>2015-03-20T22:45:30.2616742Z</revocationDate>
  <key id="eb4fc299-8808-409d-8a34-23fc83d026c9" />
  <reason>human-readable reason</reason>
</revocation>
```

Nesse caso, apenas a chave especificada é revogada. Se a id da chave é "\*", no entanto, como mostra o exemplo abaixo, cuja data de criação está antes da data de revogação especificado de todas as chaves são revogadas.

```
<?xml version="1.0" encoding="utf-8"?>
<revocation version="1">
  <revocationDate>2015-03-20T15:45:45.7366491-07:00</revocationDate>
  <!-- All keys created before the revocation date are revoked. -->
  <key id="*" />
  <reason>human-readable reason</reason>
</revocation>
```

O <motivo> elemento nunca é lido pelo sistema. Ele é simplesmente um local conveniente para armazenar um motivo legível por humanos para revogação.

# Provedores de proteção de dados efêmero no núcleo do ASP.NET

22/06/2018 • 2 minutes to read • [Edit Online](#)

Há cenários em que um aplicativo precisa de um folheto de propaganda `IDataProtectionProvider`. Por exemplo, o desenvolvedor pode apenas suas experiências em um aplicativo de console único, ou o aplicativo em si é transitório (é inserido no script ou uma unidade de projeto de teste). Para dar suporte a esses cenários de `Microsoft.AspNetCore.DataProtection` pacote inclui um tipo `EphemeralDataProtectionProvider`. Esse tipo fornece uma implementação básica de `IDataProtectionProvider` cujo repositório de chave é mantido somente na memória e não é gravado em qualquer armazenamento de backup.

Cada instância de `EphemeralDataProtectionProvider` usa sua própria chave mestra exclusiva. Portanto, se um `IDataProtector` com raiz em um `EphemeralDataProtectionProvider` gera uma carga protegida, essa carga só pode ser desprotegida por um equivalente `IDataProtector` (forem os mesmos `finalidade` cadeia) vinculados ao mesmo `EphemeralDataProtectionProvider` instância.

O exemplo a seguir demonstra a instanciação de um `EphemeralDataProtectionProvider` e usá-lo para proteger e Desproteger dados.

```
using System;
using Microsoft.AspNetCore.DataProtection;

public class Program
{
    public static void Main(string[] args)
    {
        const string purpose = "Ephemeral.App.v1";

        // create an ephemeral provider and demonstrate that it can round-trip a payload
        var provider = new EphemeralDataProtectionProvider();
        var protector = provider.CreateProtector(purpose);
        Console.WriteLine("Enter input: ");
        string input = Console.ReadLine();

        // protect the payload
        string protectedPayload = protector.Protect(input);
        Console.WriteLine($"Protect returned: {protectedPayload}");

        // unprotect the payload
        string unprotectedPayload = protector.Unprotect(protectedPayload);
        Console.WriteLine($"Unprotect returned: {unprotectedPayload}");

        // if I create a new ephemeral provider, it won't be able to unprotect existing
        // payloads, even if I specify the same purpose
        provider = new EphemeralDataProtectionProvider();
        protector = provider.CreateProtector(purpose);
        unprotectedPayload = protector.Unprotect(protectedPayload); // THROWS
    }
}

/*
 * SAMPLE OUTPUT
 *
 * Enter input: Hello!
 * Protect returned: CfDJ8AAAAAAAAAAAAAAA...uGoxWLjGKtm1SkNACQ
 * Unprotect returned: Hello!
 * << throws CryptographicException >>
*/
```

# Compatibilidade no ASP.NET Core

21/06/2018 • 2 minutes to read • [Edit Online](#)

- Substituição do ASP.NET <machineKey> no ASP.NET Core

# Substitua o machineKey ASP.NET no núcleo do ASP.NET

22/06/2018 • 4 minutes to read • [Edit Online](#)

A implementação de `<machineKey>` elemento no ASP.NET é substituível. Isso permite que a maioria das chamadas para rotinas criptográficas ASP.NET para ser roteada por meio de um mecanismo de proteção de dados de substituição, incluindo o novo sistema de proteção de dados.

## Instalação do pacote

### NOTE

O novo sistema de proteção de dados só pode ser instalado em um aplicativo ASP.NET existente direcionado ao .NET 4.5.1 ou posterior. Instalação irá falhar se o aplicativo tem como destino .NET 4.5 ou inferior.

Para instalar o novo sistema de proteção de dados em um projeto de 4.5.1+ ASP.NET existente, instale o pacote Microsoft.AspNetCore.DataProtection.SystemWeb. Isso criará uma instância de sistema de proteção de dados usando o [configuração padrão](#) configurações.

Quando você instala o pacote, ele insere uma linha em *Web.config* que diz ao ASP.NET para usá-la para [mais operações criptográficas](#), incluindo autenticação de formulários, o estado de exibição e chamadas para Protect. A linha é inserida lê da seguinte maneira.

```
<machineKey compatibilityMode="Framework45" dataProtectorType="..." />
```

### TIP

Você pode determinar se o novo sistema de proteção de dados está ativo inspecionando campos como `__VIEWSTATE`, que deve começar com "CfDJ8" como no exemplo a seguir. "CfDJ8" é a representação de base64 do cabeçalho magic "09 F0 C9 F0" que identifica um conteúdo protegido pelo sistema de proteção de dados.

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="CfDJ8AWPr2EQPTBGs3L2GCZ0pk..." />
```

## Configuração de pacote

O sistema de proteção de dados é instanciado com uma configuração de zero a instalação padrão. No entanto, uma vez que por padrão as chaves são mantidas para o sistema de arquivos local, isso não funcionará para aplicativos que são implantados em um farm. Para resolver esse problema, você pode fornecer uma configuração por meio da criação de um tipo que herda DataProtectionStartup e substitui o método ConfigureServices.

Abaixo está um exemplo de um tipo de inicialização de proteção de dados personalizados que configurado onde as chaves são persistentes e como eles são criptografados em repouso. Ela também substitui a política de isolamento de aplicativo padrão, fornecendo seu próprio nome de aplicativo.

```

using System;
using System.IO;
using Microsoft.AspNetCore.DataProtection;
using Microsoft.AspNetCore.DataProtection.SystemWeb;
using Microsoft.Extensions.DependencyInjection;

namespace DataProtectionDemo
{
    public class MyDataProtectionStartup : DataProtectionStartup
    {
        public override void ConfigureServices(IServiceCollection services)
        {
            services.AddDataProtection()
                .SetApplicationName("my-app")
                .PersistKeysToFileSystem(new DirectoryInfo(@"\\server\share\myapp-keys\"))  

                .ProtectKeysWithCertificate("thumbprint");
        }
    }
}

```

#### TIP

Você também pode usar `<machineKey applicationName="my-app" ... />` no lugar de uma chamada explícita para `SetApplicationName`. Esse é um mecanismo de conveniência para evitar forçar o desenvolvedor para criar um tipo derivado de `DataProtectionStartup` se todos os quisessem configurar foi definindo o nome do aplicativo.

Para habilitar essa configuração personalizada, volte para a `Web.config` e procure o `<appSettings>` elemento que instala o pacote adicionado ao arquivo de configuração. Ele se parecerá com a seguinte marcação:

```

<appSettings>
    <!--
    If you want to customize the behavior of the ASP.NET Core Data Protection stack, set the
    "aspnet:dataProtectionStartupType" switch below to be the fully-qualified name of a
    type which subclasses Microsoft.AspNetCore.DataProtection.SystemWeb.DataProtectionStartup.
    -->
    <add key="aspnet:dataProtectionStartupType" value="" />
</appSettings>

```

Preencha o valor em branco com o nome qualificado do assembly do tipo derivado `DataProtectionStartup` que você acabou de criar. Se o nome do aplicativo é `DataProtectionDemo`, isso seria semelhante a abaixo.

```

<add key="aspnet:dataProtectionStartupType"
    value="DataProtectionDemo.MyDataProtectionStartup, DataProtectionDemo" />

```

O sistema de proteção de dados recém-configurada agora está pronto para uso dentro do aplicativo.

# Armazenamento seguro dos segredos do aplicativo em desenvolvimento no ASP.NET Core

02/02/2019 • 16 minutes to read • [Edit Online](#)

Por [Rick Anderson](#), [Daniel Roth](#), e [Scott Addie](#)

[Exibir ou baixar código de exemplo \(como baixar\)](#)

Este documento explica as técnicas para armazenar e recuperar dados confidenciais durante o desenvolvimento de um aplicativo ASP.NET Core. Nunca armazene senhas ou outros dados confidenciais no código-fonte. Segredos de produção não devem ser usados para desenvolvimento ou teste. Você pode armazenar e proteger os segredos de teste e produção do Azure com o [provedor de configuração do Azure Key Vault](#).

## Variáveis de ambiente

Variáveis de ambiente são usadas para evitar o armazenamento de segredos do aplicativo no código ou em arquivos de configuração local. Variáveis de ambiente substituem os valores de configuração para todas as fontes de configuração especificado anteriormente.

Configure a leitura dos valores de variáveis de ambiente chamando `AddEnvironmentVariables` no `Startup` construtor:

```
public Startup(IHostingEnvironment env)
{
    var builder = new ConfigurationBuilder()
        .SetBasePath(env.ContentRootPath)
        .AddJsonFile("appsettings.json",
            optional: false,
            reloadOnChange: true)
        .AddEnvironmentVariables();

    if (env.IsDevelopment())
    {
        builder.AddUserSecrets<Startup>();
    }

    Configuration = builder.Build();
}
```

Considere um aplicativo da web ASP.NET Core no qual **contas de usuário individuais** segurança está habilitada. Uma cadeia de caracteres de conexão de banco de dados padrão está incluída no projeto do `appSettings.JSON` arquivo com a chave `DefaultConnection`. É a cadeia de caracteres de conexão padrão para o LocalDB, que é executado no modo de usuário e não requer uma senha. Durante a implantação de aplicativo, o `DefaultConnection` valor de chave pode ser substituído com o valor da variável de ambiente. A variável de ambiente pode armazenar a cadeia de caracteres de conexão completa com credenciais confidenciais.

#### **WARNING**

Variáveis de ambiente geralmente são armazenadas em texto não criptografado e sem formatação. Se o computador ou processo for comprometido, as variáveis de ambiente podem ser acessadas por pessoas não confiáveis. Medidas adicionais para evitar a divulgação de segredos do usuário podem ser necessárias.

## Secret Manager

A ferramenta Secret Manager armazena dados confidenciais durante o desenvolvimento de um projeto ASP.NET Core. Nesse contexto, uma parte dos dados confidenciais é um segredo do aplicativo. Segredos do aplicativo são armazenados em um local separado da árvore do projeto. Os segredos do aplicativo são compartilhados em vários projetos ou associados a um projeto específico. Os segredos do aplicativo não são verificados no controle de origem.

#### **WARNING**

A ferramenta Secret Manager não criptografa os segredos armazenados e não deve ser tratada como um repositório confiável. Ele é apenas a fins de desenvolvimento. As chaves e valores são armazenados em um arquivo de configuração JSON no diretório de perfil do usuário.

## Como funciona a ferramenta Secret Manager

A ferramenta Secret Manager abstrai os detalhes de implementação, como onde e como os valores são armazenados. Você pode usar a ferramenta sem conhecer esses detalhes de implementação. Os valores são armazenados em um arquivo de configuração do JSON em uma pasta de perfil do usuário do sistema protegido no computador local:

- [Windows](#)
- [macOS](#)
- [Linux](#)

Caminho do sistema de arquivos:

```
%APPDATA%\Microsoft\UserSecrets\<user_secrets_id>\secrets.json
```

Na anterior caminhos de arquivo, substitua `<user_secrets_id>` com o `UserSecretsId` valor especificado na `.csproj` arquivo.

Não escreva código que depende do local ou o formato dos dados salvos com a ferramenta Secret Manager. Esses detalhes de implementação pode ser alterado. Por exemplo, os valores secretos não são criptografados, mas pode ser no futuro.

## Instalar a ferramenta Secret Manager

A ferramenta Secret Manager é fornecido com a CLI do .NET Core no SDK do .NET Core 2.1.300 ou posterior. Para versões do SDK do .NET Core anteriores 2.1.300, a instalação da ferramenta é necessária.

#### **TIP**

Executar `dotnet --version` em um shell de comando para ver o número de versão do SDK do .NET Core instalado.

Um aviso será exibido se a ferramenta inclui o SDK do .NET Core que está sendo usada:

```
The tool 'Microsoft.Extensions.SecretManager.Tools' is now included in the .NET Core SDK.  
Information on resolving this warning is available at (https://aka.ms/dotnetclitools-in-box).
```

Instalar o [secretmanager](#) pacote do NuGet em seu projeto ASP.NET Core. Por exemplo:

```
<Project Sdk="Microsoft.NET.Sdk.Web">  
  <PropertyGroup>  
    <TargetFramework>netcoreapp1.1</TargetFramework>  
    <UserSecretsId>1242d6d6-9df3-4031-b031-d9b27d13c25a</UserSecretsId>  
  </PropertyGroup>  
  <ItemGroup>  
    <PackageReference Include="Microsoft.AspNetCore"  
      Version="1.1.6" />  
    <PackageReference Include="Microsoft.Extensions.Configuration.UserSecrets"  
      Version="1.1.2" />  
    <PackageReference Include="System.Data.SqlClient"  
      Version="4.5.0" />  
  </ItemGroup>  
  <ItemGroup>  
    <DotNetCliToolReference Include="Microsoft.Extensions.SecretManager.Tools"  
      Version="1.0.1" />  
  </ItemGroup>  
</Project>
```

Execute o seguinte comando em um shell de comando para validar a instalação da ferramenta:

```
dotnet user-secrets -h
```

A ferramenta Secret Manager exibe um exemplo de uso e opções de ajuda de comando:

```
Usage: dotnet user-secrets [options] [command]  
  
Options:  
  -?|-h|--help                                Show help information  
  --version                                     Show version information  
  -v|--verbose                                  Show verbose output  
  -p|--project <PROJECT>                         Path to project. Defaults to searching the current  
                                                directory.  
  -c|--configuration <CONFIGURATION>          The project configuration to use. Defaults to 'Debug'.  
  --id                                         The user secret ID to use.  
  
Commands:  
  clear   Deletes all the application secrets  
  list    Lists all the application secrets  
  remove  Removes the specified user secret  
  set     Sets the user secret to the specified value  
  
Use "dotnet user-secrets [command] --help" for more information about a command.
```

#### NOTE

Você deve estar no mesmo diretório que o `.csproj` arquivo para executar as ferramentas definidas na `.csproj` do arquivo `DotNetCliToolReference` elementos.

## Defina um segredo

A ferramenta Secret Manager opera em definições de configuração de específicos do projeto armazenadas no perfil do usuário. Para usar os segredos do usuário, defina uma `UserSecretsId` elemento dentro de uma `PropertyGroup` da `.csproj` arquivo. O valor de `UserSecretsId` é arbitrária, mas é exclusiva para o projeto. Os desenvolvedores geralmente geram um GUID para o `UserSecretsId`.

```
<PropertyGroup>
  <TargetFramework>netcoreapp2.1</TargetFramework>
  <UserSecretsId>79a3edd0-2092-40a2-a04d-dcb46d5ca9ed</UserSecretsId>
</PropertyGroup>
```

```
<PropertyGroup>
  <TargetFramework>netcoreapp1.1</TargetFramework>
  <UserSecretsId>1242d6d6-9df3-4031-b031-d9b27d13c25a</UserSecretsId>
</PropertyGroup>
```

#### TIP

No Visual Studio, clique com botão direito no projeto no Gerenciador de soluções e selecione **gerenciar segredos do usuário** no menu de contexto. Esse gesto adiciona uma `UserSecretsId` elemento, preenchido com um GUID para o `.csproj` arquivo. O Visual Studio abre uma `Secrets` arquivo no editor de texto. Substitua o conteúdo do `Secrets` com os pares chave-valor a ser armazenado. Por exemplo:

```
{
  "Movies": {
    "ConnectionString": "Server=(localdb)\\mssqllocaldb;Database=Movie-1;Trusted_Connection=True;MultipleActiveResultSets=true",
    "ServiceApiKey": "12345"
  }
}
```

A estrutura JSON é mesclada após modificações via `dotnet user-secrets remove` ou `dotnet user-secrets set`. Por exemplo, executando `dotnet user-secrets remove "Movies:ConnectionString"` recolhe o `Movies` literal de objeto. O arquivo modificado é semelhante a:

```
{
  "Movies:ServiceApiKey": "12345"
}
```

Defina um segredo do aplicativo consiste em uma chave e seu valor. O segredo está associado com o projeto `UserSecretsId` valor. Por exemplo, execute o seguinte comando do diretório no qual o `.csproj` arquivo existe:

```
dotnet user-secrets set "Movies:ServiceApiKey" "12345"
```

No exemplo anterior, os dois-pontos indica que `Movies` é um objeto literal com um `ServiceApiKey` propriedade.

A ferramenta Secret Manager pode ser usada de outros diretórios muito. Use o `--project` opção de fornecer o caminho do sistema de arquivos no qual o `.csproj` arquivo existe. Por exemplo:

```
dotnet user-secrets set "Movies:ServiceApiKey" "12345" --project "C:\apps\WebApp1\src\WebApp1"
```

## Defina vários segredos

Um lote de segredos pode ser definido ao canalizar o JSON para o `set` comando. No exemplo a seguir, o *Input* conteúdo do arquivo será canalizado para o `set` comando.

- [Windows](#)
- [macOS](#)
- [Linux](#)

Abra um shell de comando e execute o seguinte comando:

```
type .\input.json | dotnet user-secrets set
```

## Acesse um segredo

O [API de configuração do ASP.NET Core](#) fornece acesso aos segredos Secret Manager. Se seu projeto direcionado ao .NET Framework, instale o [Microsoft.Extensions.Configuration.UserSecrets](#) pacote do NuGet.

No ASP.NET Core 2.0 ou posterior, a fonte de configuração de segredos do usuário é adicionada automaticamente no modo de desenvolvimento quando o projeto chama `CreateDefaultBuilder` para inicializar uma nova instância do host com os padrões pré-configurados. `CreateDefaultBuilder` chamadas `AddUserSecrets` quando o `EnvironmentName` é `Development`:

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>();
```

Quando `CreateDefaultBuilder` não é chamado, adicione a fonte de configuração de segredos do usuário explicitamente chamando `AddUserSecrets` no `Startup` construtor. Chamar `AddUserSecrets` apenas quando o aplicativo é executado no ambiente de desenvolvimento, conforme mostrado no exemplo a seguir:

```
public Startup(IHostingEnvironment env)
{
    var builder = new ConfigurationBuilder()
        .SetBasePath(env.ContentRootPath)
        .AddJsonFile("appsettings.json",
            optional: false,
            reloadOnChange: true)
        .AddEnvironmentVariables();

    if (env.IsDevelopment())
    {
        builder.AddUserSecrets<Startup>();
    }

    Configuration = builder.Build();
}
```

O [API de configuração do ASP.NET Core](#) fornece acesso aos segredos Secret Manager. Instalar o [Microsoft.Extensions.Configuration.UserSecrets](#) pacote do NuGet.

Adicionar a fonte de configuração de segredos do usuário com uma chamada para `AddUserSecrets` no `Startup` construtor:

```
public Startup(IHostingEnvironment env)
{
    var builder = new ConfigurationBuilder()
        .SetBasePath(env.ContentRootPath)
        .AddJsonFile("appsettings.json",
            optional: false,
            reloadOnChange: true)
        .AddEnvironmentVariables();

    if (env.IsDevelopment())
    {
        builder.AddUserSecrets<Startup>();
    }

    Configuration = builder.Build();
}
```

Segredos do usuário podem ser recuperados por meio de `Configuration` API:

```
public class Startup
{
    private string _moviesApiKey = null;

    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        _moviesApiKey = Configuration["Movies:ServiceApiKey"];
    }

    public void Configure(IApplicationBuilder app)
    {
        var result = string.IsNullOrEmpty(_moviesApiKey) ? "Null" : "Not Null";
        app.Run(async (context) =>
        {
            await context.Response.WriteAsync($"Secret is {result}");
        });
    }
}
```

```

public class Startup
{
    private string _moviesApiKey = null;

    public Startup(IHostingEnvironment env)
    {
        var builder = new ConfigurationBuilder()
            .SetBasePath(env.ContentRootPath)
            .AddJsonFile("appsettings.json",
                optional: false,
                reloadOnChange: true)
            .AddEnvironmentVariables();

        if (env.IsDevelopment())
        {
            builder.AddUserSecrets<Startup>();
        }

        Configuration = builder.Build();
    }

    public IConfigurationRoot Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        _moviesApiKey = Configuration["Movies:ServiceApiKey"];
    }

    public void Configure(IApplicationBuilder app)
    {
        var result = string.IsNullOrEmpty(_moviesApiKey) ? "Null" : "Not Null";
        app.Run(async (context) =>
        {
            await context.Response.WriteAsync($"Secret is {result}");
        });
    }
}

```

## Segredos do mapa para um POCO

Mapeando um literal de objeto inteiro para um POCO (uma classe .NET simple com propriedades) é útil para agregar as propriedades relacionadas.

Suponha que o aplicativo *Secrets* arquivo contém os dois segredos do seguintes:

```
{
    "Movies": {
        "ServiceApiKey": "12345",
        "ConnectionString": "Server=(localdb)\\mssqllocaldb;Database=Movie-1;Trusted_Connection=True;MultipleActiveResultSets=true"
    }
}
```

Para mapear os segredos anteriores para um POCO, use o `Configuration` da API [de associação do grafo de objeto](#) recurso. O código a seguir associa a um personalizado `MovieSettings` POCO e acessa o `ServiceApiKey` valor da propriedade:

```

var moviesConfig = Configuration.GetSection("Movies")
    .Get<MovieSettings>();
_moviesApiKey = moviesConfig.ServiceApiKey;

```

```
var moviesConfig = new MovieSettings();
Configuration.GetSection("Movies").Bind(moviesConfig);
_moviesApiKey = moviesConfig.ServiceApiKey;
```

O `Movies:ConnectionString` e `Movies:ServiceApiKey` segredos são mapeados para as respectivas propriedades no `MovieSettings`:

```
public class MovieSettings
{
    public string ConnectionString { get; set; }

    public string ServiceApiKey { get; set; }
}
```

## Cadeia de caracteres de substituição com segredos

Armazenar senhas em texto sem formatação é inseguro. Por exemplo, uma cadeia de caracteres de conexão de banco de dados armazenados em `appSettings.JSON` pode incluir uma senha para o usuário especificado:

```
{
  "ConnectionStrings": {
    "Movies": "Server=(localdb)\\mssqllocaldb;Database=Movie-1;User
Id=johndoe;Password=pass123;MultipleActiveResultSets=true"
  }
}
```

Uma abordagem mais segura é armazenar a senha como um segredo. Por exemplo:

```
dotnet user-secrets set "DbPassword" "pass123"
```

Remover o `Password` par chave-valor da cadeia de conexão na `appSettings.JSON`. Por exemplo:

```
{
  "ConnectionStrings": {
    "Movies": "Server=(localdb)\\mssqllocaldb;Database=Movie-1;User
Id=johndoe;MultipleActiveResultSets=true"
  }
}
```

Valor do segredo pode ser definida em um `SqlConnectionStringBuilder` do objeto `senha` propriedade para concluir a cadeia de caracteres de conexão:

```
public class Startup
{
    private string _connection = null;

    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        var builder = new SqlConnectionStringBuilder(
            Configuration.GetConnectionString("Movies"));
        builder.Password = Configuration["DbPassword"];
        _connection = builder.ConnectionString;
    }

    public void Configure(IApplicationBuilder app)
    {
        app.Run(async (context) =>
        {
            await context.Response.WriteAsync($"DB Connection: {_connection}");
        });
    }
}
```

```

public class Startup
{
    private string _connection = null;

    public Startup(IHostingEnvironment env)
    {
        var builder = new ConfigurationBuilder()
            .SetBasePath(env.ContentRootPath)
            .AddJsonFile("appsettings.json",
                optional: false,
                reloadOnChange: true)
            .AddEnvironmentVariables();

        if (env.IsDevelopment())
        {
            builder.AddUserSecrets<Startup>();
        }

        Configuration = builder.Build();
    }

    public IConfigurationRoot Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        var builder = new SqlConnectionStringBuilder(
            Configuration.GetConnectionString("Movies"));
        builder.Password = Configuration["DbPassword"];
        _connection = builder.ConnectionString;
    }

    public void Configure(IApplicationBuilder app)
    {
        app.Run(async (context) =>
        {
            await context.Response.WriteAsync($"DB Connection: {_connection}");
        });
    }
}

```

## Lista os segredos

Suponha que o aplicativo *Secrets* arquivo contém os dois segredos do seguintes:

```
{
    "Movies": {
        "ServiceApiKey": "12345",
        "ConnectionString": "Server=(localdb)\\mssqllocaldb;Database=Movie-1;Trusted_Connection=True;MultipleActiveResultSets=true"
    }
}
```

Execute o seguinte comando no diretório no qual o *.csproj* arquivo existe:

```
dotnet user-secrets list
```

A saída a seguir é exibida:

```
Movies:ConnectionString = Server=(localdb)\mssqllocaldb;Database=Movie-  
1;Trusted_Connection=True;MultipleActiveResultSets=true  
Movies:ServiceApiKey = 12345
```

No exemplo anterior, dois-pontos em nomes de chave denota a hierarquia de objetos dentro *Secrets*.

## Remover um segredo único

Suponha que o aplicativo *Secrets* arquivo contém os dois segredos do seguintes:

```
{
  "Movies": {
    "ServiceApiKey": "12345",
    "ConnectionString": "Server=(localdb)\\mssqllocaldb;Database=Movie-  
1;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}
```

Execute o seguinte comando no diretório no qual o *.csproj* arquivo existe:

```
dotnet user-secrets remove "Movies:ConnectionString"
```

O aplicativo *Secrets* arquivo foi modificado para remover o par chave-valor associado a `MoviesConnectionString` chave:

```
{
  "Movies": {
    "ServiceApiKey": "12345"
  }
}
```

Executando `dotnet user-secrets list` exibe a seguinte mensagem:

```
Movies:ServiceApiKey = 12345
```

## Remover todos os segredos

Suponha que o aplicativo *Secrets* arquivo contém os dois segredos do seguintes:

```
{
  "Movies": {
    "ServiceApiKey": "12345",
    "ConnectionString": "Server=(localdb)\\mssqllocaldb;Database=Movie-  
1;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}
```

Execute o seguinte comando no diretório no qual o *.csproj* arquivo existe:

```
dotnet user-secrets clear
```

Todos os segredos do usuário para o aplicativo tem sido excluídos do *Secrets* arquivo:

```
{}
```

Executando `dotnet user-secrets list` exibe a seguinte mensagem:

```
No secrets configured for this application.
```

## Recursos adicionais

- [Configuração no ASP.NET Core](#)
- [Provedor de configuração do Cofre de chaves do Azure no ASP.NET Core](#)

# Impor HTTPS no ASP.NET Core

08/01/2019 • 17 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

Este documento demonstra como:

- Exigir HTTPS para todas as solicitações.
- Redirecione todas as solicitações HTTP para HTTPS.

Nenhuma API pode impedir que um cliente envie dados confidenciais na primeira solicitação.

## WARNING

Fazer **não** usar `RequireHttpsAttribute` em APIs da Web que recebe informações confidenciais. `RequireHttpsAttribute` usa códigos de status HTTP para redirecionar navegadores de HTTP para HTTPS. Os clientes da API não podem compreender ou obedecem redirecionamentos de HTTP para HTTPS. Esses clientes podem enviar informações por meio de HTTP. As APIs da Web deverá:

- Não realizar a escuta em HTTP.
- Feche a conexão com o código de status 400 (solicitação incorreta) e não atender à solicitação.

## Exigir HTTPS

É recomendável que produção do ASP.NET Core chamada de aplicativos web:

- Middleware de redirecionamento de HTTPS ([UseHttpsRedirection](#)) para redirecionar solicitações HTTP para HTTPS.
- Middleware HSTS ([UseHsts](#)) para enviar cabeçalhos de protocolo de segurança de transporte estrito (HSTS) HTTP aos clientes.

## NOTE

Aplicativos implantados em uma configuração de proxy reverso permitem que o proxy lidar com a segurança de conexão (HTTPS). Se o proxy também manipula o redirecionamento de HTTPS, não é necessário usar o Middleware de redirecionamento de HTTPS. Se o servidor proxy também manipula a HSTS cabeçalhos de gravação (por exemplo, [dar suporte a HSTS nativos no IIS 10.0 \(1709\) ou posterior](#)), Middleware HSTS não é necessário para o aplicativo. Para obter mais informações, consulte [Opt-out de HTTPS/HSTS na criação do projeto](#).

## UseHttpsRedirection

O código a seguir chama `UseHttpsRedirection` no `Startup` classe:

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseCookiePolicy();

    app.UseMvc();
}

```

O código realçado anterior:

- Usa o padrão [HttpsRedirectionOptions.RedirectStatusCode \(Status307TemporaryRedirect\)](#).
- Usa o padrão [HttpsRedirectionOptions.HttpsPort](#) (null), a menos que substituído pela `ASPNETCORE_HTTPS_PORT` variável de ambiente ou [IServerAddressesFeature](#).

É recomendável usar redirecionamentos temporários em vez de redirecionamentos permanentes. Cache de link pode causar um comportamento instável em ambientes de desenvolvimento. Se você preferir enviar um código de status de redirecionamento permanente quando o aplicativo estiver em um ambiente de não desenvolvimento, consulte o [configurar redirecionamentos permanentes em produção](#) seção. É recomendável usar [HSTS](#) para sinalizar para os clientes que proteger somente o recurso deve ser redirecionada para o aplicativo (somente em produção).

## Configuração de porta

Uma porta deve estar disponível para o middleware redirecionar uma solicitação não segura para HTTPS. Se nenhuma porta estiver disponível:

- Redirecionamento de HTTPS não ocorre.
- O middleware registra o aviso "Falha ao determinar a porta https para redirecionamento."

Especifique a porta HTTPS usando qualquer uma das seguintes abordagens:

- Definir [HttpsRedirectionOptions.HttpsPort](#).
- Defina as `ASPNETCORE_HTTPS_PORT` variável de ambiente ou [definição de configuração do Host da Web https\\_port](#):

**Chave:** `https_port`

**Tipo:** `string`

**Padrão:** Um valor padrão não está definido.

**Definido usando:** `UseSetting`

**Variável de ambiente:** `<PREFIX_>HTTPS_PORT` (É o prefixo `ASPNETCORE_` ao usar o [Host Web](#).)

Ao configurar uma [IWebHostBuilder](#) em `Program`:

```

public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseSetting("https_port", "8080")
            .UseStartup<Startup>();
}

```

- Indicar uma porta com o esquema segura usando o `ASPNETCORE_URLS` variável de ambiente. A variável de ambiente configura o servidor. O middleware indiretamente descobre a porta HTTPS por meio de `IServerAddressesFeature`. Essa abordagem não funciona em implantações de proxy reverso.
- No desenvolvimento, defina uma URL HTTPS `launchsettings.json`. Habilite HTTPS quando o IIS Express é usado.
- Configurar um ponto de extremidade de URL HTTPS para uma implantação de borda para o público do `Kestrel` server ou `HTTP.sys` server. Somente **uma porta HTTPS** é usado pelo aplicativo. O middleware descobre a porta por meio de `IServerAddressesFeature`.

#### NOTE

Quando um aplicativo é executado em uma configuração de proxy reverso, `IServerAddressesFeature` não está disponível. Defina a porta usando uma das outras abordagens descritas nesta seção.

Quando o Kestrel ou HTTP.sys é usado como um servidor de borda para o público, o Kestrel ou HTTP.sys deve ser configurado para escutar em ambos:

- A porta segura em que o cliente é redirecionado (normalmente, 443 em 5001 no desenvolvimento e produção).
- A porta não segura (normalmente, 80 na produção) e 5000 no desenvolvimento.

A porta não segura deve estar acessível pelo cliente para que o aplicativo para receber uma solicitação não segura e redirecionar o cliente para a porta segura.

Para obter mais informações, consulte [configuração de ponto de extremidade do Kestrel](#) ou [Implementação do servidor Web HTTP.sys no ASP.NET Core](#).

#### Cenários de implantação

Qualquer firewall entre o cliente e o servidor também deve ter as portas de comunicação aberto para o tráfego.

Se as solicitações são encaminhadas em uma configuração de proxy reverso, use [Middleware de cabeçalhos encaminhados](#) antes de chamar Middleware de redirecionamento de HTTPS. Encaminhado atualizações de Middleware de cabeçalhos a `Request.Scheme`, usando o `X-Forwarded-Proto` cabeçalho. O middleware permite redirecionar URLs e outras políticas de segurança para funcionar corretamente. Quando o Middleware de cabeçalhos encaminhados não for usado, o aplicativo de back-end não pode receber o esquema correto e acabar em um loop de redirecionamento. Uma mensagem de erro comum do usuário final é que muitos redirecionamentos ocorreram.

Ao implantar o serviço de aplicativo do Azure, siga as orientações em [Tutorial: vincular um certificado SSL personalizado ao serviço Aplicativos Web do Azure](#).

#### Opções

O seguinte realçado código chama `AddHttpsRedirection` para configurar as opções de middleware:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    services.AddHsts(options =>
    {
        options.Preload = true;
        options.IncludeSubDomains = true;
        options.MaxAge = TimeSpan.FromDays(60);
        options.ExcludedHosts.Add("example.com");
        options.ExcludedHosts.Add("www.example.com");
    });

    services.AddHttpsRedirection(options =>
    {
        options.RedirectStatusCode = StatusCodes.Status307TemporaryRedirect;
        options.HttpsPort = 5001;
    });
}
```

Chamando `AddHttpsRedirection` só é necessário alterar os valores de `HttpsPort` ou `RedirectStatusCode`.

O código realçado anterior:

- Conjuntos `HttpsRedirectionOptions.RedirectStatusCode` para `Status307TemporaryRedirect`, que é o valor padrão. Use os campos do `StatusCodes` classe atribuições para `RedirectStatusCode`.
- Define a porta HTTPS para 5001. O valor padrão é 443.

#### Configurar redirecionamentos permanentes em produção

O middleware usará como padrão para envio de uma `Status307TemporaryRedirect` com todos os redirecionamentos. Se você preferir enviar um código de status de redirecionamento permanente quando o aplicativo estiver em um ambiente de não desenvolvimento, encapsule a configuração de opções de middleware em uma verificação condicional para um ambiente de desenvolvimento não.

Ao configurar uma `IWebHostBuilder` na `Startup.cs`:

```
public void ConfigureServices(IServiceCollection services)
{
    // IHostingEnvironment (stored in _env) is injected into the Startup class.
    if (!_env.IsDevelopment())
    {
        services.AddHttpsRedirection(options =>
        {
            options.RedirectStatusCode = StatusCodes.Status308PermanentRedirect;
            options.HttpsPort = 443;
        });
    }
}
```

## Abordagem alternativa de Middleware de redirecionamento de HTTPS

Uma alternativa ao uso de Middleware de redirecionamento de HTTPS (`useHttpsRedirection`) é usar o Middleware de reconfiguração de URL (`AddRedirectToHttps`). `AddRedirectToHttps` também pode definir o código de status e a porta quando o redirecionamento é executado. Para obter mais informações, consulte [Middleware de reconfiguração de URL](#).

Ao redirecionar para HTTPS sem a necessidade de regras de redirecionamento adicional, é recomendável usar o Middleware de redirecionamento de HTTPS (`useHttpsRedirection`) descrito neste tópico.

O `RequireHttpsAttribute` é usada para exigir HTTPS. `[RequireHttpsAttribute]` pode decorar controladores ou métodos, ou podem ser aplicadas globalmente. Para aplicar o atributo globalmente, adicione o seguinte código ao `ConfigureServices` em `Startup`:

```
// Requires using Microsoft.AspNetCore.Mvc;
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<MvcOptions>(options =>
    {
        options.Filters.Add(new RequireHttpsAttribute());
    });
}
```

O código realçado anterior requer que todas as solicitações usem `HTTPS`; portanto, as solicitações HTTP são ignoradas. O seguinte código realçado redireciona todas as solicitações HTTP para HTTPS:

```
// Requires using Microsoft.AspNetCore.Rewrite;
public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole(Configuration.GetSection("Logging"));
    loggerFactory.AddDebug();

    var options = new RewriteOptions()
        .AddRedirectToHttps();

    app.UseRewriter(options);
}
```

Para obter mais informações, consulte [Middleware de reconfiguração de URL](#). O middleware também permite que o aplicativo para definir o código de status ou o código de status e a porta quando o redirecionamento é executado.

Exigir HTTPS globalmente (`options.Filters.Add(new RequireHttpsAttribute());`) é uma prática recomendada de segurança. Aplicando o `[RequireHttps]` atributo a todas as páginas do Razor/controladores não é considerado tão seguro quanto exigir HTTPS globalmente. Você não pode garantir o `[RequireHttps]` atributo é aplicado quando novos controladores e páginas Razor são adicionadas.

## Protocolo de segurança de transporte estrito HTTP (HSTS)

Por [OWASP](#), [segurança de transporte estrito HTTP \(HSTS\)](#) é um aprimoramento de segurança opcional é especificado por um aplicativo web com o uso de um cabeçalho de resposta. Quando um [navegador que ofereça suporte a HSTS](#) recebe esse cabeçalho:

- O navegador armazena a configuração para o domínio que evita o envio de qualquer comunicação por HTTP. O navegador força todas as comunicações via HTTPS.
- O navegador impede que o usuário usando os certificados não confiáveis ou é inválidos. O navegador desativa prompts que permitem que um usuário para confiar temporariamente esses certificados.

Como HSTS é imposta pelo cliente, ele tem algumas limitações:

- O cliente deve oferecer suporte a HSTS.
- HSTS requer pelo menos uma solicitação HTTPS bem-sucedida para estabelecer a política HSTS.
- O aplicativo deve verificar todas as solicitações HTTP e redirecionar ou rejeitar a solicitação HTTP.

ASP.NET Core 2.1 ou posterior implementa HSTS com o `UseHsts` método de extensão. O código a seguir chama `UseHsts` quando o aplicativo não está no [modo de desenvolvimento](#):

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseCookiePolicy();

    app.UseMvc();
}

```

`UseHsts` não é recomendado em desenvolvimento porque as configurações de HSTS são altamente armazenáveis em cache por navegadores. Por padrão, `UseHsts` exclui o endereço de loopback local.

Para ambientes de produção implementando HTTPS pela primeira vez, definir inicial `HstsOptions.MaxAge` com um valor pequeno usando um do `TimeSpan` métodos. Defina o valor de horas como não mais do que um único dia caso você precise reverter a infraestrutura HTTPS para HTTP. Depois que você estiver confiante em sustentabilidade da configuração do HTTPS, aumente o valor de idade máxima HSTS; um valor comumente usado é um ano.

O código a seguir:

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    services.AddHsts(options =>
    {
        options.Preload = true;
        options.IncludeSubDomains = true;
        options.MaxAge = TimeSpan.FromDays(60);
        options.ExcludedHosts.Add("example.com");
        options.ExcludedHosts.Add("www.example.com");
    });

    services.AddHttpsRedirection(options =>
    {
        options.RedirectStatusCode = StatusCodes.Status307TemporaryRedirect;
        options.HttpsPort = 5001;
    });
}

```

- Define o parâmetro de pré-carregamento do cabeçalho de segurança de transporte estrito. Pré-carregamento não faz parte dos [especificação RFC HSTS](#), mas é compatível com navegadores da web para pré-carregar sites HSTS na nova instalação. Veja <https://hstspreload.org/> para obter mais informações.
- Habilita `includeSubDomain`, que se aplica a política HSTS para hospedar subdomínios.
- Define explicitamente o parâmetro de idade máxima do cabeçalho de segurança de transporte estrito para 60 dias. Se não for definido, o padrão é 30 dias. Consulte a [max-age diretiva](#) para obter mais informações.
- Adiciona `example.com` à lista de hosts a serem excluídos.

`UseHsts` Exclui os seguintes hosts de loopback:

- `localhost` : O endereço de loopback do IPv4.
- `127.0.0.1` : O endereço de loopback do IPv4.
- `[::1]` : O endereço de loopback do IPv6.

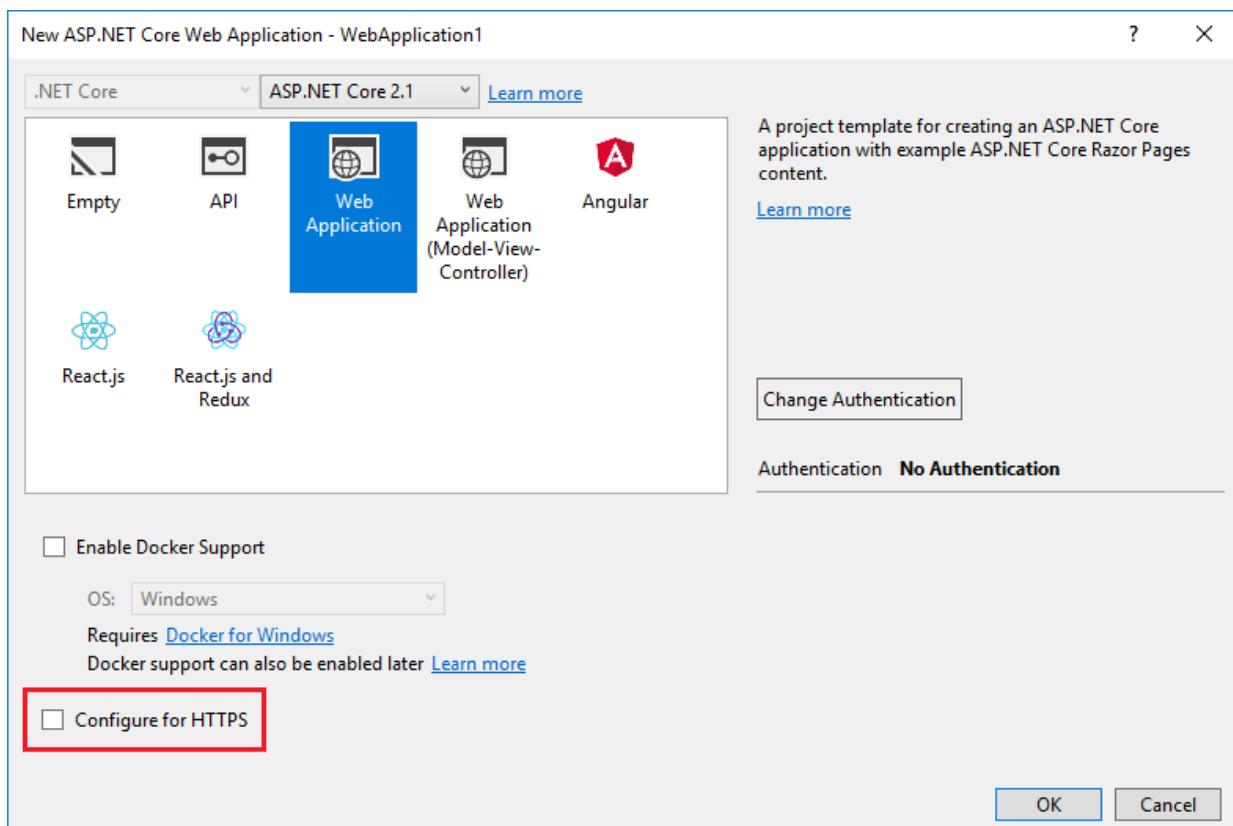
## Recusa de HTTPS/HSTS na criação do projeto

Em alguns cenários de serviço de back-end em que a segurança de conexão é manipulada na borda da rede para o público, configurando a segurança de conexão em cada nó não é necessária. Aplicativos gerados a partir de modelos no Visual Studio ou na Web a `dotnet new` comando enable [redirecionamento a HTTPS e HSTS](#). Para implantações que não exigem esses cenários, você pode recusar HTTPS/HSTS quando o aplicativo é criado a partir do modelo.

A recusa de HTTPS/HSTS:

- [Visual Studio](#)
- [CLI do .NET Core](#)

Desmarque a **configurar para HTTPS** caixa de seleção.



## Confiar no certificado de desenvolvimento do ASP.NET Core HTTPS no Windows e macOS

SDK do .NET core inclui um certificado de desenvolvimento de HTTPS. O certificado é instalado como parte da experiência de primeira execução. Por exemplo, `dotnet --info` produz uma saída semelhante à seguinte:

```
ASP.NET Core
-----
Successfully installed the ASP.NET Core HTTPS Development Certificate.
To trust the certificate run 'dotnet dev-certs https --trust' (Windows and macOS only).
For establishing trust on other platforms refer to the platform specific documentation.
For more information on configuring HTTPS see https://go.microsoft.com/fwlink/?linkid=848054.
```

Instalar o SDK do .NET Core instala o certificado de desenvolvimento do ASP.NET Core HTTPS para o repositório de certificados de usuário local. O certificado foi instalado, mas não é confiável. Confiar em um certificado execute a etapa única para executar o dotnet `dev-certs` ferramenta:

```
dotnet dev-certs https --trust
```

O comando a seguir fornece ajuda sobre o `dev-certs` ferramenta:

```
dotnet dev-certs https --help
```

## Como configurar um certificado de desenvolvedor para o Docker

[Ver esse problema de GitHub.](#)

## Informações adicionais

- [Configure o ASP.NET Core para trabalhar com servidores proxy e平衡adores de carga](#)
- [Hospede o ASP.NET Core no Linux com o Apache: Configuração HTTPS](#)
- [Hospede o ASP.NET Core no Linux com Nginx: Configuração HTTPS](#)
- [Como configurar o SSL no IIS](#)
- [Suporte a navegador HSTS OWASP](#)

# Suporte da UE Data Protection GDPR (regulamento geral) no ASP.NET Core

26/01/2019 • 9 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

O ASP.NET Core fornece modelos e APIs para ajudar a atender a alguns dos [regulamentação de proteção de dados geral \(GDPR\) da UE](#) requisitos:

- Os modelos de projeto incluem pontos de extensão e a marcação de stub que você pode substituir por sua privacidade e a política de uso de cookies.
- Um recurso de consentimento do cookie permite que você pedir consentimento (e acompanhar uma) de seus usuários para armazenar informações pessoais. Se um usuário não tiver consentido para coleta de dados e o aplicativo tem `CheckConsentNeeded` definido como `true`, não-essenciais cookies não são enviados para o navegador.
- Os cookies podem ser marcados como essenciais. Cookies essenciais são enviados ao navegador, mesmo quando o usuário não tiver consentido e acompanhamento está desabilitado.
- [Cookies de sessão e TempData](#) não são funcionais quando o rastreamento está desabilitado.
- O [gerenciar identidades](#) página fornece um link para baixar e excluir dados de usuário.

O [aplicativo de exemplo](#) permite que você teste a maioria dos pontos de extensão de GDPR e APIs adicionadas para os modelos do ASP.NET Core 2.1. Consulte a [Leiaime](#) arquivo para obter instruções de teste.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Suporte GDPR do ASP.NET Core no código de modelo gerado

Páginas do Razor e MVC projetos criados com os modelos de projeto incluem o seguinte suporte GDPR:

- `CookiePolicyOptions` e `UseCookiePolicy` são definidos no `Startup`.
- O `_CookieConsentPartial.cshtml` exibição parcial.
- O `Pages/Privacy.cshtml` página ou `Views/Home/Privacy.cshtml` exibição fornece uma página para a política de privacidade do seu site de detalhe. O `_CookieConsentPartial.cshtml` arquivo gera um link para a página de privacidade.
- Para os aplicativos criados com as contas de usuário individual, a página Gerenciar fornece links para baixar e excluir [dados pessoais do usuário](#).

### **CookiePolicyOptions e UseCookiePolicy**

`CookiePolicyOptions` são inicializadas em `Startup.ConfigureServices`:

```

public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services
    // to the container.
    public void ConfigureServices(IServiceCollection services)
    {
        services.Configure<CookiePolicyOptions>(options =>
        {
            // This lambda determines whether user consent for non-essential cookies
            // is needed for a given request.
            options.CheckConsentNeeded = context => true;
            options.MinimumSameSitePolicy = SameSiteMode.None;
        });

        services.AddDbContext<ApplicationContext>(options =>
            options.UseSqlServer(
                Configuration.GetConnectionString("DefaultConnection")));
        services.AddDefaultIdentity<IdentityUser>()
            .AddEntityFrameworkStores<ApplicationContext>();

        // If the app uses session state, call AddSession.
        // services.AddSession();

        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }
}

// This method gets called by the runtime. Use this method to configure the
// HTTP request pipeline.
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseCookiePolicy();

    app.UseAuthentication();

    // If the app uses session state, call Session Middleware after Cookie
    // Policy Middleware and before MVC Middleware.
    // app.UseSession();

    app.UseMvc();
}
}

```

[UseCookiePolicy](#) é chamado no `Startup.Configure` :

```

public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services
    // to the container.
    public void ConfigureServices(IServiceCollection services)
    {
        services.Configure<CookiePolicyOptions>(options =>
        {
            // This lambda determines whether user consent for non-essential cookies
            // is needed for a given request.
            options.CheckConsentNeeded = context => true;
            options.MinimumSameSitePolicy = SameSiteMode.None;
        });

        services.AddDbContext<ApplicationContext>(options =>
            options.UseSqlServer(
                Configuration.GetConnectionString("DefaultConnection")));
        services.AddDefaultIdentity<IdentityUser>()
            .AddEntityFrameworkStores<ApplicationContext>();

        // If the app uses session state, call AddSession.
        // services.AddSession();

        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }
}

// This method gets called by the runtime. Use this method to configure the
// HTTP request pipeline.
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseCookiePolicy();

    app.UseAuthentication();

    // If the app uses session state, call Session Middleware after Cookie
    // Policy Middleware and before MVC Middleware.
    // app.UseSession();

    app.UseMvc();
}
}

```

### **\_CookieConsentPartial.cshtml partial view**

O *\_CookieConsentPartial.cshtml* exibição parcial:

```

@using Microsoft.AspNetCore.Http.Features

{@
    var consentFeature = Context.Features.Get<ITrackingConsentFeature>();
    var showBanner = !consentFeature?.CanTrack ?? false;
    var cookieString = consentFeature?.CreateConsentCookie();
}

@if (showBanner)
{
    <nav id="cookieConsent" class="navbar navbar-default navbar-fixed-top" role="alert">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#cookieConsent .navbar-collapse">
                    <span class="sr-only">Toggle cookie consent banner</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <span class="navbar-brand"><span class="glyphicon glyphicon-info-sign" aria-hidden="true"></span></span>
            </div>
            <div class="collapse navbar-collapse">
                <p class="navbar-text">
                    Use this space to summarize your privacy and cookie use policy.
                </p>
                <div class="navbar-right">
                    <a asp-page="/Privacy" class="btn btn-info navbar-btn">Learn More</a>
                    <button type="button" class="btn btn-default navbar-btn" data-cookie-string="@cookieString">Accept</button>
                </div>
            </div>
        </div>
    </nav>
    <script>
        (function () {
            document.querySelector("#cookieConsent button[data-cookie-string]").addEventListener("click", function (el) {
                document.cookie = el.target.dataset.cookieString;
                document.querySelector("#cookieConsent").classList.add("hidden");
            }, false);
        })();
    </script>
}

```

Essa parcial:

- Obtém o estado do controle para o usuário. Se o aplicativo estiver configurado para exigir o consentimento, o usuário deve consentir antes de cookies podem ser controlados. Se for necessário o consentimento, o painel de consentimento do cookie é fixa na parte superior da barra de navegação criada pelo *layout.cshtml* arquivo.
- Fornece um HTML `<p>` elemento para resumir a sua privacidade e cookie usar a política.
- Fornece um link para a página de privacidade ou exibição onde você pode detalhar a política de privacidade do seu site.

## Cookies essenciais

Se o consentimento não foi fornecido, somente os cookies marcados essenciais são enviados para o navegador. O código a seguir faz com que um cookie essenciais:

```
public IActionResult OnPostCreateEssentialAsync()
{
    HttpContext.Response.Cookies.Append(Constants.EssentialSec,
        DateTime.Now.Second.ToString(),
        new CookieOptions() { IsEssential = true });

    ResponseCookies = Response.Headers[HeaderNames.SetCookie].ToString();

    return RedirectToPage("./Index");
}
```

## Cookies de estado de sessão e o provedor de TempData não são essenciais

O [provedor de Tempdata](#) cookie não é essencial. Se o controle estiver desabilitado, o provedor de TempData não é funcional. Para habilitar o provedor de TempData quando o rastreamento está desabilitado, marcar o cookie de TempData como essencial para `Startup.ConfigureServices`:

```
// The TempData provider cookie is not essential. Make it essential
// so TempData is functional when tracking is disabled.
services.Configure<CookieTempDataProviderOptions>(options => {
    options.Cookie.IsEssential = true;
});
```

[Estado de sessão](#) cookies não são essenciais. Estado de sessão não está funcionando quando o rastreamento está desabilitado.

## Dados pessoais

Aplicativos ASP.NET Core criados com contas de usuário individuais incluem código para baixar e excluir dados pessoais.

Selecione o nome de usuário e, em seguida, selecione **dados pessoais**:

The screenshot shows a web browser window titled "WebApp1" with the URL "https://localhost:44306/Identity/Account/Manage/PersonalData". The page is titled "Manage your account" and has a sub-header "Change your account settings". There are several navigation links: "Profile", "Password", "Two-factor authentication", and "Personal data" (which is highlighted with a blue background). Below these, a section titled "Personal Data" contains a note: "Your account contains personal data that you have given us. This page allows you to download or delete that data. Deleting this data will permanently remove your account, and this cannot be recovered." Two buttons are present: "Download" and "Delete". At the bottom of the page, there is a copyright notice: "© 2018 - WebApp1".

Notas:

- Para gerar a `Account/Manage` de código, consulte [Scaffold identidade](#).
- O **exclua** e **baixar** links atuam somente em dados de identidade padrão. Aplicativos que criam os dados de usuário personalizada devem ser estendidos para exclusão/baixar os dados de usuário personalizada. Para obter mais informações, consulte [adicionar, baixar e excluir dados de usuário personalizada para identidade](#).
- Salva os tokens para o usuário que são armazenados na tabela de banco de dados de identidade `AspNetUserTokens` são excluídos quando o usuário é excluído por meio do comportamento de exclusão em cascata devido ao [chave estrangeira](#).
- [Autenticação de provedor externo](#), como Facebook e Google, não está disponível antes da política de cookie é aceito.

## Criptografia em repouso

Alguns bancos de dados e mecanismos de armazenamento permitem a criptografia em repouso.

Criptografia em repouso:

- Criptografa dados armazenados automaticamente.
- Criptografa sem configuração, programação ou outro trabalho para o software que acessa os dados.
- É a opção mais segura e fácil.
- Permite que o banco de dados gerenciar chaves e criptografia.

Por exemplo:

- Microsoft SQL e SQL Azure fornecem [Transparent Data Encryption \(TDE\)](#).
- [SQL Azure](#) criptografa o banco de dados por padrão
- [Blobs, arquivos, tabela e o armazenamento de filas do Azure](#) são criptografados por padrão.

Para bancos de dados que não fornecem internos de criptografia em repouso, você poderá usar a criptografia de disco para fornecer a mesma proteção. Por exemplo:

- [BitLocker para Windows Server](#)
- Linux:
  - [eCryptfs](#)
  - [EncFS](#).

## Recursos adicionais

- [Microsoft.com/GDPR](#)

# Provedor de configuração do Cofre de chaves do Azure no ASP.NET Core

12/02/2019 • 28 minutes to read • [Edit Online](#)

Por [Luke Latham](#) e [Andrew Stanton-Nurse](#)

Este documento explica como usar o [Microsoft Azure Key Vault](#) provedor de configuração para carregar valores de configuração de aplicativo de segredos do Cofre de chaves do Azure. O Azure Key Vault é um serviço baseado em nuvem que ajuda a proteger chaves criptográficas e segredos usados por aplicativos e serviços. Cenários comuns para usar o Azure Key Vault com aplicativos ASP.NET Core incluem:

- Controlando o acesso aos dados de configuração confidenciais.
- Atende ao requisito para FIPS 140-2 nível 2 validados módulos de segurança de Hardware (HSM) ao armazenar dados de configuração.

Esse cenário está disponível para aplicativos que usam o ASP.NET Core 2.1 ou posterior.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Pacotes

Para usar o provedor de configuração do Cofre de chave do Azure, adicione uma referência de pacote para o [Microsoft.Extensions.Configuration.AzureKeyVault](#) pacote.

Para adotar a [identidades para recursos do Azure gerenciadas](#) cenário, adicione uma referência de pacote para o [Microsoft.Azure.Services.AppAuthentication](#) pacote.

### NOTE

No momento da gravação, a versão estável mais recente do `Microsoft.Azure.Services.AppAuthentication`, versão `1.0.3`, fornece suporte para [atribuído pelo sistema de identidades gerenciadas](#). Suporte para [atribuída ao usuário identidades gerenciadas](#) está disponível no `1.0.2-preview` pacote. Este tópico demonstra o uso de identidades gerenciadas pelo sistema, e o aplicativo de exemplo fornecido usa a versão `1.0.3` do `Microsoft.Azure.Services.AppAuthentication` pacote.

## Aplicativo de exemplo

O aplicativo de exemplo é executado em qualquer um dos dois modos, determinados pelo `#define` instrução na parte superior dos *Program.cs* arquivo:

- `Basic` – Demonstra o uso de uma ID do aplicativo do Azure Key Vault e a senha (segredo do cliente) para acessar os segredos armazenados no cofre de chaves. Implantar o `Basic` versão de exemplo em qualquer host capaz de atender a um aplicativo ASP.NET Core. Siga as orientações na [ID do aplicativo de uso e o segredo do cliente para aplicativos do Azure não hospedados](#) seção.
- `Managed` – Demonstra como usar [identidades para recursos do Azure gerenciadas](#) para autenticar o aplicativo ao Azure Key Vault com autenticação do Azure AD sem credenciais armazenadas no código ou na configuração do aplicativo. Ao usar identidades gerenciadas para autenticar, uma ID de aplicativo do Azure AD e a senha (segredo do cliente) não é necessário. O `Managed` versão deste exemplo deve ser implantado no Azure. Siga as orientações na [usar as identidades de gerenciado para recursos do Azure](#) seção.

Para obter mais informações sobre como configurar um aplicativo de exemplo usando diretivas de pré-processador (`#define`), consulte [Introdução ao ASP.NET Core](#).

## Armazenamento secreto no ambiente de desenvolvimento

Definir segredos localmente usando o [ferramenta Secret Manager](#). Quando o aplicativo de exemplo é executado no computador local no ambiente de desenvolvimento, os segredos são carregados do repositório local Secret Manager.

A ferramenta Secret Manager exige um `<UserSecretsId>` propriedade no arquivo de projeto do aplicativo. Defina o valor da propriedade (`{GUID}`) para o GUID exclusivo:

```
<PropertyGroup>
  <UserSecretsId>{GUID}</UserSecretsId>
</PropertyGroup>
```

Segredos são criados como pares nome-valor. Valores hierárquicos (seções de configuração) usam um `:` (dois-pontos) como um separador no [configuração do ASP.NET Core](#) nomes de chave.

O Gerenciador de segredo é usado em um shell de comando aberto para a raiz do conteúdo do projeto, onde `{SECRET NAME}` é o nome e `{SECRET VALUE}` é o valor:

```
dotnet user-secrets set "{SECRET NAME}" "{SECRET VALUE}"
```

Execute os seguintes comandos em um shell de comando na raiz do conteúdo do projeto para definir os segredos para o aplicativo de exemplo:

```
dotnet user-secrets set "SecretName" "secret_value_1_dev"
dotnet user-secrets set "Section:SecretName" "secret_value_2_dev"
```

Quando esses segredos são armazenados no Azure Key Vault na [armazenamento secreto no ambiente de produção com o Azure Key Vault](#) seção, o `_dev` sufixo é alterado para `_prod`. O sufixo fornece uma indicação visual na saída do aplicativo que indica a origem dos valores de configuração.

## Armazenamento secreto no ambiente de produção com o Azure Key Vault

As instruções fornecidas pelo [guia de início rápido: Definir e recuperar um segredo do Azure Key Vault usando a CLI do Azure](#) tópico são resumidas aqui para criar um Azure Key Vault e armazenar segredos usados pelo aplicativo de exemplo. Consulte o tópico para obter mais detalhes.

1. Abra Azure Cloud shell usando qualquer um dos métodos a seguir na [portal do Azure](#):

- Selecione **Experimente** no canto superior direito de um bloco de código. Use a cadeia de caracteres de pesquisa "Da CLI do Azure" na caixa de texto.
- Abra o Cloud Shell no navegador com o **iniciar Cloud Shell** botão.
- Selecione o **Cloud Shell** botão no menu no canto superior direito do portal do Azure.

Para obter mais informações, consulte [Interface de linha de comando \(CLI do Azure\)](#) e [visão geral do Azure Cloud Shell](#).

2. Se você já não estiver autenticado, entrar com o `az login` comando.

3. Criar um grupo de recursos com o comando a seguir, onde `{RESOURCE GROUP NAME}` é o nome do grupo

de recursos para o novo grupo de recursos e {LOCATION} é a região do Azure (datacenter):

```
az group create --name "{RESOURCE GROUP NAME}" --location {LOCATION}
```

4. Criar um cofre de chaves no grupo de recursos com o comando a seguir, onde {KEY VAULT NAME} é o nome para o novo cofre de chaves e {LOCATION} é a região do Azure (datacenter):

```
az keyvault create --name "{KEY VAULT NAME}" --resource-group "{RESOURCE GROUP NAME}" --location {LOCATION}
```

5. Crie os segredos no cofre de chaves como pares nome-valor.

Nomes de segredos de Cofre de chaves do Azure são limitados a caracteres alfanuméricos e traços. Usam valores hierárquicos (seções de configuração) -- (dois traços) como um separador. Dois-pontos, que normalmente são usadas para delimitar uma seção de uma subchave no [configuração do ASP.NET Core](#), não são permitidos em nomes de segredos do Cofre de chaves. Portanto, os dois traços são usados e trocados para dois-pontos quando os segredos são carregados para a configuração do aplicativo.

Os segredos a seguir são para uso com o aplicativo de exemplo. Os valores incluem uma \_prod sufixo para distingui-los da \_dev sufixo valores carregados no ambiente de desenvolvimento de segredos do usuário. Substitua {KEY VAULT NAME} com o nome do Cofre de chaves que você criou na etapa anterior:

```
az keyvault secret set --vault-name "{KEY VAULT NAME}" --name "SecretName" --value "secret_value_1_prod"  
az keyvault secret set --vault-name "{KEY VAULT NAME}" --name "Section--SecretName" --value "secret_value_2_prod"
```

## Use a ID do aplicativo e segredo do cliente para aplicativos não hospedados do Azure

Configurar o Azure AD, Azure Key Vault e o aplicativo para usar uma ID do aplicativo e a senha (segredo do cliente) para autenticar para um cofre de chaves **quando o aplicativo está hospedado fora do Azure**.

### NOTE

Embora haja suporte para usar uma ID do aplicativo e a senha (segredo do cliente) para aplicativos hospedados no Azure, recomendamos o uso [identidades para recursos do Azure gerenciadas](#) ao hospedar um aplicativo no Azure. Identidades gerenciadas exigem armazenamento de credenciais no aplicativo ou sua configuração, portanto, ele é considerado como uma abordagem mais segura em geral.

O aplicativo de exemplo usa uma ID do aplicativo e a senha (segredo do cliente) quando o #define instrução na parte superior dos *Program.cs* arquivo é definido como Basic.

1. Registrar o aplicativo com o Azure AD e estabeleça uma senha (segredo do cliente) para a identidade do aplicativo.
2. Store o nome do Cofre de chaves, a ID do aplicativo e a senha/segredo do cliente do aplicativo *appSettings.json* arquivo.
3. Navegue até **cofres de chaves** no portal do Azure.
4. Selecione o Cofre de chaves que você criou na [armazenamento secreto no ambiente de produção com o Azure Key Vault](#) seção.
5. Selecione **políticas de acesso**.

6. Selecione **adicionar novo**.
7. Selecione **selecionar entidade** e selecione o aplicativo registrado por nome. Selecione o **selecionar** botão.
8. Abra **permissões do segredo** e forneça o aplicativo com **obter** e **lista** permissões.
9. Selecione **OK**.
10. Selecione **Salvar**.
11. Implante o aplicativo.

O `Basic` aplicativo de exemplo obtém seus valores de configuração de `IConfigurationRoot` com o mesmo nome que o nome do segredo:

- Valores não são hierárquicos: O valor para `SecretName` é obtido com `config["SecretName"]`.
- Valores hierárquicos (seções): Use `:` notação (dois-pontos) ou o `GetSection` método de extensão. Use qualquer uma dessas abordagens para obter o valor de configuração:
  - `config["Section:SecretName"]`
  - `config.GetSection("Section")["SecretName"]`

O aplicativo chama `AddAzureKeyVault` com os valores fornecidos pelo *appSettings.json* arquivo:

```
// using Microsoft.Extensions.Configuration;

public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .ConfigureAppConfiguration((context, config) =>
    {
        if (context.HostingEnvironment.IsProduction())
        {
            var builtConfig = config.Build();

            config.AddAzureKeyVault(
                $"https://{{builtConfig["KeyVaultName"]}}.vault.azure.net/",
                builtConfig["AzureADApplicationId"],
                builtConfig["AzureADPassword"]);
        }
    })
    .UseStartup<Startup>();
}
```

Valores de exemplo:

- Nome do Cofre de chaves: `contosovault`
- ID do aplicativo: `627e911e-43cc-61d4-992e-12db9c81b413`
- Senha: `g58K3dtg59o1Pa+e59v2Tx829w6VxTB2yv9sv/101di=`

*appsettings.json*:

```
{
  "KeyVaultName": "Key Vault Name",
  "AzureADApplicationId": "Azure AD Application ID",
  "AzureADPassword": "Azure AD Password/Client Secret"
}
```

Quando você executa o aplicativo, uma página da Web mostra os valores secretos carregados. No ambiente de desenvolvimento, os valores secretos carregar com o `_dev` sufixo. No ambiente de produção, os valores de carga com o `_prod` sufixo.

## Usar identidades de gerenciado para recursos do Azure

**Um aplicativo implantado no Azure** podem aproveitar [identidades para recursos do Azure gerenciadas](#), que permite que o aplicativo autenticar com o Azure Key Vault usando a autenticação do Azure AD sem credenciais (ID do aplicativo e Segredo Password/Client) armazenados no aplicativo.

O aplicativo de exemplo usa identidades de gerenciado para recursos do Azure quando o `#define` instrução na parte superior dos *Program.cs* arquivo é definido como `Managed`.

Insira o nome do cofre para o aplicativo *appSettings.json* arquivo. O aplicativo de exemplo não exige uma ID do aplicativo e a senha (segredo do cliente) quando definido como o `Managed` versão, portanto, você pode ignorar essas entradas de configuração. O aplicativo é implantado no Azure e o Azure autentica o aplicativo para acessar o Azure Key Vault usando apenas o nome do cofre armazenada em do *appSettings.json* arquivo.

Implante o aplicativo de exemplo no serviço de aplicativo do Azure.

Um aplicativo implantado no serviço de aplicativo do Azure é registrado automaticamente com o Azure AD quando o serviço é criado. Obter a ID de objeto de implantação para uso no comando a seguir. A ID de objeto é mostrada no portal do Azure na **identidade** painel do serviço de aplicativo.

Usando a CLI do Azure e a ID de objeto do aplicativo, forneça o aplicativo com `list` e `get` permissões para acessar o Cofre de chaves:

```
az keyvault set-policy --name '{KEY VAULT NAME}' --object-id {OBJECT ID} --secret-permissions get list
```

**Reinic peace o aplicativo** usando o portal do Azure, PowerShell ou CLI do Azure.

O aplicativo de exemplo:

- Cria uma instância do `AzureServiceTokenProvider` classe sem uma cadeia de caracteres de conexão. Quando uma cadeia de caracteres de conexão não for fornecida, o provedor tenta obter um token de acesso de identidades de gerenciado para recursos do Azure.
- Uma nova `KeyVaultClient` é criado com o `AzureServiceTokenProvider` token retorno de chamada de instância.
- O `KeyVaultClient` instância é usada com uma implementação padrão de `IKeyVaultSecretManager` que carrega todos os valores do segredo e substitui os dois traços (`--`) com dois-pontos (`:`) em nomes de chave.

```

// using Microsoft.Azure.KeyVault;
// using Microsoft.Azure.Services.AppAuthentication;
// using Microsoft.Extensions.Configuration.AzureKeyVault;

public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .ConfigureAppConfiguration((context, config) =>
    {
        if (context.HostingEnvironment.IsProduction())
        {
            var builtConfig = config.Build();

            var azureServiceTokenProvider = new AzureServiceTokenProvider();
            var keyVaultClient = new KeyVaultClient(
                new KeyVaultClient.AuthenticationCallback(
                    azureServiceTokenProvider.KeyVaultTokenCallback));

            config.AddAzureKeyVault(
                $"https://{builtConfig["KeyVaultName"]}.vault.azure.net/",
                keyVaultClient,
                new DefaultKeyVaultSecretManager());
        }
    })
    .UseStartup<Startup>();

```

Quando você executa o aplicativo, uma página da Web mostra os valores secretos carregados. No ambiente de desenvolvimento, os valores do segredo têm o `_dev` sufixo porque elas são fornecidas por segredos do usuário. No ambiente de produção, os valores de carga com o `_prod` sufixo porque elas são fornecidas pelo Azure Key Vault.

Se você receber um `Access denied` erro, confirme se o aplicativo está registrado com o Azure AD e recebe acesso ao Cofre de chaves. Confirme que você tiver reiniciado o serviço no Azure.

## Usar um prefixo de nome da chave

`AddAzureKeyVault` Fornece uma sobrecarga que aceita uma implementação de `IKeyVaultSecretManager`, que permite que você controle como os principais segredos do cofre são convertidas em chaves de configuração. Por exemplo, você pode implementar a interface para carregar os valores do segredo com base em um valor de prefixo que você fornece na inicialização do aplicativo. Isso permite que você, por exemplo, para carregar segredos de acordo com a versão do aplicativo.

### WARNING

Não usar prefixos de segredos do Cofre de chaves para colocar segredos para vários aplicativos no mesmo Cofre de chaves ou para colocar segredos ambientais (por exemplo, *development* versus *produção* segredos) no mesmo cofre. É recomendável que diferentes aplicativos e ambientes de desenvolvimento/produção usam cofres de chaves separados para isolar os ambientes de aplicativo para o nível mais alto de segurança.

No exemplo a seguir, um segredo é estabelecido na chave do cofre (e usar a ferramenta Secret Manager no ambiente de desenvolvimento) para `5000-AppSecret` (períodos não são permitidos em nomes de segredos do Cofre de chaves). Esse segredo representa um segredo do aplicativo para versão Version=5.0.0.0 do aplicativo. Para outra versão do aplicativo, 5.1.0.0, um segredo é adicionado à chave de cofre (e usar a ferramenta Secret Manager) para `5100-AppSecret`. Cada versão do aplicativo carrega seu valor com controle de versão do segredo em sua configuração como `AppSecret`, remoção desativar a versão ele carrega o segredo.

`AddAzureKeyVault` é chamado com um personalizado `IKeyVaultSecretManager`:

```

// using Microsoft.Extensions.Configuration;

public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .ConfigureAppConfiguration((context, config) =>
    {
        if (context.HostingEnvironment.IsProduction())
        {
            // The appVersion obtains the app version (5.0.0.0), which
            // is set in the project file and obtained from the entry
            // assembly. The versionPrefix holds the version without
            // dot notation for the PrefixKeyVaultSecretManager.
            var appVersion = Assembly.GetEntryAssembly().GetName().Version.ToString();
            var versionPrefix = appVersion.Replace(".", string.Empty);

            var builtConfig = config.Build();

            config.AddAzureKeyVault(
                $"https://{{builtConfig["KeyVaultName"]}}.vault.azure.net/",
                builtConfig["AzureADApplicationId"],
                builtConfig["AzureADPassword"],
                new PrefixKeyVaultSecretManager(versionPrefix));
        }
    })
    .UseStartup<Startup>();

```

Os valores para nome do Cofre de chaves, a ID do aplicativo e a senha (segredo do cliente) são fornecidos pela *appSettings.json* arquivo:

```
{
    "KeyVaultName": "Key Vault Name",
    "AzureADApplicationId": "Azure AD Application ID",
    "AzureADPassword": "Azure AD Password/Client Secret"
}
```

Valores de exemplo:

- Nome do Cofre de chaves: `contosovault`
- ID do aplicativo: `627e911e-43cc-61d4-992e-12db9c81b413`
- Senha: `g58K3dtg59o1Pa+e59v2Tx829w6VxTB2yv9sv/101di=`

O `IKeyVaultSecretManager` implementação reage aos prefixos de segredos para carregar o segredo apropriado na configuração de versão:

```

public class PrefixKeyVaultSecretManager : IKeyVaultSecretManager
{
    private readonly string _prefix;

    public PrefixKeyVaultSecretManager(string prefix)
    {
        _prefix = $"{prefix}--";
    }

    public bool Load(SecretItem secret)
    {
        // Load a vault secret when its secret name starts with the
        // prefix. Other secrets won't be loaded.
        return secret.Identifier.Name.StartsWith(_prefix);
    }

    public string GetKey(SecretBundle secret)
    {
        // Remove the prefix from the secret name and replace two
        // dashes in any name with the KeyDelimiter, which is the
        // delimiter used in configuration (usually a colon). Azure
        // Key Vault doesn't allow a colon in secret names.
        return secret.SecretIdentifier.Name
            .Substring(_prefix.Length)
            .Replace("--", ConfigurationPath.KeyDelimiter);
    }
}

```

O `Load` método é chamado por um algoritmo de provedor que itera os segredos do cofre para encontrar aqueles que têm o prefixo de versão. Quando um prefixo de versão é encontrado com `Load`, o algoritmo usa o `GetKey` método para retornar o nome da configuração do nome do segredo. Ele ignora o prefixo de versão do nome do segredo e retorna o restante do nome do segredo para carregamento na configuração do aplicativo pares nome-valor.

Quando esse método é implementado:

1. A versão do aplicativo especificada no arquivo de projeto do aplicativo. No exemplo a seguir, a versão do aplicativo é definida como `5.0.0.0`:

```

<PropertyGroup>
    <Version>5.0.0.0</Version>
</PropertyGroup>

```

2. Confirme se um `<UserSecretsId>` propriedade estiver presente no arquivo de projeto do aplicativo, onde `{GUID}` é um GUID fornecido pelo usuário:

```

<PropertyGroup>
    <UserSecretsId>{GUID}</UserSecretsId>
</PropertyGroup>

```

Salvar os segredos seguir localmente com o [ferramenta Secret Manager](#):

```

dotnet user-secrets set "5000-AppSecret" "5.0.0.0_secret_value_dev"
dotnet user-secrets set "5100-AppSecret" "5.1.0.0_secret_value_dev"

```

3. Segredos são salvos no Azure Key Vault usando os seguintes comandos de CLI do Azure:

```
az keyvault secret set --vault-name "{KEY VAULT NAME}" --name "5000-AppSecret" --value "5.0.0.0_secret_value_prod"
az keyvault secret set --vault-name "{KEY VAULT NAME}" --name "5100-AppSecret" --value "5.1.0.0_secret_value_prod"
```

4. Quando o aplicativo é executado, os segredos do Cofre de chaves são carregados. O segredo de cadeia de caracteres para `5000-AppSecret` corresponde à versão do aplicativo especificada no arquivo de projeto do aplicativo (`5.0.0.0`).
5. A versão `5000` (com o traço) é removida do nome de chave. Em todo o aplicativo, lendo a configuração com a chave `AppSecret` carrega o valor do segredo.
6. Se a versão do aplicativo é alterada no arquivo de projeto para `5.1.0.0` e o aplicativo é executado novamente, é o valor do segredo retornado `5.1.0.0_secret_value_dev` no ambiente de desenvolvimento e `5.1.0.0_secret_value_prod` em produção.

#### NOTE

Você também pode fornecer seus próprios `KeyVaultClient` implementação para `AddAzureKeyVault`. Compartilhando uma única instância do cliente entre o aplicativo permite que um cliente personalizado.

## Autenticar no Azure Key Vault com um certificado X.509

Ao desenvolver um aplicativo do .NET Framework em um ambiente que dá suporte a certificados, você pode autenticar para o Azure Key Vault com um certificado X.509. Chave privada do certificado X.509 é gerenciado pelo sistema operacional. Para obter mais informações, consulte [autenticar com um certificado em vez de um segredo do cliente](#). Use o `AddAzureKeyVault` sobrecarga que aceita um `X509Certificate2` (`_env` no exemplo a seguir):

```
var builtConfig = config.Build();

var store = new X509Store(StoreLocation.CurrentUser);
store.Open(OpenFlags.ReadOnly);
var cert = store.Certificates
    .Find(X509FindType.FindByThumbprint,
        config["CertificateThumbprint"], false);

config.AddAzureKeyVault(
    builtConfig["KeyVaultName"],
    builtConfig["AzureADApplicationId"],
    cert.OfType<X509Certificate2>().Single(),
    new EnvironmentSecretManager(context.HostingEnvironment.ApplicationName));

store.Close();
```

## Associar uma matriz a uma classe

O provedor é capaz de ler os valores de configuração em uma matriz para a associação para uma matriz POCO.

Durante a leitura de uma fonte de configuração que permite que as chaves conter dois-pontos (`:`) separadores, um segmento de chave numérico é usado para distinguir as chaves que compõem uma matriz (`:0:, :1:, ... :{n}:` ). Para obter mais informações, consulte [configuração: Associar uma matriz a uma classe](#).

Chaves do Azure Key Vault não podem usar dois-pontos como um separador. A abordagem descrita neste

tópico usa double traços ( `--` ) como separador de valores hierárquicos (seções). As chaves de matriz são armazenadas no Azure Key Vault com double traços e segmentos de chave numéricos ( `--0--` , `--1--` ,... `--{n}--` ).

Examine o seguinte [Serilog](#) fornecida por um arquivo JSON de configuração do provedor de log. Há dois objetos literais definidos na `WriteTo` matriz que refletem os dois Serilog *coletores*, que descrevem os destinos para saída de log:

```
"Serilog": {  
    "WriteTo": [  
        {  
            "Name": "AzureTableStorage",  
            "Args": {  
                "storageTableName": "logs",  
                "connectionString": "DefaultEnd...ountKey=Eby8...GMGw=="  
            }  
        },  
        {  
            "Name": "AzureDocumentDB",  
            "Args": {  
                "endpointUrl": "https://contoso.documents.azure.com:443",  
                "authorizationKey": "Eby8...GMGw=="  
            }  
        }  
    ]  
}
```

A configuração mostrada no arquivo JSON anterior é armazenada no Azure Key Vault usando o traço duplo ( `--` ) segmentos notação e numérico:

CHAVE	VALOR
<code>Serilog--WriteTo--0--Name</code>	<code>AzureTableStorage</code>
<code>Serilog--WriteTo--0--Args--storageTableName</code>	<code>logs</code>
<code>Serilog--WriteTo--0--Args--connectionString</code>	<code>DefaultEnd...ountKey=Eby8...GMGw==</code>
<code>Serilog--WriteTo--1--Name</code>	<code>AzureDocumentDB</code>
<code>Serilog--WriteTo--1--Args--endpointUrl</code>	<code>https://contoso.documents.azure.com:443</code>
<code>Serilog--WriteTo--1--Args--authorizationKey</code>	<code>Eby8...GMGw==</code>

## Recarregue os segredos

Segredos são armazenados em cache até `IConfigurationRoot.Reload()` é chamado. Expirado, desabilitado, e atualizados segredos no cofre de chaves não serão respeitados pelo aplicativo até `Reload` é executado.

```
Configuration.Reload();
```

## Segredos desabilitados e expirados

Segredos desabilitados e expirados lançar um `KeyVaultClientException`. Para impedir que seu aplicativo aione, substitua o seu aplicativo ou atualizar o segredo desabilitado/expirado.

## Solução de problemas

Quando o aplicativo falha ao carregar a configuração usando o provedor, uma mensagem de erro é gravada para o [infra-estrutura de log do ASP.NET Core](#). As seguintes condições impedirá que a configuração de carregamento:

- O aplicativo não está configurado corretamente no Azure Active Directory.
- O Cofre de chaves não existe no Azure Key Vault.
- O aplicativo não está autorizado a acessar o Cofre de chaves.
- A política de acesso não inclui `Get` e `List` permissões.
- No cofre de chaves, os dados de configuração (par nome-valor) estão nomeados incorretamente, ausente, desabilitado ou expirado.
- O aplicativo tem o nome do cofre da chave incorreta (`KeyVaultName`), Id de aplicativo do Azure AD (`AzureADApplicationId`), ou a senha (segredo do cliente) do Azure AD (`AzureADPassword`).
- A senha (segredo do cliente) do Azure AD (`AzureADPassword`) expirou.
- A chave de configuração (nome) está incorreta no aplicativo para o valor que você está tentando carregar.

## Recursos adicionais

- [Configuração no ASP.NET Core](#)
- [Microsoft Azure: Cofre de chaves](#)
- [Microsoft Azure: Documentação do Key Vault](#)
- [Como gerar e transferir protegida por HSM chaves para o Azure Key Vault](#)
- [Classe KeyVaultClient](#)

# Ataques de evitar entre solicitação intersite forjada (CSRF/XSRF) no ASP.NET Core

02/02/2019 • 27 minutes to read • [Edit Online](#)

Por [Steve Smith](#), [Fiyaz Hasan](#), e [Rick Anderson](#)

Falsificação de solicitação intersite forjada (também conhecida como XSRF ou CSRF, pronunciado *surfe consulte*) é um ataque contra aplicativos hospedados na web, no qual um aplicativo web mal-intencionado pode influenciar a interação entre um navegador cliente e um aplicativo web que confia que Navegador. Esses ataques são possíveis, pois navegadores da web enviam alguns tipos de tokens de autenticação automaticamente com cada solicitação de um site. Essa forma de exploração é também conhecido como um *ataque de um clique* ou *sequestro de sessão* porque o ataque aproveita o usuário do autenticado anteriormente sessão.

Um exemplo de um ataque CSRF:

1. Um usuário entra em `www.good-banking-site.com` usando a autenticação de formulários. O servidor autentica o usuário e emite uma resposta que inclui um cookie de autenticação. O site é vulnerável a ataques porque ele confia qualquer solicitação recebida com um cookie de autenticação válido.
2. O usuário visitar um site mal-intencionado, `www.bad-crook-site.com`.

O site mal-intencionado, `www.bad-crook-site.com`, contém um formulário HTML semelhante ao seguinte:

```
<h1>Congratulations! You're a Winner!</h1>
<form action="http://good-banking-site.com/api/account" method="post">
    <input type="hidden" name="Transaction" value="withdraw">
    <input type="hidden" name="Amount" value="1000000">
    <input type="submit" value="Click to collect your prize!">
</form>
```

Observe que o formulário `action` postagens para o site vulnerável, não para o site mal-intencionado. Essa é a parte de "site cruzado" de CSRF.

3. O usuário seleciona o botão Enviar. O navegador faz a solicitação e inclui automaticamente o cookie de autenticação para o domínio solicitado, `www.good-banking-site.com`.
4. A solicitação é executada `www.good-banking-site.com` server com o contexto de autenticação do usuário e pode executar qualquer ação que um usuário autenticado tem permissão para executar.

Além do cenário em que o usuário seleciona o botão para enviar o formulário, o site mal-intencionado pode:

- Execute um script que envia automaticamente o formulário.
- Envie o envio do formulário como uma solicitação AJAX.
- Oculte o formulário usando CSS.

Esses cenários alternativos não exigem qualquer entrada do usuário que não sejam inicialmente visitar o site mal-intencionado ou ação.

Usar o HTTPS não impede que um ataque CSRF. O site mal-intencionado pode enviar um

<https://www.good-banking-site.com/> solicitar tão fácil quanto ele pode enviar uma solicitação não segura.

Alguns ataques são direcionados a pontos de extremidade que respondem às solicitações GET, neste caso, uma marca de imagem pode ser usada para executar a ação. Essa forma de ataque é comum nos sites de Fórum que permitem imagens, mas bloqueiam o JavaScript. Aplicativos que alteram o estado em solicitações GET, em que as variáveis ou os recursos são alterados, são vulneráveis a ataques mal-intencionados. **Solicitações GET que alteram o estado são inseguras. Uma prática recomendada é nunca alterar o estado em uma solicitação GET.**

Ataques de CSRF são possíveis em relação a aplicativos web que usam cookies para autenticação porque:

- Os navegadores armazenam os cookies emitidos por um aplicativo web.
- Armazenado cookies incluem cookies de sessão para usuários autenticados.
- Os navegadores enviam a que todos os cookies associados com um domínio para o aplicativo web cada solicitação, independentemente de como a solicitação para o aplicativo foi gerada dentro do navegador.

No entanto, os ataques de CSRF não estão limitadas a exploração de cookies. Por exemplo, a autenticação básica e Digest também são vulneráveis. Depois que um usuário entra com a autenticação básica ou Digest, o navegador automaticamente envia as credenciais até que a sessão<sup>†</sup> termina.

<sup>†</sup>Nesse contexto, sessão refere-se à sessão do lado do cliente durante o qual o usuário é autenticado. Ele é não relacionada a sessões do lado do servidor ou [Middleware de sessão do ASP.NET Core](#).

Os usuários podem se proteger contra vulnerabilidades CSRF tomando precauções:

- Entrar em aplicativos web quando terminar de usá-los.
- Limpar os cookies do navegador periodicamente.

No entanto, as vulnerabilidades CSRF são basicamente um problema com o aplicativo web, não pelo usuário final.

## Conceitos básicos de autenticação

Autenticação baseada em cookie é uma forma popular de autenticação. Sistemas de autenticação baseada em token estão crescendo em termos de popularidade, especialmente para aplicativos de página única (SPAs).

### Autenticação baseada em cookie

Quando um usuário é autenticado usando o nome de usuário e senha, elas emitidas um token, que contém um tíquete de autenticação que pode ser usado para autenticação e autorização. O token é armazenado como um cookie que acompanha cada solicitação, o cliente faz. Gerar e validar esse cookie é realizada pelo Middleware de autenticação de Cookie. O [middleware](#) serializa uma entidade de usuário em um cookie criptografado. Nas próximas solicitações, o middleware valida o cookie, recria a entidade de segurança e a atribui a entidade de segurança de [usuário](#) propriedade de [HttpContext](#).

### Autenticação baseada em token

Quando um usuário é autenticado, elas emitidas um token (não um token antifalsificação). O token contém informações de usuário na forma de [declarações](#) ou um token de referência que aponta o aplicativo para o estado do usuário mantido no aplicativo. Quando um usuário tenta acessar um recurso que requer autenticação, o token é enviado para o aplicativo com um cabeçalho de autorização adicionais na forma de token de portador. Isso torna o aplicativo sem monitoração de estado. Em cada solicitação subsequente, o token é passado na solicitação para a validação do lado do servidor. Não é esse token *criptografado*; ele tem *codificado*. No servidor, o token é decodificado para acessar suas informações. Para enviar o token em solicitações subsequentes, armazene o token no armazenamento local do navegador. Não fique preocupado sobre vulnerabilidade CSRF, se o token é armazenado no armazenamento local do navegador. CSRF é uma preocupação quando o token é armazenado em um cookie.

## Vários aplicativos hospedados em um domínio

Ambientes de hospedagem compartilhados são vulneráveis a sequestro de sessão, login CSRF e outros ataques.

Embora `example1.contoso.net` e `example2.contoso.net` são hosts diferentes, há uma relação de confiança implícita entre os hosts sob o `*.contoso.net` domínio. Essa relação de confiança implícita permite que os hosts potencialmente não confiáveis afetam uns dos outros cookies (as políticas de mesma origem que regem as solicitações AJAX não necessariamente se aplicam aos cookies HTTP).

Ataques que exploram confiáveis cookies entre aplicativos hospedados no mesmo domínio podem ser evitadas por meio do compartilhamento não domínios. Quando cada aplicativo é hospedado em seu próprio domínio, não há nenhuma relação de confiança implícita de cookie para explorar.

## Configuração antifalsificação do ASP.NET Core

### WARNING

ASP.NET Core implementa usando antifalsificação [proteção de dados do ASP.NET Core](#). A pilha da proteção de dados deve ser configurada para funcionar em um farm de servidores. Ver [Configurando a proteção de dados](#) para obter mais informações.

No ASP.NET Core 2.0 ou posterior, o [FormTagHelper](#) injeta tokens antifalsificação em elementos de formulário HTML. A marcação a seguir em um arquivo Razor gera automaticamente os tokens antifalsificação:

```
<form method="post">
  ...
</form>
```

Similarly, [IHtmlHelper.BeginForm](#) gera tokens antifalsificação por padrão, se o método do formulário não é GET.

A geração automática de tokens antifalsificação para elementos de formulário HTML acontece quando o `<form>` marca contém o `method="post"` atributo e uma das opções a seguir forem verdadeira:

- O atributo `action` está vazio (`action=""`).
- O atributo de ação não for fornecido (`<form method="post">`).

Geração automática de tokens antifalsificação para elementos de formulário HTML pode ser desabilitada:

- Desabilitar explicitamente tokens antifalsificação com o `asp-antiforgery` atributo:

```
<form method="post" asp-antiforgery="false">
  ...
</form>
```

- O elemento de formulário é aceito-out dos auxiliares de marca usando o auxiliar de marca [! recusar símbolo](#):

```
<!form method="post">
  ...
</!form>
```

- Remover o `FormTagHelper` da exibição. O `FormTagHelper` pode ser removido de uma exibição,

adicinando a seguinte diretiva para o modo de exibição do Razor:

```
@removeTagHelper Microsoft.AspNetCore.Mvc.TagHelpers.FormTagHelper,  
Microsoft.AspNetCore.Mvc.TagHelpers
```

#### NOTE

Páginas do Razor são protegidos automaticamente contra XSRF/CSRF. Para obter mais informações, consulte [XSRF/CSRF e páginas Razor](#).

A abordagem mais comum para proteger contra ataques CSRF é usar o *padrão de Token do sincronizador* (STP). STP é usado quando o usuário solicita uma página com os dados de formulário:

1. O servidor envia um token associado com a atual identidade do usuário para o cliente.
2. O cliente envia o token para o servidor para verificação.
3. Se o servidor recebe um token que não corresponde à identidade do usuário autenticado, a solicitação será rejeitada.

O token é exclusivo e imprevisíveis. O token também pode ser usado para garantir que o sequenciamento adequado de uma série de solicitações (por exemplo, garantindo a sequência de solicitação de: a página 1 – página 2 – página 3). Todos os formulários em modelos do ASP.NET Core MVC e páginas do Razor geram tokens contra falsificação. O seguinte par de exemplos do modo de exibição gera tokens contra falsificação:

```
<form asp-controller="Manage" asp-action="ChangePassword" method="post">  
    ...  
</form>  
  
@using (Html.BeginForm("ChangePassword", "Manage"))  
{  
    ...  
}
```

Adicionar explicitamente um token antifalsificação para um `<form>` elemento sem o uso de auxiliares de marca com o auxiliar HTML [@Html.AntiForgeryToken](#):

```
<form action="/" method="post">  
    @Html.AntiForgeryToken()  
</form>
```

Em cada um dos casos anteriores, o ASP.NET Core adiciona um campo de formulário oculto semelhante ao seguinte:

```
<input name="__RequestVerificationToken" type="hidden" value="CfDJ8NrAkS ... s2-m9Yw">
```

ASP.NET Core inclui três [filtros](#) para trabalhar com tokens antifalsificação:

- [ValidateAntiForgeryFilter](#)
- [AutoValidateAntiforgeryFilter](#)
- [IgnoreAntiforgeryFilter](#)

## Opções de antifalsificação

Personalize [opções antifalsificação](#) em `Startup.ConfigureServices`:

```

services.AddAntiforgery(options =>
{
    // Set Cookie properties using CookieBuilder properties†.
    options.FormFieldName = "AntiforgeryFieldname";
    options.HeaderName = "X-CSRF-TOKEN-HEADERNAME";
    options.SuppressXFrameOptionsHeader = false;
});

```

<sup>†</sup>Definir a antifalsificação `Cookie` usando as propriedades do objeto de propriedades de `CookieBuilder` classe.

OPÇÃO	DESCRIÇÃO
<code>Cookie</code>	Determina as configurações usadas para criar o cookie antifalsificação.
<code>FormFieldName</code>	O nome do campo de formulário oculto usado pelo sistema antifalsificação para processar tokens antifalsificação nos modos de exibição.
<code>HeaderName</code>	O nome do cabeçalho usado pelo sistema antifalsificação. Se <code>null</code> , o sistema considera apenas os dados de formulário.
<code>SuppressXFrameOptionsHeader</code>	Especifica se deve suprimir a geração do <code>X-Frame-Options</code> cabeçalho. Por padrão, o cabeçalho é gerado com um valor de "SAMEORIGIN". Assume o padrão de <code>false</code> .

```

services.AddAntiforgery(options =>
{
    options.CookieDomain = "contoso.com";
    options.CookieName = "X-CSRF-TOKEN-COOKIEENAME";
    options.CookiePath = "Path";
    options.FormFieldName = "AntiforgeryFieldname";
    options.HeaderName = "X-CSRF-TOKEN-HEADERNAME";
    options.RequireSsl = false;
    options.SuppressXFrameOptionsHeader = false;
});

```

OPÇÃO	DESCRIÇÃO
<code>Cookie</code>	Determina as configurações usadas para criar o cookie antifalsificação.
<code>CookieDomain</code>	O domínio do cookie. Assume o padrão de <code>null</code> . Essa propriedade está obsoleta e será removida em uma versão futura. A alternativa recomendada é <code>Cookie.Domain</code> .
<code>CookieName</code>	O nome do cookie. Se não definido, o sistema gera um nome exclusivo que começa com o <code>DefaultCookiePrefix</code> ("AspNetCore.Antiforgery."). Essa propriedade está obsoleta e será removida em uma versão futura. A alternativa recomendada é <code>Cookie.Name</code> .

OPÇÃO	DESCRIÇÃO
<a href="#">CookiePath</a>	O caminho definido no cookie. Essa propriedade está obsoleta e será removida em uma versão futura. A alternativa recomendada é <code>Cookie.Path</code> .
<a href="#">FormFieldName</a>	O nome do campo de formulário oculto usado pelo sistema antifalsificação para processar tokens antifalsificação nos modos de exibição.
<a href="#">HeaderName</a>	O nome do cabeçalho usado pelo sistema antifalsificação. Se <code>null</code> , o sistema considera apenas os dados de formulário.
<a href="#">RequireSsl</a>	Especifica se o HTTPS é exigido pelo sistema antifalsificação. Se <code>true</code> , não HTTPS solicitações falham. Assume o padrão de <code>false</code> . Essa propriedade está obsoleta e será removida em uma versão futura. A alternativa recomendada é definir <code>Cookie.SecurePolicy</code> .
<a href="#">SuppressXFrameOptionsHeader</a>	Especifica se deve suprimir a geração do <code>X-Frame-Options</code> cabeçalho. Por padrão, o cabeçalho é gerado com um valor de "SAMEORIGIN". Assume o padrão de <code>false</code> .

Para obter mais informações, consulte [CookieAuthenticationOptions](#).

## Configurar recursos antiforgery com `IAntiforgery`

`IAntiforgery` fornece a API para configurar recursos antifalsificação. `IAntiforgery` podem ser solicitadas a `Configure` método da `Startup` classe. O exemplo a seguir usa o middleware da home page do aplicativo para gerar um token antifalsificação e enviá-lo na resposta como um cookie (usando a convenção de nomenclatura Angular da padrão descrita mais adiante neste tópico):

```
public void Configure(IApplicationBuilder app, IAntiforgery antiforgery)
{
    app.Use(next => context =>
    {
        string path = context.Request.Path.Value;

        if (
            string.Equals(path, "/", StringComparison.OrdinalIgnoreCase) ||
            string.Equals(path, "/index.html", StringComparison.OrdinalIgnoreCase))
        {
            // The request token can be sent as a JavaScript-readable cookie,
            // and Angular uses it by default.
            var tokens = antiforgery.GetAndStoreTokens(context);
            context.Response.Cookies.Append("XSRF-TOKEN", tokens.RequestToken,
                new CookieOptions() { HttpOnly = false });
        }

        return next(context);
    });
}
```

### Exigir validação antifalsificação

`ValidateAntiForgeryToken` é um filtro de ação que pode ser aplicado a uma ação individual, um controlador ou globalmente. As solicitações feitas a ações que têm esse filtro aplicado são bloqueadas, a menos que a

solicitação inclui um token antifalsificação válido.

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> RemoveLogin(RemoveLoginViewModel account)
{
    ManageMessageId? message = ManageMessageId.Error;
    var user = await GetCurrentUserAsync();

    if (user != null)
    {
        var result =
            await _userManager.RemoveLoginAsync(
                user, account.LoginProvider, account.ProviderKey);

        if (result.Succeeded)
        {
            await _signInManager.SignInAsync(user, isPersistent: false);
            message = ManageMessageId.RemoveLoginSuccess;
        }
    }

    return RedirectToAction(nameof(ManageLogins), new { Message = message });
}
```

O `ValidateAntiForgeryToken` atributo requer um token para solicitações para os métodos de ação decora, incluindo as solicitações HTTP GET. Se o `ValidateAntiForgeryToken` atributo é aplicado em controladores do aplicativo, ele pode ser substituído com o `IgnoreAntiforgeryToken` atributo.

#### NOTE

ASP.NET Core não dá suporte a adição de tokens antifalsificação para solicitações GET automaticamente.

### Automaticamente validar tokens antifalsificação para métodos HTTP não seguros apenas

Aplicativos ASP.NET Core não geram tokens contra falsificação para métodos HTTP seguros (GET, HEAD, opções e rastreamento). Em vez de aplicar amplamente a `ValidateAntiForgeryToken` atributo e, em seguida, substituindo-o com `IgnoreAntiforgeryToken` atributos, o `AutoValidateAntiforgeryToken` atributo pode ser usado. Esse atributo funciona de maneira idêntica ao `ValidateAntiForgeryToken` de atributo, exceto que ele não exige tokens para solicitações feitas usando os seguintes métodos HTTP:

- OBTER
- HOME
- OPÇÕES
- TRACE

É recomendável o uso de `AutoValidateAntiforgeryToken` em larga escala para cenários de não-API. Isso garante que as ações de POSTAGEM são protegidas por padrão. A alternativa é ignorar tokens antifalsificação por padrão, a menos que `ValidateAntiForgeryToken` é aplicado a métodos de ação individual. Ele tem mais provavelmente neste cenário para um método de ação a ser deixado POST desprotegidos por engano, deixando o aplicativo vulnerável a ataques CSRF. Todas as postagens devem enviar o token antifalsificação.

APIs não têm um mecanismo automático para enviar a parte não-cookie do token. A implementação provavelmente depende a implementação de código do cliente. Alguns exemplos são mostrados abaixo:

Exemplo de nível de classe:

```
[Authorize]
[AutoValidateAntiforgeryToken]
public class ManageController : Controller
{
```

Exemplo global:

```
services.AddMvc(options =>
    options.Filters.Add(new AutoValidateAntiforgeryTokenAttribute()));
```

### Substituição global ou atributos antifalsificação do controlador

O [IgnoreAntiforgeryToken](#) filtro é usado para eliminar a necessidade de um token antifalsificação para uma determinada ação (ou controlador). Quando aplicado, esse filtro substitui [ValidateAntiForgeryToken](#) e [AutoValidateAntiforgeryToken](#) filtros especificados em um nível mais alto (globalmente ou em um controlador).

```
[Authorize]
[AutoValidateAntiforgeryToken]
public class ManageController : Controller
{
    [HttpPost]
    [IgnoreAntiforgeryToken]
    public async Task<IActionResult> DoSomethingSafe(SomeViewModel model)
    {
        // no antiforgery token required
    }
}
```

## Tokens de atualização após a autenticação

Tokens devem ser atualizados depois que o usuário é autenticado ao redirecionar o usuário para uma exibição ou página do Razor.

## JavaScript, AJAX e SPAs

Em aplicativos tradicionais baseados em HTML, tokens antifalsificação são passados para o servidor usando os campos de formulário oculto. Em aplicativos modernos baseados em JavaScript e SPAs, muitas solicitações são feitas por meio de programação. Essas solicitações AJAX podem usar outras técnicas (como cabeçalhos de solicitação ou cookies) para enviar o token.

Se os cookies são usados para armazenar tokens de autenticação e para autenticar solicitações de API no servidor, CSRF é um problema potencial. Se o armazenamento local é usado para armazenar o token, vulnerabilidade CSRF pode ser reduzida porque os valores do armazenamento local não são enviadas automaticamente para o servidor com cada solicitação. Dessa forma, usando o armazenamento local para armazenar o token antifalsificação no cliente e enviar o token como um cabeçalho de solicitação é uma abordagem recomendada.

### JavaScript

Usando o JavaScript com modos de exibição, o token pode ser criado usando um serviço de dentro da exibição. Injetar o [Microsoft.AspNetCore.Antiforgery.IAntiforgery](#) serviço para o modo de exibição e chame [GetAndStoreTokens](#):

```

@{
    ViewData["Title"] = "AJAX Demo";
}
@inject Microsoft.AspNetCore.Antiforgery.IAntiforgery Xsrf
@functions{
    public string GetAntiXsrfRequestToken()
    {
        return Xsrf.GetAndStoreTokens(Context).RequestToken;
    }
}

<input type="hidden" id="RequestVerificationToken"
       name="RequestVerificationToken" value="@GetAntiXsrfRequestToken()">

<h2>@ViewData["Title"].</h2>
<h3>@ViewData["Message"]</h3>

<div class="row">
    <p><input type="button" id="antiforgery" value="Antiforgery"></p>
    <script>
        var xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function() {
            if (xhttp.readyState == XMLHttpRequest.DONE) {
                if (xhttp.status == 200) {
                    alert(xhttp.responseText);
                } else {
                    alert('There was an error processing the AJAX request.');
                }
            }
        };
        document.addEventListener('DOMContentLoaded', function() {
            document.getElementById("antiforgery").onclick = function () {
                xhttp.open('POST', '@Url.Action("Antiforgery", "Home")', true);
                xhttp.setRequestHeader("RequestVerificationToken",
                    document.getElementById('RequestVerificationToken').value);
                xhttp.send();
            }
        });
    </script>
</div>

```

Essa abordagem elimina a necessidade de lidar diretamente com definir cookies a partir do servidor ou lê-los a partir do cliente.

O exemplo anterior usa JavaScript para ler o valor do campo oculto para o cabeçalho de POSTAGEM de AJAX.

JavaScript pode também tokens de acesso em cookies e usar o conteúdo do cookie para criar um cabeçalho com o valor do token.

```

context.Response.Cookies.Append("CSRF-TOKEN", tokens.RequestToken,
    new Microsoft.AspNetCore.Http.CookieOptions { HttpOnly = false });

```

Supondo que o script solicita para enviar o token em um cabeçalho chamado `X-CSRF-TOKEN`, configure o serviço antifalsificação para procurar o `X-CSRF-TOKEN` cabeçalho:

```

services.AddAntiforgery(options => options.HeaderName = "X-CSRF-TOKEN");

```

O exemplo a seguir usa JavaScript para fazer uma solicitação AJAX com o cabeçalho apropriado:

```

function getCookie(cname) {
    var name = cname + "=";
    var decodedCookie = decodeURIComponent(document.cookie);
    var ca = decodedCookie.split(';');
    for(var i = 0; i <ca.length; i++) {
        var c = ca[i];
        while (c.charAt(0) == ' ') {
            c = c.substring(1);
        }
        if (c.indexOf(name) == 0) {
            return c.substring(name.length, c.length);
        }
    }
    return "";
}

var csrfToken = getCookie("CSRF-TOKEN");

var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (xhttp.readyState == XMLHttpRequest.DONE) {
        if (xhttp.status == 200) {
            alert(xhttp.responseText);
        } else {
            alert('There was an error processing the AJAX request.');
        }
    }
};
xhttp.open('POST', '/api/password/changepassword', true);
xhttp.setRequestHeader("Content-type", "application/json");
xhttp.setRequestHeader("X-CSRF-TOKEN", csrfToken);
xhttp.send(JSON.stringify({ "newPassword": "ReallySecurePassword999$$" }));

```

## AngularJS

AngularJS usa uma convenção para endereço CSRF. Se o servidor envia um cookie com o nome `XSRF-TOKEN`, o AngularJS `$http` serviço adiciona o valor do cookie a um cabeçalho quando ele envia uma solicitação ao servidor. Esse processo é automático. O cabeçalho não precisa ser definido explicitamente no cliente. O nome do cabeçalho é `X-XSRF-TOKEN`. O servidor deve detectar esse cabeçalho e validar seu conteúdo.

API do ASP.NET Core trabalhar com essa convenção na inicialização do seu aplicativo:

- Configurar seu aplicativo para fornecer um token em um cookie chamado `XSRF-TOKEN`.
- Configurar o serviço antifalsificação para procurar por um cabeçalho chamado `X-XSRF-TOKEN`.

```

public void Configure(IApplicationBuilder app, IAntiforgery antiforgery)
{
    app.Use(next => context =>
    {
        string path = context.Request.Path.Value;

        if (
            string.Equals(path, "/", StringComparison.OrdinalIgnoreCase) ||
            string.Equals(path, "/index.html", StringComparison.OrdinalIgnoreCase))
        {
            // The request token can be sent as a JavaScript-readable cookie,
            // and Angular uses it by default.
            var tokens = antiforgery.GetAndStoreTokens(context);
            context.Response.Cookies.Append("XSRF-TOKEN", tokens.RequestToken,
                new CookieOptions() { HttpOnly = false });
        }

        return next(context);
    });
}

public void ConfigureServices(IServiceCollection services)
{
    // Angular's default header name for sending the XSRF token.
    services.AddAntiforgery(options => options.HeaderName = "X-XSRF-TOKEN");
}

```

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Estender antifalsificação

O [IAntiForgeryAdditionalDataProvider](#) tipo permite aos desenvolvedores estender o comportamento do sistema anti-CSRF por dados adicionais de ciclo completo em cada token. O [GetAdditionalData](#) método é chamado sempre que um token de campo é gerado e o valor de retorno é incorporado dentro do token gerado. Um implementador poderia retornar um carimbo de hora, um nonce ou qualquer outro valor e, em seguida, chame [ValidateAdditionalData](#) para validar dados quando o token é validado. Nome de usuário do cliente já está incorporado em tokens gerados, portanto, não há nenhuma necessidade de incluir essas informações. Se um token inclui dados complementares, mas não [IAntiForgeryAdditionalDataProvider](#) é configurado, os dados complementares não são validados.

## Recursos adicionais

- [CSRF](#) na [Abrir Web Application Security Project](#) (OWASP).
- [Hospedar o ASP.NET Core em um web farm](#)

# Impedir ataques de redirecionamento aberto no ASP.NET Core

23/08/2018 • 6 minutes to read • [Edit Online](#)

Um aplicativo web que redireciona para uma URL que é especificada por meio de solicitação, como os dados de formulário ou cadeia de consulta potencialmente pode ser violado para redirecionar usuários para uma URL externa e mal-intencionado. Essa violação é chamado de um ataque de redirecionamento aberto.

Sempre que a lógica do aplicativo redireciona para uma URL especificada, verifique se a URL de redirecionamento não foi adulterada. O ASP.NET Core tem funcionalidade interna para ajudar a proteger aplicativos contra ataques de redirecionamento aberto (também conhecido como open redirecionamento).

## O que é um ataque de redirecionamento aberto?

Aplicativos Web com frequência redirecionar os usuários para uma página de logon quando acessarem os recursos que exigem a autenticação. O redirecionamento normalmente inclui uma `returnUrl` parâmetro querystring para que o usuário pode ser retornado para a URL solicitada originalmente depois que eles fizeram logon com êxito. Depois que o usuário é autenticado, ele serão redirecionados para a URL que eles solicitaram originalmente.

Como a URL de destino é especificada na sequência de consulta da solicitação, um usuário mal-intencionado pode violar a querystring. Uma cadeia de consulta violada pode permitir que o site redirecionar o usuário para um site externo, mal-intencionado. Essa técnica é chamada de um ataque de redirecionamento (ou redirecionamento) aberto.

### Um ataque de exemplo

Um usuário mal-intencionado pode desenvolver um ataque de objetivo de permitir que o usuário mal-intencionado acesso a informações confidenciais ou as credenciais de um usuário. Para iniciar o ataque, o usuário mal-intencionado convence o usuário clicar em um link para a página de logon do seu site com um `returnUrl` valor de cadeia de consulta adicionada à URL. Por exemplo, considere um aplicativo na `contoso.com` que inclui uma página de logon no `http://contoso.com/Account/LogOn?returnUrl=/Home/About`. O ataque segue estas etapas:

1. O usuário clica no link para mal-intencionado

`http://contoso.com/Account/LogOn?returnUrl=http://contoso1.com/Account/LogOn` (a segunda URL é "contoso1.com", não "contoso.com").

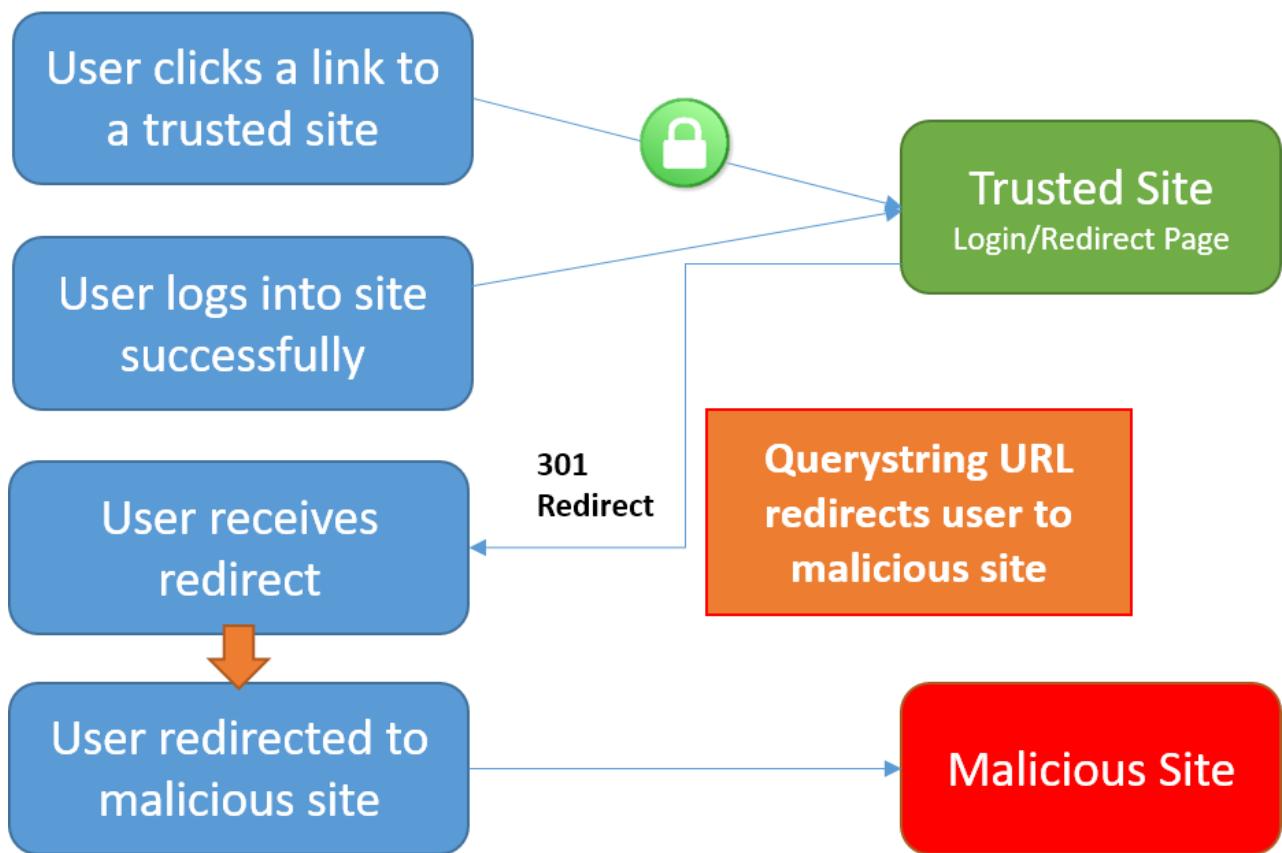
2. O usuário fizer logon com êxito.

3. O usuário é redirecionado (pelo site) para `http://contoso1.com/Account/LogOn` (um site mal-intencionado que se parece exatamente com o site real).

4. O usuário fizer logon novamente (dando mal-intencionado suas credenciais do site) e é redirecionado para o site real.

O usuário provavelmente acredita que sua primeira tentativa de fazer logon falhou e que sua segunda tentativa seja bem-sucedida. O usuário provavelmente permanecerá ciente de que suas credenciais estão comprometidas.

# Open Redirection Attack Process



Além das páginas de logon, alguns sites fornecem páginas de redirecionamento ou pontos de extremidade. Imagine que seu aplicativo tem uma página com um redirecionamento aberto, `/Home/Redirect`. Um invasor pode criar, por exemplo, um link em um email que vai para

`[yoursite]/Home/Redirect?url=http://phishingsite.com/Home/Login`. Um usuário típico examinará a URL e ver que ele começa com o nome do site. Confiar em que, eles clicarem no link. O redirecionamento aberto, em seguida, enviaria o usuário para o site de phishing, que parece ser idêntico ao seu, e provavelmente o usuário faça logon no que acreditam for seu site.

## Proteção contra ataques de redirecionamento abertos

Ao desenvolver aplicativos da web, trate todos os dados fornecidos pelo usuário como não confiáveis. Se seu aplicativo tem funcionalidade que redireciona o usuário com base no conteúdo da URL, certifique-se de que tais redirecionamentos são feitos somente localmente dentro de seu aplicativo (ou uma URL conhecida, não qualquer URL que pode ser fornecido na cadeia de consulta).

### LocalRedirect

Use o `LocalRedirect` o método auxiliar da base `Controller` classe:

```
public IActionResult SomeAction(string redirectUrl)
{
    return LocalRedirect(redirectUrl);
}
```

`LocalRedirect` lançará uma exceção se uma URL de local não for especificada. Caso contrário, ele se comporta exatamente como o `Redirect` método.

## IsLocalUrl

Use o `IsLocalUrl` método para testar a antes de redirecionar URLs:

O exemplo a seguir mostra como verificar se uma URL é local antes de redirecionar.

```
private IActionResult RedirectToLocal(string returnUrl)
{
    if (Url.IsLocalUrl(returnUrl))
    {
        return Redirect(returnUrl);
    }
    else
    {
        return RedirectToAction(nameof(HomeController.Index), "Home");
    }
}
```

O `IsLocalUrl` método protege os usuários contra inadvertidamente sendo redirecionado para um site mal-intencionado. Você pode registrar os detalhes da URL que foi fornecido quando uma URL de local não é fornecida em uma situação em que você esperava uma URL local. URLs de redirecionamento de registro em log podem ajudar no diagnóstico de ataques de redirecionamento.

# Evitar Cross-Site Scripting (XSS) no ASP.NET Core

10/10/2018 • 13 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

Criação de scripts entre sites (XSS) é uma vulnerabilidade de segurança que permite que um invasor inserir os scripts do lado do cliente (geralmente o JavaScript) em páginas da web. Quando outros usuários carregar páginas afetadas, scripts do invasor serão executados, permitindo que o invasor roubar cookies e tokens de sessão, altere o conteúdo da página da web por meio de manipulação de DOM ou redirecionar o navegador para outra página. As vulnerabilidades de XSS geralmente ocorrem quando um aplicativo recebe entrada do usuário e os coloca em uma página sem validação, codificação ou escape-lo.

## Proteger seu aplicativo contra o XSS

AT um XSS de nível básico funciona por enganar o seu aplicativo para inserir uma `<script>` marca em sua página processada, ou inserindo um `on*` eventos em um elemento. Os desenvolvedores devem usar as seguintes etapas de prevenção para evitar a introdução de XSS em seu aplicativo.

1. Nunca coloque dados não confiáveis em sua entrada HTML, a menos que você siga o restante das etapas a seguir. Dados não confiáveis são todos os dados que podem ser controlados por um invasor, entradas de formulário HTML, cadeias de caracteres de consulta, os cabeçalhos HTTP, até mesmo dados originados de um banco de dados como um invasor pode ser capaz de violar o seu banco de dados, mesmo se eles não podem violar o seu aplicativo.
2. Antes de colocar os dados não confiáveis dentro de um elemento HTML, verifique se que ele é codificado em HTML. A codificação HTML usa caracteres, como `<` e alterá-los em um formulário seguro como `&lt;`
3. Antes de colocar os dados não confiáveis em um atributo HTML, verifique se que ele é codificado em HTML. Codificação do atributo HTML é um superconjunto de codificação HTML e codifica os caracteres adicionais, como `"` e `'`.
4. Antes de colocar os dados não confiáveis em JavaScript, colocar os dados em um elemento HTML cujo conteúdo você recuperar em tempo de execução. Se isso não for possível, verifique se que os dados são codificado de JavaScript. Codificação de JavaScript usa caracteres perigosos para JavaScript e os substitui por sua hex, por exemplo `<` seria codificado como `\u003C`.
5. Antes de colocar os dados não confiáveis em uma cadeia de caracteres de consulta de URL, verifique se que ele é codificado por URL.

## Codificação HTML usando o Razor

O mecanismo Razor usado no MVC automaticamente codifica todos originado de saída de variáveis, a menos que você trabalha muito para impedi-lo ao fazer isso. Ele usa as regras de codificação HTML atributo sempre que você usar o `@` diretiva. Como HTML a codificação do atributo é um superconjunto de codificação de HTML, que isso significa que você não precisa se preocupar com se você deve usar a codificação HTML ou codificação do atributo HTML. Você deve garantir que você só usar em um contexto HTML, não quando a tentativa de inserir entradas não confiáveis diretamente em JavaScript. Os auxiliares de marca codificará também usar parâmetros de marca de entrada.

Execute o seguinte modo de exibição do Razor:

```
@{  
    var untrustedInput = "<\"123\">";  
}  
  
@untrustedInput
```

Essa exibição mostra o conteúdo do `untrustedInput` variável. Essa variável inclui alguns caracteres que são usadas em ataques de XSS, ou seja, <, "e >. Examinar o código-fonte mostra a saída renderizada codificada como:

```
&lt;&quot;123&quot;&gt;
```

### WARNING

ASP.NET Core MVC fornece uma `HtmlString` classe que não é codificado automaticamente após a saída. Isso nunca deve ser usado em combinação com entradas não confiáveis, pois isso irá expor uma vulnerabilidade de XSS.

## Usando o Razor de codificação do JavaScript

Pode haver ocasiões que você deseja inserir um valor em JavaScript para processar no modo de exibição. Há duas formas de fazer isso. A maneira mais segura para inserir valores é colocar o valor em um atributo de dados de uma marca e recuperá-lo em seu JavaScript. Por exemplo:

```
@{  
    var untrustedInput = "<\"123\">";  
}  
  
<div  
    id="injectedData"  
    data-untrustedinput="@untrustedInput" />  
  
<script>  
    var injectedData = document.getElementById("injectedData");  
  
    // All clients  
    var clientSideUntrustedInputOldStyle =  
        injectedData.getAttribute("data-untrustedinput");  
  
    // HTML 5 clients only  
    var clientSideUntrustedInputHtml5 =  
        injectedData.dataset.untrustedinput;  
  
    document.write(clientSideUntrustedInputOldStyle);  
    document.write("<br />")  
    document.write(clientSideUntrustedInputHtml5);  
</script>
```

Isso produzirá o HTML a seguir

```

<div
    id="injectedData"
    data-untrustedinput="<">123<">" />

<script>
    var injectedData = document.getElementById("injectedData");

    var clientSideUntrustedInputOldStyle =
        injectedData.getAttribute("data-untrustedinput");

    var clientSideUntrustedInputHtml5 =
        injectedData.dataset.untrustedinput;

    document.write(clientSideUntrustedInputOldStyle);
    document.write("<br />")
    document.write(clientSideUntrustedInputHtml5);
</script>

```

Que, quando ele é executado, será renderizado o seguinte:

```

<"123">
<"123">

```

Você também pode chamar diretamente o codificador de JavaScript:

```

@using System.Text.Encodings.Web;
@inject JavaScriptEncoder encoder;

@{
    var untrustedInput = "<">123<">";
}

<script>
    document.write("@encoder.Encode(untrustedInput)");
</script>

```

Isso será renderizado no navegador da seguinte maneira:

```

<script>
    document.write("\u003C\u0022123\u0022\u003E");
</script>

```

#### WARNING

Não concatene entradas não confiáveis em JavaScript para criar elementos DOM. Você deve usar `createElement()` e atribuir valores de propriedade adequadamente, como `node.textContent=`, ou use `element.setAttribute() / element[attribute]=` caso contrário, você se exporá a XSS baseado em DOM.

## Acessando codificadores no código

Os codificadores HTML, JavaScript e a URL estão disponíveis para seu código de duas maneiras, você pode colocá-los por meio [injeção de dependência](#) ou você pode usar os codificadores padrão contidos no `System.Text.Encodings.Web` namespace. Se você usar os codificadores padrão, qualquer aplicadas ao intervalos de caracteres a serem tratados como seguro não terão efeito - os codificadores padrão usam as regras de codificação mais seguras possíveis.

Para usar os codificadores configuráveis por meio da DI seus construtores devem levar uma `HtmlEncoder`, `JavaScriptEncoder` e `UrlEncoder` parâmetro conforme apropriado. Por exemplo:

```
public class HomeController : Controller
{
    HtmlEncoder _htmlEncoder;
    JavaScriptEncoder _javascriptEncoder;
    UrlEncoder _urlEncoder;

    public HomeController(HtmlEncoder htmlEncoder,
                          JavaScriptEncoder javascriptEncoder,
                          UrlEncoder urlEncoder)
    {
        _htmlEncoder = htmlEncoder;
        _javascriptEncoder = javascriptEncoder;
        _urlEncoder = urlEncoder;
    }
}
```

## Parâmetros de codificação de URL

Se você quiser criar uma cadeia de caracteres de consulta de URL com a entrada não confiável como um valor, use o `UrlEncoder` para codificar o valor. Por exemplo,

```
var example = "\"Quoted Value with spaces and &\"";
var encodedValue = _urlEncoder.Encode(example);
```

Após a codificação de `encodedValue` variável conterá `%22Quoted%20Value%20with%20spaces%20and%20%26%22`. Espaços, aspas, pontuação e outros caracteres não seguros serão porcentagem codificado para seu valor hexadecimal, por exemplo, um caractere de espaço tornará % 20.

### WARNING

Não use entradas não confiáveis como parte de um caminho de URL. Sempre passe a entrada não confiável como um valor de cadeia de caracteres de consulta.

## Personalizando os codificadores

Por padrão, codificadores usam uma lista segura limitada ao intervalo Unicode de Latim básico e codificar todos os caracteres fora do intervalo como seus equivalentes de código de caractere. Esse comportamento também afeta a renderização TagHelper Razor e HtmlHelper como ele utilizará os codificadores para suas cadeias de caracteres de saída.

O raciocínio por trás disso é proteger contra bugs no navegador desconhecido ou futuras (bugs no navegador anterior tiveram atropelados análise com base no processamento de caracteres do inglês). Se seu site da web faz uso intenso de caracteres não latinos, como o chinês, cirílico ou outras pessoas isso provavelmente não é o comportamento desejado.

Você pode personalizar as listas de seguro de codificador para incluir Unicode intervalos apropriadas para seu aplicativo durante a inicialização, no `ConfigureServices()`.

Por exemplo, usando a configuração padrão que você pode usar um `HtmlHelper` Razor assim;

```
<p>This link text is in Chinese: @Html.ActionLink("汉语/漢語", "Index")</p>
```

Quando você exibir a origem da página da web, você verá que ele foi renderizado da seguinte maneira, com o texto codificado; em chinês

<p>This link text is in Chinese: <a href="/">&#xC49;&#xBED;/&#xF22;&#x8A9E;</a></p>

Ampliar os caracteres tratados como seguro pelo codificador você inseriria a linha a seguir para o `ConfigureServices()` método no `startup.cs`:

```
services.AddSingleton<HtmlEncoder>(  
    HtmlEncoder.Create(allowedRanges: new[] { UnicodeRanges.BasicLatin,  
                                            UnicodeRanges.CjkUnifiedIdeographs }));
```

Este exemplo amplia a lista segura para incluir o CjkUnifiedIdeographs de intervalo Unicode. A saída renderizada tornaria

<p>This link text is in Chinese: <a href="/">汉语/漢語</a></p>

Intervalos de lista segura são especificados como gráficos de código Unicode, não os idiomas. O [padrão Unicode](#) tem uma lista de [gráficos de código](#) você pode usar para localizar o gráfico que contém os caracteres. Cada codificador, Html, JavaScript e a Url, deve ser configurado separadamente.

## NOTE

Personalização da lista de confiáveis afeta apenas os codificadores originados por meio da DI. Se você acessar diretamente um codificador via `System.Text.Encodings.Web.*Encoder.Default`, em seguida, o padrão, Latim básico somente lista segura será usada.

## Onde a codificação take deve colocar?

Em geral aceito prática é que a codificação ocorre no ponto de saída e valores codificados nunca devem ser armazenados em um banco de dados. Codificação no ponto de saída permite que você altere o uso de dados, por exemplo, de HTML para um valor de cadeia de caracteres de consulta. Ele também permite que você pesquise facilmente seus dados sem a necessidade de codificar valores antes de pesquisar e permite que você se beneficie de quaisquer alterações ou correções de bug feitas aos codificadores.

## Validação como uma técnica de prevenção de XSS

Validação pode ser uma ferramenta útil na limitação de ataques de XSS. Por exemplo, uma cadeia de caracteres numérica que contém somente os caracteres de 0 a 9 não disparar um ataque XSS. Validação se torna mais complicada ao aceitar HTML na entrada do usuário. Analisar a entrada em HTML é difícil, se não impossível. Markdown, juntamente com um analisador que retira HTML incorporado, é uma opção mais segura para aceitar a entrada avançada. Nunca confie em validação sozinha. Sempre codificar entradas não confiáveis antes da saída, não importa quais validação ou limpeza foi executada.

# Habilitar solicitações entre origens (CORS) no ASP.NET Core

28/11/2018 • 20 minutes to read • [Edit Online](#)

Por [Mike Wasson](#), [Shayne Boyer](#), e [Tom Dykstra](#)

Segurança do navegador impede que uma página da web fazendo solicitações para um domínio diferente daquele que atendia a página da web. Essa restrição é chamada de *política de mesma origem*. A política de mesma origem impede que um site mal-intencionado leia dados confidenciais de outro site. Às vezes, você talvez queira permitir que outros sites fazem solicitações entre origens em seu aplicativo.

[Entre o compartilhamento de recursos de origem](#) (CORS) é um padrão W3C que permite que um servidor relaxar a política de mesma origem. Usando o CORS, um servidor pode explicitamente permitir algumas solicitações entre origens enquanto rejeita outras. O CORS é mais seguro e flexível que técnicas anteriores, tais como [JSONP](#). Este tópico mostra como habilitar o CORS em um aplicativo ASP.NET Core.

## Mesma origem

Duas URLs têm a mesma origem se eles têm esquemas idênticas, hosts e portas ([6454 RFC](#)).

Essas duas URLs têm a mesma origem:

- `https://example.com/foo.html`
- `https://example.com/bar.html`

Essas URLs têm diferentes origens que as duas URLs anteriores:

- `https://example.net` – Domínio diferente
- `https://www.example.com/foo.html` – Subdomínio diferente
- `http://example.com/foo.html` – Esquema diferente
- `https://example.com:9000/foo.html` – Porta diferente

### NOTE

Internet Explorer não considera a porta ao comparar as origens.

## Registrar os serviços do CORS

Referência a [metapacote do Microsoft](#) ou adicionar uma referência de pacote para o [Microsoft.AspNetCore.Cors](#) pacote.

Referência a [metapacote Microsoft.AspNetCore.All](#) ou adicionar uma referência de pacote para o [Microsoft.AspNetCore.Cors](#) pacote.

Adicionar uma referência de pacote para o [Microsoft.AspNetCore.Cors](#) pacote.

Chame `AddCors` em `Startup.ConfigureServices` para adicionar serviços do CORS ao contêiner de serviço do aplicativo:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddCors();
}
```

## Habilitar o CORS

Depois de registrar os serviços do CORS, use qualquer uma das abordagens a seguir para habilitar o CORS em um aplicativo ASP.NET Core:

- [Middleware do CORS](#) – políticas CORS se aplicam globalmente para o aplicativo por meio do middleware.
- [O CORS no MVC](#) – políticas aplicar CORS por controlador ou por ação. Middleware CORS não é usada.

### Habilitar o CORS com Middleware CORS

Middleware CORS manipula solicitações entre origens para o aplicativo. Para habilitar o CORS Middleware no pipeline de processamento de solicitação, chame o `UseCors` método de extensão no `Startup.Configure`.

Middleware CORS devem preceder quaisquer pontos de extremidade definidos em seu aplicativo onde você deseja dar suporte a solicitações entre origens (por exemplo, antes de chamar `UseMvc` de Middleware de páginas do Razor/MVC).

Um *política entre origens* pode ser especificado ao adicionar o Middleware CORS usando o `CorsPolicyBuilder` classe. Há duas abordagens para definir uma política CORS:

- Chamar `UseCors` com um lambda:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    // Shows UseCors with CorsPolicyBuilder.
    app.UseCors(builder =>
        builder.WithOrigins("http://example.com"));

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Hello World!");
    });
}
```

O lambda utiliza um `CorsPolicyBuilder` objeto. [Opções de configuração](#), tais como `WithOrigins`, são descritas posteriormente neste tópico. No exemplo anterior, a política permite que solicitações entre origens de `https://example.com` e sem outras origens.

A URL deve ser especificada sem uma barra à direita (`/`). Se a URL termina com `/`, a comparação retorna `false` e nenhum cabeçalho é retornado.

`CorsPolicyBuilder` tem uma API fluente, portanto, é possível encadear chamadas de método:

```
app.UseCors(builder =>
    builder.WithOrigins("http://example.com")
        .AllowAnyHeader()
);
```

- Definir uma ou mais políticas CORS e selecione a política por nome em tempo de execução. O exemplo a seguir adiciona uma política CORS definida pelo usuário chamada *AllowSpecificOrigin*. Para selecionar a política, passe o nome para `UseCors`:

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddCors(options =>
    {
        options.AddPolicy("AllowSpecificOrigin",
            builder => builder.WithOrigins("http://example.com"));
    });
}

public void Configure(IApplicationBuilder app, IHostingEnvironment env,
    ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    // Shows UseCors with named policy.
    app.UseCors("AllowSpecificOrigin");

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Hello World!");
    });
}

```

## Habilitar o CORS no MVC

Como alternativa, você pode usar o MVC para aplicar políticas CORS específicas por ação ou por controlador. Ao usar o MVC para habilitar o CORS, os serviços registrados de CORS são usados. O Middleware do CORS não é usado.

### Por ação

Para especificar uma política CORS para uma ação específica, adicione a `[EnableCors]` atributo à ação. Especifique o nome da política.

```

[HttpGet]
[EnableCors("AllowSpecificOrigin")]
public IEnumerable<string> Get()
{
    return new string[] { "value1", "value2" };
}

```

### Por controlador

Para especificar a política CORS para um controlador específico, adicione a `[EnableCors]` atributo à classe do controlador. Especifique o nome da política.

```

[Route("api/[controller]")]
[EnableCors("AllowSpecificOrigin")]
public class ValuesController : ControllerBase

```

A ordem de precedência é:

1. ação

## 2. controlador

### Desabilitar o CORS

Para desabilitar CORS para um controlador ou ação, use o [\[DisableCors\]](#) atributo:

```
[HttpGet("{id}")]
[DisableCors]
public string Get(int id)
{
    return "value";
}
```

## Opções de política de CORS

Esta seção descreve as várias opções que podem ser definidas em uma política CORS. O [AddPolicy](#) método é chamado no `Startup.ConfigureServices`.

- [Defina as origens permitidas](#)
- [Defina os métodos HTTP permitidos](#)
- [Definir os cabeçalhos de solicitação permitido](#)
- [Definir os cabeçalhos de resposta exposto](#)
- [Credenciais nas solicitações entre origens](#)
- [Definir o tempo de expiração de simulação](#)

Para algumas opções, pode ser útil ler o [funciona como o CORS](#) seção pela primeira vez.

### Defina as origens permitidas

O middleware do CORS no ASP.NET Core MVC tem algumas maneiras de especificar origens permitidas:

- [WithOrigins](#) – Permite especificar uma ou mais URLs. A URL pode incluir o esquema, nome de host e porta sem nenhuma informação de caminho. Por exemplo, `https://example.com`. A URL deve ser especificada sem uma barra à direita (`/`).

```
options.AddPolicy("AllowSpecificOrigins",
    builder =>
{
    builder.WithOrigins("http://example.com",
        "http://www.contoso.com");
});
```

- [AllowAnyOrigin](#) – Permite que as solicitações CORS de todas as origens com qualquer esquema (`http` ou `https`).

```
options.AddPolicy("AllowAllOrigins",
    builder =>
{
    builder.AllowAnyOrigin();
});
```

Considere cuidadosamente antes de permitir que solicitações de qualquer origem. Permitir solicitações de qualquer origem significa que *de qualquer site* pode fazer solicitações entre origens ao seu aplicativo.

#### NOTE

Especificando `AllowAnyOrigin` e `AllowCredentials` é uma configuração insegura e podem resultar em falsificação de solicitação entre sites. O serviço CORS retorna uma resposta inválida do CORS, quando um aplicativo é configurado com os dois métodos.

#### NOTE

Especificando `AllowAnyOrigin` e `AllowCredentials` é uma configuração insegura e podem resultar em falsificação de solicitação entre sites. Considere a especificação de uma lista exata de origens se o cliente deve autorizar a mesmo para acessar recursos do servidor.

Essa configuração afeta as solicitações de simulação e o `Access-Control-Allow-Origin` cabeçalho. Para obter mais informações, consulte o [solicitações de simulação](#) seção.

- `SetIsOriginAllowedToAllowWildcardSubdomains` – Conjuntos de `IsOriginAllowed` propriedade da política para ser uma função que permite origens corresponder a um domínio de curinga configurado ao avaliar se a origem é permitida.

```
options.AddPolicy("AllowSubdomain",
    builder =>
{
    builder.SetIsOriginAllowedToAllowWildcardSubdomains();
});
```

#### Defina os métodos HTTP permitidos

Para permitir que todos os métodos HTTP, chamar `AllowAnyMethod`:

```
options.AddPolicy("AllowAllMethods",
    builder =>
{
    builder.WithOrigins("http://example.com")
        .AllowAnyMethod();
});
```

Essa configuração afeta as solicitações de simulação e o `Access-Control-Allow-Methods` cabeçalho. Para obter mais informações, consulte o [solicitações de simulação](#) seção.

#### Definir os cabeçalhos de solicitação permitido

Para permitir que os cabeçalhos específicos a serem enviados em uma solicitação CORS, chamado *criar cabeçalhos de solicitação*, chame `WithHeaders` e especificar os cabeçalhos permitidos:

```
options.AddPolicy("AllowHeaders",
    builder =>
{
    builder.WithOrigins("http://example.com")
        .WithHeaders(HeaderNames.ContentType, "x-custom-header");
});
```

Para permitir que todos os cabeçalhos de solicitação do autor chamar `AllowAnyHeader`:

```
options.AddPolicy("AllowAllHeaders",
    builder =>
{
    builder.WithOrigins("http://example.com")
        .AllowAnyHeader();
});
```

Essa configuração afeta as solicitações de simulação e o `Access-Control-Request-Headers` cabeçalho. Para obter mais informações, consulte o [solicitações de simulação](#) seção.

Uma correspondência de política de Middleware do CORS para cabeçalhos específicos especificado por `WithHeaders` só é possível quando os cabeçalhos enviados `Access-Control-Request-Headers` coincidir exatamente com os cabeçalhos indicados na `WithHeaders`.

Por exemplo, considere um aplicativo configurado da seguinte maneira:

```
app.UseCors(policy => policy.WithHeaders(HeaderNames.CacheControl));
```

Middleware CORS recusar uma solicitação de simulação com o seguinte cabeçalho de solicitação, pois `Content-Language` (`HeaderNames.ContentLanguage`) não está listado no `WithHeaders`:

```
Access-Control-Request-Headers: Cache-Control, Content-Language
```

O aplicativo retorna um `200 OK` resposta, mas não envia de volta os cabeçalhos de CORS. Portanto, o navegador não tenta a solicitação entre origens.

Middleware CORS sempre permite que os cabeçalhos de quatro no `Access-Control-Request-Headers` a ser enviado, independentemente dos valores configurados no `CorsPolicy.Headers`. Esta lista de cabeçalhos inclui:

- `Accept`
- `Accept-Language`
- `Content-Language`
- `Origin`

Por exemplo, considere um aplicativo configurado da seguinte maneira:

```
app.UseCors(policy => policy.WithHeaders(HeaderNames.CacheControl));
```

Middleware CORS responde com êxito a uma solicitação de simulação com o seguinte cabeçalho de solicitação porque `Content-Language` é sempre na lista de permissões:

```
Access-Control-Request-Headers: Cache-Control, Content-Language
```

## Definir os cabeçalhos de resposta exposto

Por padrão, o navegador não expõe todos os cabeçalhos de resposta para o aplicativo. Para obter mais informações, consulte [W3C Cross-Origin Resource Sharing \(terminologia\): o cabeçalho de resposta simples](#).

Os cabeçalhos de resposta que estão disponíveis por padrão são:

- `Cache-Control`
- `Content-Language`
- `Content-Type`

- `Expires`
- `Last-Modified`
- `Pragma`

A especificação CORS chama esses cabeçalhos *cabeçalhos de resposta simples*. Para disponibilizar outros cabeçalhos para o aplicativo, chame [WithExposedHeaders](#):

```
options.AddPolicy("ExposeResponseHeaders",
    builder =>
{
    builder.WithOrigins("http://example.com")
        .WithExposedHeaders("x-custom-header");
});
```

### Credenciais nas solicitações entre origens

Credenciais exigem tratamento especial em uma solicitação CORS. Por padrão, o navegador não envia as credenciais com uma solicitação entre origens. As credenciais incluem cookies e esquemas de autenticação HTTP. Para enviar as credenciais com uma solicitação entre origens, o cliente deve definir

`XMLHttpRequest.withCredentials` para `true`.

Usando `XMLHttpRequest` diretamente:

```
var xhr = new XMLHttpRequest();
xhr.open('get', 'https://www.example.com/api/test');
xhr.withCredentials = true;
```

No jQuery:

```
$.ajax({
    type: 'get',
    url: 'https://www.example.com/home',
    xhrFields: {
        withCredentials: true
    }
})
```

Além disso, o servidor deve permitir que as credenciais. Para permitir credenciais entre origens, chamar [AllowCredentials](#):

```
options.AddPolicy("AllowCredentials",
    builder =>
{
    builder.WithOrigins("http://example.com")
        .AllowCredentials();
});
```

A resposta HTTP inclui um `Access-Control-Allow-Credentials` cabeçalho, que informa ao navegador que o servidor permite que as credenciais para uma solicitação entre origens.

Se o navegador envia as credenciais, mas a resposta não inclui um válido `Access-Control-Allow-Credentials` cabeçalho, o navegador não expõe a resposta para o aplicativo e a solicitação entre origens falhar.

Tenha cuidado ao permitindo credenciais entre origens. Um site em outro domínio pode enviar credenciais do usuário conectado para o aplicativo em nome do usuário sem o conhecimento do usuário.

A especificação CORS também declara que a configuração origens `"*"` (todas as origens) é inválida se o `Access-Control-Allow-Credentials` cabeçalho está presente.

## Solicitações de simulação

Para algumas solicitações CORS, o navegador envia uma solicitação adicional antes de fazer a solicitação real. Essa solicitação é chamada de um *solicitação de simulação*. O navegador pode ignorar a solicitação de simulação se as seguintes condições forem verdadeiras:

- O método de solicitação é GET, HEAD ou POST.
- O aplicativo não define cabeçalhos de solicitação diferente de `Accept`, `Accept-Language`, `Content-Language`, `Content-Type`, OU `Last-Event-ID`.
- O `Content-Type` cabeçalho, se definido, tem um dos seguintes valores:
  - `application/x-www-form-urlencoded`
  - `multipart/form-data`
  - `text/plain`

A regra em cabeçalhos de solicitação definido para a solicitação do cliente se aplica aos cabeçalhos que o aplicativo define chamando `setRequestHeader` sobre o `XMLHttpRequest` objeto. A especificação CORS chama esses cabeçalhos *criar cabeçalhos de solicitação*. A regra não se aplica aos cabeçalhos, o navegador pode definir, tais como `User-Agent`, `Host`, OU `Content-Length`.

Este é um exemplo de uma solicitação de simulação:

```
OPTIONS https://myservice.azurewebsites.net/api/test HTTP/1.1
Accept: /*
Origin: https://myclient.azurewebsites.net
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: accept, x-my-custom-header
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0)
Host: myservice.azurewebsites.net
Content-Length: 0
```

A solicitação de simulação usa o método HTTP OPTIONS. Ele inclui dois cabeçalhos especiais:

- `Access-Control-Request-Method` : O método HTTP que será usado para a solicitação real.
- `Access-Control-Request-Headers` : Uma lista de cabeçalhos de solicitação que o aplicativo define a solicitação real. Conforme mencionado anteriormente, isso não inclui os cabeçalhos que define o navegador, tais como `User-Agent`.

Uma solicitação de simulação de CORS pode incluir um `Access-Control-Request-Headers` cabeçalho, que indica ao servidor, os cabeçalhos que são enviados com a solicitação real.

Para permitir que os cabeçalhos específicos, chame [WithHeaders](#):

```
options.AddPolicy("AllowHeaders",
    builder =>
{
    builder.WithOrigins("http://example.com")
        .WithHeaders(HeaderNames.ContentType, "x-custom-header");
});
```

Para permitir que todos os cabeçalhos de solicitação do autor chamar [AllowAnyHeader](#):

```
options.AddPolicy("AllowAllHeaders",
    builder =>
{
    builder.WithOrigins("http://example.com")
        .AllowAnyHeader();
});
```

Navegadores não estão totalmente consistentes em como eles definir `Access-Control-Request-Headers`. Se você definir os cabeçalhos para algo diferente de `"*"` (ou use [AllowAnyHeader](#)), você deve incluir pelo menos `Accept`, `Content-Type`, e `Origin`, além de quaisquer cabeçalhos personalizados que você deseja dar suporte.

Este é um exemplo de resposta à solicitação de simulação (supondo que o servidor permite que a solicitação):

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 0
Access-Control-Allow-Origin: https://myclient.azurewebsites.net
Access-Control-Allow-Headers: x-my-custom-header
Access-Control-Allow-Methods: PUT
Date: Wed, 20 May 2015 06:33:22 GMT
```

A resposta inclui um `Access-Control-Allow-Methods` cabeçalho que lista os métodos permitidos e, opcionalmente, um `Access-Control-Allow-Headers` cabeçalho, que lista os cabeçalhos permitidos. Se a solicitação de simulação for bem-sucedida, o navegador envia a solicitação real.

Se a solicitação de simulação for negada, o aplicativo retornar um `200 Okey` resposta, mas não envia de volta os cabeçalhos de CORS. Portanto, o navegador não tente a solicitação entre origens.

### Definir o tempo de expiração de simulação

O `Access-Control-Max-Age` cabeçalho Especifica quanto tempo a resposta à solicitação de simulação pode ser armazenados em cache. Para definir esse cabeçalho, chame [SetPreflightMaxAge](#):

```
options.AddPolicy("SetPreflightExpiration",
    builder =>
{
    builder.WithOrigins("http://example.com")
        .SetPreflightMaxAge(TimeSpan.FromSeconds(2520));
});
```

## Como funciona o CORS

Esta seção descreve o que acontece em uma solicitação CORS no nível das mensagens HTTP. É importante entender o funcionamento de CORS para que a política CORS pode ser configurada corretamente e depurada quando comportamentos inesperados ocorrerem.

A especificação CORS apresenta vários novos cabeçalhos HTTP que permitem que solicitações entre origens. Se um navegador oferece suporte a CORS, ele define esses cabeçalhos automaticamente para solicitações entre origens. Código JavaScript personalizado não é necessário para habilitar o CORS.

O exemplo a seguir é um exemplo de uma solicitação entre origens. O `Origin` cabeçalho fornece o domínio do site que está fazendo a solicitação:

```
GET https://myservice.azurewebsites.net/api/test HTTP/1.1
Referer: https://myclient.azurewebsites.net/
Accept: /*
Accept-Language: en-US
Origin: https://myclient.azurewebsites.net
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0)
Host: myservice.azurewebsites.net
```

Se o servidor permite que a solicitação, ele define o `Access-Control-Allow-Origin` cabeçalho na resposta. O valor desse cabeçalho também corresponde a `Origin` cabeçalho da solicitação ou é o valor de curinga `"*"`, o que significa que qualquer origem é permitida:

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: text/plain; charset=utf-8
Access-Control-Allow-Origin: https://myclient.azurewebsites.net
Date: Wed, 20 May 2015 06:27:30 GMT
Content-Length: 12

Test message
```

Se a resposta não inclui o `Access-Control-Allow-Origin` falha do cabeçalho, a solicitação entre origens. Especificamente, o navegador não permite a solicitação. Mesmo se o servidor retorna uma resposta bem-sucedida, o navegador não disponibiliza a resposta para o aplicativo cliente.

## Recursos adicionais

- [Cross-Origin Resource Sharing \(CORS\)](#)

# Compartilhar cookies entre aplicativos com o ASP.NET e ASP.NET Core

30/10/2018 • 11 minutes to read • [Edit Online](#)

Por [Rick Anderson](#) e [Luke Latham](#)

Sites geralmente consistem em aplicativos web individuais trabalhando juntos. Para fornecer uma experiência de logon único (SSO), aplicativos web dentro de um site devem compartilhar cookies de autenticação. Para dar suporte a esse cenário, a pilha de proteção de dados permite o compartilhamento de autenticação de cookie do Katana e tiquetes de autenticação de cookie do ASP.NET Core.

## [Exibir ou baixar código de exemplo \(como baixar\)](#)

O exemplo ilustra o cookie compartilhando entre três aplicativos que usam a autenticação de cookie:

- Aplicativo do ASP.NET Core 2.0 as páginas do Razor sem usar [ASP.NET Core Identity](#)
- Aplicativo MVC do ASP.NET Core 2.0 com o ASP.NET Core Identity
- Aplicativo MVC do ASP.NET Framework 4.6.1 com o ASP.NET Identity

Nos exemplos a seguir:

- O nome do cookie de autenticação é definido como um valor comum de `.AspNet.SharedCookie`.
- O `AuthenticationType` é definido como `Identity.Application` explicitamente ou por padrão.
- Um nome de aplicativo comum é usado para permitir que o sistema de proteção de dados compartilhar as chaves de proteção de dados (`SharedCookieApp`).
- `Identity.Application` é usado como o esquema de autenticação. Qualquer esquema é usada, ele deve ser usado de forma consistente *dentro e entre* os aplicativos de cookie compartilhado como o esquema padrão ou ao defini-lo. O esquema é usado ao criptografar e descriptografar cookies, portanto, um esquema consistente deve ser usado entre aplicativos.
- Um comum [chave da proteção de dados](#) armazenamento local é usado. O aplicativo de exemplo usa uma pasta chamada *token de autenticação* na raiz da solução para manter as chaves de proteção de dados.
- Em aplicativos ASP.NET Core, `PersistKeysToFileSystem` é usado para definir o local de armazenamento de chaves. `SetApplicationName` é usado para configurar um nome de aplicativo compartilhado comum.
- No aplicativo do .NET Framework, o middleware de autenticação de cookie usa uma implementação `DataProtectionProvider`. `DataProtectionProvider` fornece serviços de proteção de dados para a criptografia e descriptografia de dados de conteúdo do cookie de autenticação. O `DataProtectionProvider` instância é isolada do sistema de proteção de dados usado por outras partes do aplicativo.
  - `DataProtectionProvider.Create (DirectoryInfo, uma ação<IDataProtectionBuilder >)` aceita uma  `DirectoryInfo` para especificar o local para armazenamento de chaves de proteção de dados. O aplicativo de exemplo fornece o caminho da *token de autenticação* pasta a ser  `DirectoryInfo`.  
`DataProtectionBuilderExtensions.SetApplicationName` define o nome comum do aplicativo.
  - `DataProtectionProvider` exige as [Microsoft.AspNetCore.DataProtection.Extensions](#) pacote do NuGet. Para obter esse pacote para o ASP.NET Core 2.1 e posteriores aplicativos, fazer referência a [metapacote Microsoft](#). Ao direcionar o .NET Framework, adicione uma referência de pacote ao `Microsoft.AspNetCore.DataProtection.Extensions`.

## Compartilhar cookies de autenticação entre aplicativos do ASP.NET Core

Ao usar a identidade do ASP.NET Core:

No `ConfigureServices` método, use o `ConfigureApplicationCookie` método de extensão para configurar o serviço de proteção de dados para cookies.

```
services.AddDataProtection()
    .PersistKeysToFileSystem(GetKeyRingDirInfo())
    .SetApplicationName("SharedCookieApp");

services.ConfigureApplicationCookie(options => {
    options.Cookie.Name = ".AspNet.SharedCookie";
});
```

Chaves de proteção de dados e o nome do aplicativo devem ser compartilhados entre aplicativos. Em aplicativos de exemplo, `GetKeyRingDirInfo` retorna o local de armazenamento de chaves comuns para o `PersistKeysToFileSystem` método. Use `SetApplicationName` para configurar um nome de aplicativo compartilhado comum (`SharedCookieApp` no exemplo). Para obter mais informações, consulte [Configurando a proteção de dados](#).

Ao hospedar aplicativos que compartilham cookies entre subdomínios, especifique um domínio comum na `Cookie.Domain` propriedade. Compartilhar cookies entre aplicativos no `contoso.com`, como

`first_subdomain.contoso.com` e `second_subdomain.contoso.com`, especifique o `Cookie.Domain` como `.contoso.com`:

```
options.Cookie.Domain = ".contoso.com";
```

Consulte a `CookieAuthWithIdentity.Core` do projeto na [código de exemplo \(como baixar\)](#).

No `Configure` método, use o `CookieAuthenticationOptions` configurar:

- O serviço de proteção de dados para cookies.
- O `AuthenticationScheme` para coincidir com o ASP.NET 4.x.

```
app.AddIdentity< ApplicationUser, IdentityRole>(options =>
{
    options.Cookies.ApplicationCookie.AuthenticationScheme =
        "ApplicationCookie";

    var protectionProvider =
        DataProtectionProvider.Create(
            new DirectoryInfo(@"PATH_TO_KEY_RING_FOLDER"));

    options.Cookies.ApplicationCookie.DataProtectionProvider =
        protectionProvider;

    options.Cookies.ApplicationCookie.TicketDataFormat =
        new TicketDataFormat(protectionProvider.CreateProtector(
            "Microsoft.AspNetCore.Authentication.Cookies.CookieAuthenticationMiddleware",
            "Cookies",
            "v2"));
});
```

Ao usar cookies diretamente:

```

services.AddDataProtection()
    .PersistKeysToFileSystem(GetKeyRingDirInfo())
    .SetApplicationName("SharedCookieApp");

services.AddAuthentication("Identity.Application")
    .AddCookie("Identity.Application", options =>
{
    options.Cookie.Name = ".AspNet.SharedCookie";
});

```

Chaves de proteção de dados e o nome do aplicativo devem ser compartilhados entre aplicativos. Em aplicativos de exemplo, `GetKeyRingDirInfo` retorna o local de armazenamento de chaves comuns para o `PersistKeysToFileSystem` método. Use `SetApplicationName` para configurar um nome de aplicativo compartilhado comum (`SharedCookieApp` no exemplo). Para obter mais informações, consulte [Configurando a proteção de dados](#).

Ao hospedar aplicativos que compartilham cookies entre subdomínios, especifique um domínio comum na `Cookie.Domain` propriedade. Compartilhar cookies entre aplicativos no `contoso.com`, como

`first_subdomain.contoso.com` e `second_subdomain.contoso.com`, especifique o `Cookie.Domain` como `.contoso.com`:

```
options.Cookie.Domain = ".contoso.com";
```

Consulte a `CookieAuth.Core` do projeto na [código de exemplo \(como baixar\)](#).

```

app.UseCookieAuthentication(new CookieAuthenticationOptions
{
    DataProtectionProvider =
        DataProtectionProvider.Create(
            new DirectoryInfo(@"PATH_TO_KEY_RING_FOLDER"))
});

```

## Criptografar as chaves de proteção de dados em repouso

Para implantações de produção, configure o `DataProtectionProvider` para criptografar as chaves em repouso com DPAPI ou um X509Certificate. Ver [chave de criptografia em repouso](#) para obter mais informações.

```
services.AddDataProtection()
    .ProtectKeysWithCertificate("thumbprint");
```

```

app.UseCookieAuthentication(new CookieAuthenticationOptions
{
    DataProtectionProvider = DataProtectionProvider.Create(
        new DirectoryInfo(@"PATH_TO_KEY_RING"),
        configure =>
    {
        configure.ProtectKeysWithCertificate("thumbprint");
    })
});
```

## Compartilhamento de cookies de autenticação entre o ASP.NET 4.x e aplicativos ASP.NET Core

Os aplicativos do ASP.NET 4.x que usam o middleware de autenticação de cookie do Katana podem ser configurados para gerar os cookies de autenticação que são compatíveis com o middleware de autenticação de cookie do ASP.NET Core. Isso permite atualizar aplicativos individuais de um grande site gradativamente,

fornecendo uma experiência de SSO suave em todo o site.

Quando um aplicativo usa o middleware de autenticação de cookie do Katana, ele chama `UseCookieAuthentication` do projeto *Startup.Auth.cs* arquivo. Projetos de aplicativo web do ASP.NET 4.x criado com o Visual Studio 2013 e posterior, use o middleware de autenticação de cookie do Katana por padrão. Embora `UseCookieAuthentication` está obsoleto e sem suporte para aplicativos ASP.NET Core, chamando `UseCookieAuthentication` em um aplicativo do ASP.NET 4.x que usa o Katana middleware de autenticação de cookie é válido.

Um aplicativo do ASP.NET 4.x deve ter como destino do .NET Framework 4.5.1 ou superior. Caso contrário, não conseguir instalar os pacotes NuGet necessários.

Para compartilhar cookies de autenticação entre um aplicativo do ASP.NET 4.x e um aplicativo ASP.NET Core, configurar o aplicativo ASP.NET Core, conforme mencionado acima e, em seguida, configurar o aplicativo do ASP.NET 4.x, seguindo estas etapas:

1. Instale o pacote [Microsoft.Owin.Security.Interop](#) em cada aplicativo do ASP.NET 4.x.
2. Na *Startup.Auth.cs*, localize a chamada para `UseCookieAuthentication` e modifíca-lo da seguinte maneira. Altere o nome do cookie para corresponder ao nome usado pelo middleware de autenticação de cookie do ASP.NET Core. Fornecer uma instância de um `DataProtectionProvider` inicializado para o local de armazenamento de chaves de proteção de dados comuns. Certifique-se de que o nome do aplicativo é definido como o nome de aplicativo comum usado por todos os aplicativos que compartilham os cookies, `SharedCookieApp` no aplicativo de exemplo.

```
app.UseCookieAuthentication(new CookieAuthenticationOptions
{
    AuthenticationType = "Identity.Application",
    CookieName = ".AspNet.SharedCookie",
    LoginPath = new PathString("/Account/Login"),
    Provider = new CookieAuthenticationProvider
    {
        OnValidateIdentity =
            SecurityStampValidator
                .OnValidateIdentity<ApplicationUserManager, ApplicationUser>(
                    validateInterval: TimeSpan.FromMinutes(30),
                    regenerateIdentity: (manager, user) =>
                        user.GenerateUserIdentityAsync(manager))
    },
    TicketDataFormat = new AspNetTicketDataFormat(
        new DataProtectorShim(
            DataProtectionProvider.Create(GetKeyRingDirInfo()),
            (builder) => { builder.SetApplicationName("SharedCookieApp"); })
        .CreateProtector(
            "Microsoft.AspNetCore.Authentication.Cookies.CookieAuthenticationMiddleware",
            "Identity.Application",
            "v2")),
    CookieManager = new ChunkingCookieManager()
});
// If not setting http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier and
// http://schemas.microsoft.com/accesscontrolservice/2010/07/claims/identityprovider,
// then set UniqueClaimTypeIdentifier to a claim that distinguishes unique users.
System.Web.Helpers.AntiForgeryToken.UniqueClaimTypeIdentifier =
    "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name";
```

Consulte a *CookieAuthWithIdentity.NETFramework* do projeto na [código de exemplo \(como baixar\)](#).

Ao gerar uma identidade de usuário, o tipo de autenticação deve corresponder ao tipo definido em `AuthenticationType` definido com `UseCookieAuthentication`.

*Models/IdentityModels.cs*:

```
public async Task<ClaimsIdentity> GenerateUserIdentityAsync(UserManager<ApplicationUser> manager)
{
    // Note the authenticationType must match the one defined in
    CookieAuthenticationOptions.AuthenticationType
    var userIdentity = await manager.CreateIdentityAsync(this, "Identity.Application");
    // Add custom user claims here
    return userIdentity;
}
```

## Usar um banco de dados de usuário comum

Confirme que o sistema de identidade para cada aplicativo está apontado para o mesmo banco de dados do usuário. Caso contrário, o sistema de identidade gera falhas em tempo de execução quando ele tenta corresponder as informações no cookie de autenticação em relação as informações em seu banco de dados.

## Recursos adicionais

[Hospedar o ASP.NET Core em um web farm](#)

# Lista segura IP do cliente para o ASP.NET Core

30/10/2018 • 5 minutes to read • [Edit Online](#)

Por [Damien Bowden](#) e [Tom Dykstra](#)

Este artigo mostra três maneiras de implementar uma lista segura IP (também conhecido como uma lista de permissões) em um aplicativo ASP.NET Core. Você pode usar:

- Middleware para verificar o endereço IP remoto de cada solicitação.
- Filtros de ação para verificar o endereço IP remoto de solicitações para controladores específicos ou métodos de ação.
- Filtros de páginas do Razor para verificar o endereço IP remoto de solicitações de páginas do Razor.

O aplicativo de exemplo ilustra as duas abordagens. Em cada caso, uma cadeia de caracteres que contém os endereços IP de cliente aprovados é armazenada em uma configuração de aplicativo. O middleware ou filtro analisa a cadeia de caracteres em uma lista e verifica se o IP remoto estiver na lista. Caso contrário, será retornado um código de status HTTP 403 Proibido.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## A lista segura

A lista é configurada na `appSettings.json` arquivo. Ele é uma lista delimitada por ponto e vírgula e pode conter endereços IPv4 e IPv6.

```
{  
    "AdminSafeList": "127.0.0.1;192.168.1.5::1",  
    "Logging": {  
        "IncludeScopes": false,  
        "LogLevel": {  
            "Default": "Debug",  
            "System": "Information",  
            "Microsoft": "Information"  
        }  
    }  
}
```

## Middleware

O `Configure` método adiciona o middleware e passa a cadeia de caracteres de lista segura para ele em um parâmetro de construtor.

```
public void Configure(
    IApplicationBuilder app,
    IHostingEnvironment env,
    ILoggerFactory loggerFactory)
{
    loggerFactory.AddNLog();

    app.UseStaticFiles();

    app.UseMiddleware<AdminSafeListMiddleware>(
        Configuration["AdminSafeList"]);
    app.UseMvc();
}
```

O middleware analisa a cadeia de caracteres em uma matriz e procura o endereço IP remoto na matriz. Se o endereço IP remoto não for encontrado, o middleware retornará HTTP 401 proibido. Esse processo de validação é ignorado para solicitações HTTP Get.

```

public class AdminSafeListMiddleware
{
    private readonly RequestDelegate _next;
    private readonly ILogger<AdminSafeListMiddleware> _logger;
    private readonly string _adminSafeList;

    public AdminSafeListMiddleware(
        RequestDelegate next,
        ILogger<AdminSafeListMiddleware> logger,
        string adminSafeList)
    {
        _adminSafeList = adminSafeList;
        _next = next;
        _logger = logger;
    }

    public async Task Invoke(HttpContext context)
    {
        if (context.Request.Method != "GET")
        {
            var remoteIp = context.Connection.RemoteIpAddress;
            _logger.LogDebug($"Request from Remote IP address: {remoteIp}");

            string[] ip = _adminSafeList.Split(';');

            var bytes = remoteIp.GetAddressBytes();
            var badIp = true;
            foreach (var address in ip)
            {
                var testIp = IPAddress.Parse(address);
                if(testIp.GetAddressBytes().SequenceEqual(bytes))
                {
                    badIp = false;
                    break;
                }
            }

            if(badIp)
            {
                _logger.LogInformation(
                    $"Forbidden Request from Remote IP address: {remoteIp}");
                context.Response.StatusCode = (int) HttpStatusCode.Forbidden;
                return;
            }
        }

        await _next.Invoke(context);
    }
}

```

## Filtro de ação

Se você quiser uma lista segura somente para controladores específicos ou métodos de ação, use um filtro de ação.  
Veja um exemplo:

```

using System.Linq;
using System.Net;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Authorization;
using Microsoft.AspNetCore.Mvc.Filters;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;

namespace ClientIpAspNetCore.Filters
{
    public class ClientIdCheckFilter : ActionFilterAttribute
    {
        private readonly ILogger _logger;
        private readonly string _safelist;

        public ClientIdCheckFilter
            (ILoggerFactory loggerFactory, IConfiguration configuration)
        {
            _logger = loggerFactory.CreateLogger("ClientIdCheckFilter");
            _safelist = configuration["AdminSafeList"];
        }

        public override void OnActionExecuting(ActionExecutingContext context)
        {
            _logger.LogInformation(
                $"Remote IpAddress: {context.HttpContext.Connection.RemoteIpAddress}");

            var remoteIp = context.HttpContext.Connection.RemoteIpAddress;
            _logger.LogDebug($"Request from Remote IP address: {remoteIp}");

            string[] ip = _safelist.Split(';');

            var bytes = remoteIp.GetAddressBytes();
            var badIp = true;
            foreach (var address in ip)
            {
                var testIp = IPAddress.Parse(address);
                if (testIp.GetAddressBytes().SequenceEqual(bytes))
                {
                    badIp = false;
                    break;
                }
            }

            if (badIp)
            {
                _logger.LogInformation(
                    $"Forbidden Request from Remote IP address: {remoteIp}");
                context.Result = new StatusCodeResult(401);
                return;
            }

            base.OnActionExecuting(context);
        }
    }
}

```

O filtro de ação é adicionado ao contêiner de serviços.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddScoped<ClientIdCheckFilter>();

    services.AddMvc(options =>
    {
        options.Filters.Add
            (new ClientIdCheckPageFilter
                (_loggerFactory, Configuration));
    }).SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
}
```

O filtro, em seguida, pode ser usado em um método de ação ou controlador.

```
[ServiceFilter(typeof(ClientIdCheckFilter))]
[HttpGet]
public IEnumerable<string> Get()
```

No aplicativo de exemplo, o filtro é aplicado para o `Get` método. Portanto, quando você testar o aplicativo, enviando um `Get` solicitação de API, o atributo é validar o endereço IP do cliente. Quando você testa chamando a API com qualquer outro método HTTP, o middleware é validar o IP do cliente.

## Filtro de páginas do Razor

Se você quiser uma lista segura para um aplicativo páginas Razor, use um filtro de páginas do Razor. Veja um exemplo:

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;
using System;
using System.Linq;
using System.Net;

namespace ClientIpAspNetCore
{
    public class ClientIdCheckPageFilter : IPageFilter
    {
        private readonly ILogger _logger;
        private readonly string _safelist;

        public ClientIdCheckPageFilter
            (ILoggerFactory loggerFactory, IConfiguration configuration)
        {
            _logger = loggerFactory.CreateLogger("ClientIdCheckPageFilter");
            _safelist = configuration["AdminSafeList"];
        }

        public void OnPageHandlerExecuting(PageHandlerContext context)
        {
            _logger.LogInformation(
                $"Remote IpAddress: {context.HttpContext.Connection.RemoteIpAddress}");

            var remoteIp = context.HttpContext.Connection.RemoteIpAddress;
            _logger.LogDebug($"Request from Remote IP address: {remoteIp}");

            string[] ip = _safelist.Split(';');

            var bytes = remoteIp.GetAddressBytes();
            var badIp = true;
            foreach (var address in ip)
            {
                var testIp = IPAddress.Parse(address);
                if (testIp.GetAddressBytes().SequenceEqual(bytes))
                {
                    badIp = false;
                    break;
                }
            }

            if (badIp)
            {
                _logger.LogInformation(
                    $"Forbidden Request from Remote IP address: {remoteIp}");
                context.Result = new StatusCodeResult(401);
                return;
            }
        }

        public void OnPageHandlerExecuted(PageHandlerContext context)
        {}

        public void OnPageHandlerSelected(PageHandlerSelectedContext context)
        {}
    }
}

```

Esse filtro é habilitado adicionando-o à coleção de filtros MVC.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddScoped<ClientIdCheckFilter>();

    services.AddMvc(options =>
    {
        options.Filters.Add
            (new ClientIdCheckPageFilter
                (_loggerFactory, Configuration));
    }).SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
}
```

Quando você executa o aplicativo e solicita uma página Razor, o filtro de páginas do Razor é validar o IP do cliente.

## Próximas etapas

[Saiba mais sobre o Middleware do ASP.NET Core.](#)

# Práticas recomendadas de desempenho do ASP.NET Core

08/01/2019 • 14 minutes to read • [Edit Online](#)

Por [Mike Rousos](#)

Este tópico fornece diretrizes para desempenho, as práticas recomendadas com o ASP.NET Core.

Neste documento, um caminho de código a quente é definido como um caminho de código que é chamado com frequência e em que grande parte do tempo de execução ocorre. Caminhos de código ativo normalmente limitam a expansão de aplicativo e o desempenho.

## Armazenar em cache agressivamente

Armazenamento em cache é discutido em várias partes deste documento. Para obter mais informações, consulte [Cache de resposta no ASP.NET Core](#).

## Evitar chamadas de bloqueio

Aplicativos ASP.NET Core devem ser projetados para processar muitas solicitações simultaneamente. APIs assíncronas permitem que um pequeno pool de threads para lidar com milhares de solicitações simultâneas por não aguardando chamadas de bloqueio. Em vez de esperar em uma tarefa de longa execução síncrona para ser concluída, o thread pode trabalhar em outra solicitação.

Um problema de desempenho comuns em aplicativos ASP.NET Core está bloqueando chamadas que poderiam ser assíncronas. Muitas chamadas de bloqueio síncronas leva à [privação de Pool de threads](#) e prejudicar os tempos de resposta.

### Não:

- Bloquear a execução assíncrona chamando [Task.wait](#) ou [Task.Result](#).
- Adquira bloqueios em caminhos de código comum. Aplicativos ASP.NET Core são mais eficazes quando projetado para executar código em paralelo.

### Fazer:

- Tornar [hot caminhos de código](#) assíncrona.
- Chame APIs de operações de longa execução e de acesso a dados de forma assíncrona.
- Tornar o controlador/Razor ações de página assíncrono. A pilha de chamadas inteira precisa ser assíncrono para se beneficiar da [async/await](#) padrões.

Como um criador de perfil [PerfView](#) pode ser usado para procurar por segmentos com frequência que está sendo adicionados para o [Pool de threads](#). O `Microsoft-Windows-DotNETRuntime/ThreadPoolWorkerThread/Start` evento indica que um thread que está sendo adicionado ao pool de threads.

## Minimizar as alocações de objeto grande

O [coletor de lixo do .NET Core](#) gerencia a alocação e liberação de memória automaticamente em aplicativos ASP.NET Core. Coleta de lixo automática geralmente significa que os desenvolvedores não precisam se preocupar sobre como ou quando a memória é liberada. No entanto, limpeza de objetos não referenciados leva tempo de CPU, portanto, os desenvolvedores devem minimizar alocando objetos no [hot caminhos de código](#). Coleta de lixo é

especialmente dispendiosa em objetos grandes (> 85 mil bytes). Objetos grandes são armazenados na [heap de objeto grande](#) e exigem uma completa (geração 2) a coleta de lixo para limpar. Ao contrário de geração 0 e coletas da geração 1, uma coleta de geração 2 requer a execução do aplicativo ser temporariamente suspenso. Frequentes alocação e desalocação de objetos grandes podem causar inconsistência do desempenho.

Recomendações:

- **Fazer** armazenar em cache os objetos grandes que são usados com frequência. O cache de objetos grandes evita alocações caras.
- **Fazer** pool de buffers usando um [ArrayPool<T>](#) para armazenar as matrizes grandes.
- **Não o fazem** alocar curta duração, muitos objetos grandes na [hot caminhos de código](#).

Problemas de memória, como anterior pode ser diagnosticado ao examinar estatísticas de (GC) coleta de lixo na [PerfView](#) e examinando:

- Tempo de pausa da coleta de lixo.
- Qual é a porcentagem do tempo do processador é gasto na coleta de lixo.
- Quantas coletas de lixo são a geração 0, 1 e 2.

Para obter mais informações, consulte [coleta de lixo e desempenho](#).

## Otimizar o acesso a dados

Interações com um armazenamento de dados ou outros serviços remotos costumam ser a parte mais lenta de um aplicativo ASP.NET Core. Ler e gravar dados com eficiência é essencial para um bom desempenho.

Recomendações:

- **Fazer** chamar APIs de acesso a todos os dados de forma assíncrona.
- **Não** recuperar mais dados do que é necessário. Escreva consultas para retornar apenas os dados que é necessário para a solicitação HTTP atual.
- **Fazer** considere cache frequentemente acessado dados recuperados de um banco de dados ou serviço remoto, se for aceitável para os dados sejam um pouco desatualizadas. Dependendo do cenário, você pode usar um [MemoryCache](#) ou um [DistributedCache](#). Para obter mais informações, consulte [Cache de resposta no ASP.NET Core](#).
- Minimizar viagens de ida e volta de rede. O objetivo é recuperar todos os dados que serão necessários em uma única chamada, em vez de várias chamadas.
- **Fazer** usar [consultas sem controle](#) no Entity Framework Core ao acessar dados para propósitos de somente leitura. O EF Core pode retornar os resultados de consultas sem controle com mais eficiência.
- **Fazer** filtro e consultas de agregação LINQ (com `.Where`, `.Select`, ou `.Sum` instruções, por exemplo) para que a filtragem é feita pelo banco de dados.
- **Fazer** considerar que o EF Core resolve alguns operadores de consulta no cliente, que pode levar à execução de consulta ineficiente. Para obter mais informações, consulte [problemas de desempenho de avaliação do cliente](#)
- **Não** usar consultas de projeção em coleções, que podem resultar na execução de "N + 1" consultas SQL. Para obter mais informações, consulte [otimização de subconsultas correlacionadas](#).

Ver [EF alto desempenho](#) para abordagens que podem melhorar o desempenho em aplicativos de alta escala:

- [Pool de DbContext](#)
- [Consultas explicitamente compiladas](#)

É recomendável que você mede o impacto das abordagens de alto desempenho anteriores antes de confirmar sua base de código. A complexidade adicional de consultas compiladas pode não justificar a melhoria de desempenho.

Consulta problemas podem ser detectados, examinando o tempo gasto acesso a dados com [Application Insights](#)

ou com ferramentas de criação de perfil. A maioria dos bancos de dados também disponibilizar as estatísticas relacionadas a consultas executadas com frequência.

## HTTP do pool de conexões com HttpClientFactory

Embora [HttpClient](#) implementa o [IDisposable](#) interface, que ele foi projetado para ser reutilizado. Fechado [HttpClient](#) instâncias deixa soquetes abertos no [TIME\\_WAIT](#) estado por um curto período de tempo. Consequentemente, se um caminho de código que cria e descarta [HttpClient](#) objetos é usado com frequência, o aplicativo pode esgotar soquetes disponíveis. [HttpClientFactory](#) foi introduzido no ASP.NET Core 2.1 como uma solução para esse problema. Ele lida com o pooling de conexões de HTTP para otimizar o desempenho e confiabilidade.

Recomendações:

- **Não o fazem** criar e descartar [HttpClient](#) instâncias diretamente.
- **Fazer** usar [HttpClientFactory](#) recuperar [HttpClient](#) instâncias. Para obter mais informações, consulte [HttpClientFactory de uso para implementar as solicitações HTTP resilientes](#).

## Manter os caminhos de código comuns rápidos

Você deseja que todos os seus códigos para ser rápido, mas caminhos de código chamado com frequência são mais importantes para otimizar:

- Componentes de middleware no pipeline de processamento de solicitação do aplicativo, especialmente middleware executado no início do pipeline. Esses componentes têm um grande impacto no desempenho.
- Código que é executado para cada solicitação ou várias vezes por solicitação. Por exemplo, registro em log personalizado, manipuladores de autorização ou inicialização de serviços temporários.

Recomendações:

- **Não** usar componentes de middleware personalizado com tarefas de longa execução.
- **Fazer** usar ferramentas de criação de perfil de desempenho (como [ferramentas de diagnóstico Visual Studio](#) ou [PerfView](#)) para identificar [hot caminhos de código](#).

## Concluir tarefas de execução longa fora de solicitações HTTP

A maioria das solicitações para um aplicativo ASP.NET Core podem ser tratadas por um controlador ou o modelo de página chamando serviços necessários e retornando uma resposta HTTP. Para algumas solicitações que envolvem tarefas de longa execução, é melhor tornar o processo de solicitação-resposta inteira assíncrona.

Recomendações:

- **Não** Aguarde até que as tarefas de longa execução ser concluída como parte do processamento de solicitação HTTP comuns.
- **Fazer** considere a manipulação de solicitações de execução longa com [serviços em segundo plano](#) ou fora do processo com um [Azure Function](#). A conclusão do trabalho fora do processo é especialmente útil para tarefas de uso intensivo de CPU.
- **Fazer** use opções de comunicação em tempo real, como [SignalR](#) para se comunicar com clientes de forma assíncrona.

## Minificar ativos do cliente

Aplicativos ASP.NET Core com front-ends complexos com frequência servem muitos JavaScript, CSS ou arquivos de imagem. Desempenho de solicitações de carga inicial pode ser melhorado por:

- O agrupamento, que combina vários arquivos em um.
- Minificação, que reduz o tamanho dos arquivos por.

Recomendações:

- **Fazer** usar o ASP.NET Core [suporte interno](#) para agrupamento e minificação de ativos de cliente.
- **Fazer** considerar outras ferramentas de terceiros, como [Gulp](#) ou [Webpack](#) para gerenciamento de ativos mais complexo do cliente.

## Compactar respostas

Reducindo o tamanho da resposta geralmente aumenta a capacidade de resposta de um aplicativo, muitas vezes drasticamente. É uma maneira de reduzir os tamanhos do conteúdo compactar respostas do aplicativo. Para obter mais informações, consulte [compactação de resposta](#).

## Use a versão mais recente do ASP.NET Core

Cada nova versão do ASP.NET inclui aprimoramentos de desempenho. Otimizações no .NET Core e ASP.NET Core significam que versões mais recentes superará as versões mais antigas. Por exemplo, o .NET Core 2.1 adicionado suporte para expressões regulares compiladas e se beneficiaram [Span<T>](#). Suporte do ASP.NET Core 2.2 adicionado para HTTP/2. Se o desempenho for uma prioridade, considere atualizar para a versão mais atual do ASP.NET Core.

## Minimizar as exceções

As exceções devem ser raras. Lançar e capturar exceções são lento em relação a outros padrões de fluxo de código. Por causa disso, as exceções não devem ser usadas para controlar o fluxo normal do programa.

Recomendações:

- **Não** uso gerar ou capturar exceções como um meio de fluxo normal do programa, especialmente em caminhos de código a quente.
- **Fazer** incluir lógica no aplicativo para detectar e lidar com condições que poderiam causar uma exceção.
- **Fazer** lançar ou capturar exceções para condições incomuns ou inesperadas.

Ferramentas de diagnóstico de aplicativo (como o Application Insights) podem ajudar a identificar exceções comuns em um aplicativo que podem afetar o desempenho.

# Cache de resposta no ASP.NET Core

08/01/2019 • 16 minutes to read • [Edit Online](#)

Por [John Luo](#), [Rick Anderson](#), [Steve Smith](#), e [Luke Latham](#)

## NOTE

Cache de resposta nas páginas do Razor está disponível no ASP.NET Core 2.1 ou posterior.

### [Exibir ou baixar código de exemplo \(como baixar\)](#)

O cache das respostas reduz o número de solicitações de que um cliente ou proxy faz a um servidor web. Cache de resposta também reduz a quantidade de trabalho do servidor web executa para gerar uma resposta. Cache de resposta é controlado por cabeçalhos que especificam como deseja cliente, o proxy e middleware para respostas em cache.

O [atributo ResponseCache](#) participa na configuração de cabeçalhos, quais clientes podem honrar ao armazenar em cache as respostas do cache de resposta. [Middleware de cache de resposta](#) pode ser usada para respostas em cache no servidor. O middleware pode usar `ResponseCache` propriedades para influenciar o comportamento de cache do lado do servidor do atributo.

## Cache de resposta baseado em HTTP

O [especificação de cache do HTTP 1.1](#) descreve como caches de Internet devem se comportar. É o cabeçalho HTTP principal usado para armazenar em cache [Cache-Control](#), que é usada para especificar o cache *diretivas*. As diretivas de controlam o comportamento de cache conforme as solicitações cheguem de clientes para servidores e respostas passarão de servidores de volta aos clientes. Solicitações e respostas movem através de servidores proxy e servidores proxy devem também estar em conformidade com a especificação de cache do HTTP 1.1.

Common `Cache-Control` diretivas são mostradas na tabela a seguir.

DIRETIVA	AÇÃO
<code>public</code>	Um cache pode armazenar a resposta.
<code>private</code>	A resposta não deve ser armazenada por um cache compartilhado. Um cache privado pode armazenar e reutilizar a resposta.
<code>max-age</code>	O cliente não aceitará uma resposta cuja idade é maior que o número especificado de segundos. Exemplos: <code>max-age=60</code> (60 segundos), <code>max-age=2592000</code> (1 mês)

DIRETIVA	AÇÃO
no-cache	<p><b>Em solicitações:</b> Um cache não deve usar uma resposta armazenada para atender à solicitação. Observação: O servidor de origem gera novamente a resposta para o cliente e o middleware atualiza a resposta armazenada em seu cache.</p> <p><b>Nas respostas:</b> A resposta não deve ser usada para uma solicitação subsequente sem validação no servidor de origem.</p>
no-store	<p><b>Em solicitações:</b> Um cache não deve armazenar a solicitação.</p> <p><b>Nas respostas:</b> Um cache não deve armazenar qualquer parte da resposta.</p>

Outros cabeçalhos de cache que desempenham uma função em cache são mostrados na tabela a seguir.

CABEÇALHO	FUNÇÃO
Idade	Uma estimativa da quantidade de tempo em segundos desde a resposta foi gerada ou validada com êxito no servidor de origem.
Expira	A data/hora após o qual a resposta é considerada obsoleta.
Pragma	Existe para com versões anteriores a compatibilidade com o HTTP/1.0 armazena em cache para a configuração <code>no-cache</code> comportamento. Se o <code>Cache-Control</code> cabeçalho estiver presente, o <code>Pragma</code> cabeçalho será ignorado.
Variar	Especifica que uma resposta em cache não deve ser enviada, a menos que todos os do <code>vary</code> correspondem de campos de cabeçalho na solicitação original da resposta em cache e a nova solicitação.

## As diretivas de controle de Cache de solicitação de aspectos de cache baseada em HTTP

O [especificação de cache do HTTP 1.1 para o cabeçalho Cache-Control](#) requer um cache para honrar válido `Cache-Control` cabeçalho enviado pelo cliente. Um cliente pode fazer solicitações com um `no-cache` força o servidor gere uma nova resposta para cada solicitação e o valor do cabeçalho.

Respeitando sempre cliente `Cache-Control` cabeçalhos de solicitação faz sentido se você considerar o objetivo do cache de HTTP. Sob a especificação oficial, cache destina-se para reduzir a sobrecarga de rede e latência de satisfazer as solicitações em uma rede de clientes, proxies e servidores. Ele não é necessariamente uma maneira de controlar a carga em um servidor de origem.

Não é possível controlar de desenvolvedor sobre o comportamento de cache ao usar o [Middleware de cache de resposta](#) porque o middleware adere à especificação de cache oficial. [Planejado aprimoramentos para o middleware](#) são uma oportunidade para configurar o middleware para ignorar uma solicitação `Cache-Control` cabeçalho ao decidir servir uma resposta em cache. Aprimoramentos planejados oferecem uma oportunidade de melhor carga do servidor de controle.

# Outra tecnologia de armazenamento em cache no ASP.NET Core

## Cache na memória

Cache na memória usa a memória do servidor para armazenar dados armazenados em cache. Esse tipo de cache é adequado para um único servidor ou vários servidores usando *sessões adesivas*. Sessões adesivas significa que as solicitações de um cliente sejam sempre roteadas para o mesmo servidor para processamento.

Para obter mais informações, consulte [armazenar em Cache na memória](#).

## Cache distribuído

Use um cache distribuído para armazenar dados na memória quando o aplicativo é hospedado em um farm de servidor ou de nuvem. O cache é compartilhado entre os servidores que processam solicitações. Um cliente pode enviar uma solicitação que é tratada por qualquer servidor no grupo se os dados armazenados em cache para o cliente estão disponíveis. ASP.NET Core oferece o SQL Server e os caches distribuído do Redis.

Para obter mais informações, consulte [O cache no ASP.NET Core distribuído](#).

## Auxiliar de marca de cache

Você pode armazenar em cache o conteúdo de uma exibição MVC ou a página do Razor com o auxiliar de marca de Cache. O auxiliar de marca de Cache usa o cache na memória para armazenar dados.

Para obter mais informações, consulte [auxiliar de marca de Cache no ASP.NET Core MVC](#).

## Auxiliar de Marca de Cache Distribuído

Você pode armazenar em cache o conteúdo de uma exibição MVC ou a página do Razor em nuvem distribuído ou cenários de web farm com o auxiliar de marca de Cache distribuído. O auxiliar de marca de Cache distribuído usa o SQL Server ou do Redis para armazenar dados.

Para obter mais informações, consulte [auxiliar de marca de Cache distribuído](#).

## Atributo ResponseCache

O [ResponseCacheAttribute](#) especifica os parâmetros necessários para a configuração de cabeçalhos apropriados no cache de resposta.

### WARNING

Desabilite o cache para o conteúdo que contém informações para clientes autenticados. Armazenamento em cache deve ser habilitado apenas para conteúdo que não são alteradas com base na identidade do usuário ou se um usuário está conectado.

[VaryByQueryKeys](#) a resposta armazenada de varia de acordo com os valores de determinada lista de chaves de consulta. Quando um valor único de `*` é fornecido, o middleware varia as respostas de todos os parâmetros de cadeia de caracteres de consulta de solicitação. `VaryByQueryKeys` exige o ASP.NET Core 1.1 ou posterior.

[Middleware de cache de resposta](#) deve ser habilitado para definir o `VaryByQueryKeys` propriedade; caso contrário, uma exceção de tempo de execução é gerada. Não há um cabeçalho HTTP correspondente para o `VaryByQueryKeys` propriedade. A propriedade é um recurso HTTP tratado pelo Middleware de cache de resposta. Para o middleware servir uma resposta em cache, a cadeia de caracteres de consulta e o valor de cadeia de caracteres de consulta devem corresponder com uma solicitação anterior. Por exemplo, considere a sequência de solicitações e os resultados mostrados na tabela a seguir.

SOLICITAÇÃO	RESULTADO
http://example.com?key1=value1	Retornado do servidor
http://example.com?key1=value1	Retornado de middleware
http://example.com?key1=value2	Retornado do servidor

A primeira solicitação é retornada pelo servidor e armazenados em cache no middleware. A segunda solicitação é retornada pelo middleware, como a cadeia de caracteres de consulta corresponde a solicitação anterior. A terceira solicitação não está no cache do middleware porque o valor de cadeia de caracteres de consulta não corresponde a uma solicitação anterior.

O `ResponseCacheAttribute` é usado para configurar e criar (via `IFilterFactory`) uma `ResponseCacheFilter`. O `ResponseCacheFilter` realiza o trabalho de atualização de cabeçalhos HTTP apropriados e recursos da resposta.

O filtro:

- Remove qualquer cabeçalho existente para `Vary`, `Cache-Control`, e `Pragma`.
- Grava os cabeçalhos apropriados com base nas propriedades definidas `ResponseCacheAttribute`.
- Atualiza a resposta de armazenamento em cache o recurso HTTP se `VaryByQueryKeys` está definido.

## Variar

Esse cabeçalho só será gravado quando o `VaryByHeader` propriedade está definida. Ele é definido como o `Vary` valor da propriedade. O exemplo a seguir usa o `VaryByHeader` propriedade:

```
[ResponseCache(VaryByHeader = "User-Agent", Duration = 30)]
public IActionResult About2()
{
```

```
[ResponseCache(VaryByHeader = "User-Agent", Duration = 30)]
public IActionResult About2()
{
```

Você pode exibir os cabeçalhos de resposta com as ferramentas de rede do seu navegador. A imagem a seguir mostra F12 borda de saída na `rede` guia quando o `About2` método de ação é atualizado:

Headers	Body	Parameters	Cookies	Timings
Request URL: <a href="http://localhost/Home/About2">http://localhost/Home/About2</a>				
Request Method: GET				
Status Code: <span style="background-color: #2e7131; color: white; padding: 2px 5px;">200 / OK</span>				
<b>Request Headers</b> Accept: text/html, application/xhtml+xml, image/jxr, */* Accept-Encoding: gzip, deflate Accept-Language: en-US, en; q=0.8, zh-Hans-CN; q=0.5, zh-Hans; q=0.3 Connection: Keep-Alive Host: localhost User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36...				
<b>Response Headers</b> Cache-Control: public, max-age=30 Content-Type: text/html; charset=utf-8 Date: Fri, 18 Nov 2016 21:45:14 GMT Server: Kestrel Transfer-Encoding: chunked Vary: User-Agent				

## NoStore e Location.None

`NoStore` substitui a maioria das outras propriedades. Quando essa propriedade é definida como `true`, o `Cache-Control` cabeçalho é definido como `no-store`. Se `Location` é definido como `None`:

- `Cache-Control` é definido como `no-store,no-cache`.
- `Pragma` é definido como `no-cache`.

Se `NoStore` está `false` e `Location` é `None`, `Cache-Control` e `Pragma` são definidos como `no-cache`.

Você normalmente define `NoStore` para `true` nas páginas de erro. Por exemplo:

```
[ResponseCache(Location = ResponseCacheLocation.None, NoStore = true)]
public IActionResult Error()
{
    return View();
}
```

```
[ResponseCache(Location = ResponseCacheLocation.None, NoStore = true)]
public IActionResult Error()
{
    return View();
}
```

Isso resulta nos seguintes cabeçalhos:

```
Cache-Control: no-store,no-cache
Pragma: no-cache
```

## Local e a duração

Para habilitar o cache, `Duration` deve ser definida como um valor positivo e `Location` deve ser `Any` (o padrão) ou `client`. Nesse caso, o `Cache-Control` cabeçalho é definido como o valor do local seguido o `max-age` da resposta.

#### NOTE

`Location` da opções dos `Any` e `Client` traduzir `Cache-Control` valores de cabeçalho da `public` e `private`, respectivamente. Conforme observado anteriormente, definindo `Location` à `None` define ambas `Cache-Control` e `Pragma` cabeçalhos para `no-cache`.

Veja abaixo um exemplo que mostra os cabeçalhos é produzido pela configuração `Duration` e deixar o padrão `Location` valor:

```
[ResponseCache(Duration = 60)]
public IActionResult Contact()
{
    ViewData["Message"] = "Your contact page.";

    return View();
}
```

```
[ResponseCache(Duration = 60)]
public IActionResult Contact()
{
    ViewData["Message"] = "Your contact page.";

    return View();
}
```

Isso produz o seguinte cabeçalho:

```
Cache-Control: public,max-age=60
```

#### Perfis de cache

Em vez de duplicar `ResponseCache` configurações de muitos atributos de ação do controlador, perfis de cache podem ser configuradas como opções ao configurar o MVC em de `ConfigureServices` método na `Startup`. Valores encontrados em um perfil de cache referenciada são usados como padrões pelo `ResponseCache` de atributos e são substituídas por qualquer propriedade especificada no atributo.

Como configurar um perfil de cache:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(options =>
    {
        options.CacheProfiles.Add("Default",
            new CacheProfile()
            {
                Duration = 60
            });
        options.CacheProfiles.Add("Never",
            new CacheProfile()
            {
                Location = ResponseCacheLocation.None,
                NoStore = true
            });
    }).SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
}
```

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(options =>
    {
        options.CacheProfiles.Add("Default",
            new CacheProfile()
            {
                Duration = 60
            });
        options.CacheProfiles.Add("Never",
            new CacheProfile()
            {
                Location = ResponseCacheLocation.None,
                NoStore = true
            });
    });
}

```

Um perfil de cache de referência:

```

[ResponseCache(Duration = 30)]
public class HomeController : Controller
{
    [ResponseCache(CacheProfileName = "Default")]
    public IActionResult Index()
    {
        return View();
    }
}

```

```

[ResponseCache(Duration = 30)]
public class HomeController : Controller
{
    [ResponseCache(CacheProfileName = "Default")]
    public IActionResult Index()
    {
        return View();
    }
}

```

O `ResponseCache` atributo pode ser aplicado a ações (métodos) e controladores (classes). Atributos de nível de método substituem as configurações especificadas no nível de classe de atributos.

No exemplo acima, um atributo de nível de classe especifica uma duração de 30 segundos, enquanto um atributo de nível de método faz referência a um perfil de cache com uma duração definida como 60 segundos.

O cabeçalho resultante:

```
Cache-Control: public,max-age=60
```

## Recursos adicionais

- [Armazena as respostas em Caches](#)
- [Cache-Control](#)
- [Memória de cache no ASP.NET Core](#)
- [O cache no ASP.NET Core distribuído](#)
- [Detectar alterações com tokens de alteração no ASP.NET Core](#)
- [Middleware no ASP.NET Core de cache de resposta](#)

- [Auxiliar de Marca de Cache no ASP.NET Core MVC](#)
- [Auxiliar de Marca de Cache Distribuído no ASP.NET Core](#)

# Memória de cache no ASP.NET Core

12/02/2019 • 15 minutes to read • [Edit Online](#)

Por Rick Anderson, John Luo, e Steve Smith

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Noções básicas de cache

O cache pode melhorar significativamente o desempenho e a escalabilidade de um aplicativo, reduzindo o trabalho necessário para gerar o conteúdo. O armazenamento em cache funciona melhor com dados que raramente são alterados. O cache faz uma cópia dos dados que pode ser retornada muito mais rapidamente do que a partir da origem. Você deve escrever e testar seu aplicativo para que ele nunca dependa de dados armazenados em cache.

O ASP.NET Core dá suporte a vários caches diferentes. O cache mais simples se baseia no `IMemoryCache`, que representa um cache armazenado na memória do servidor web. Aplicativos que são executados em um farm de servidores devem garantir que as sessões sejam aderentes ao usar o cache na memória. Sessões aderentes garantem que todas as solicitações subsequentes de um cliente vão para o mesmo servidor. Por exemplo, uso de aplicativos da Web do Azure [Application Request Routing](#) (ARR) para rotear todas as solicitações subsequentes para o mesmo servidor.

Não adesivo sessões em um farm da web exigem uma [cache distribuído](#) para evitar problemas de consistência de cache. Para alguns aplicativos, um cache distribuído pode dar suporte a mais alta escalabilidade horizontal que um cache na memória. Usando um cache distribuído libera a memória de cache para um processo externo.

O cache `IMemoryCache` removerá entradas de cache sob pressão de memória, a menos que a [prioridade de cache](#) seja definida como `CacheItemPriority.NeverRemove`. Você pode definir o `CacheItemPriority` para ajustar a prioridade com que o cache remove itens sob pressão de memória.

O cache de memória pode armazenar qualquer objeto, enquanto a interface de cache distribuída é limitada a `byte[]`. Os itens de cache de armazenamento de cache na memória e distribuído como pares chave-valor.

## System.Runtime.Caching/MemoryCache

[System.Runtime.Caching/MemoryCache \(Pacote do NuGet\)](#) pode ser usado com:

- .NET standard 2.0 ou posterior.
- Qualquer [implementação do .NET](#) que tem como alvo o .NET Standard 2.0 ou posterior. Por exemplo, ASP.NET Core 2.0 ou posterior.
- .NET framework 4.5 ou posterior.

[Extensions / IMemoryCache](#) (descrita neste tópico) é mais recomendada `System.Runtime.Caching / MemoryCache` porque ele é melhor integrado ao ASP.NET Core. Por exemplo, `IMemoryCache` funciona nativamente com o ASP.NET Core [injeção de dependência](#).

Use `System.Runtime.Caching / MemoryCache` como uma ponte de compatibilidade ao portar código do ASP.NET 4.x ASP.NET Core.

## Diretrizes de cache

- Código deveria ter sempre uma opção de fallback para buscar dados e **não** dependem de um valor em

cache que está sendo disponível.

- O cache usa um recurso escasso, de memória. Limitar o crescimento de cache:
  - Fazer **não** usar entrada externo como chaves de cache.
  - Use as expirações para limitar o crescimento de cache.
  - [Usar SetSize, tamanho e SizeLimit para limitar o tamanho do cache](#)

## Usando IMemoryCache

O cache na memória é um *service* que é referenciado em seu aplicativo usando [injeção de dependência](#). Chame `AddMemoryCache` em `ConfigureServices`:

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.DependencyInjection;

public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMemoryCache();

        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }

    public void Configure(IApplicationBuilder app)
    {
        app.UseMvcWithDefaultRoute();
    }
}
```

Solicitar o `IMemoryCache` instância no construtor:

```
public class HomeController : Controller
{
    private IMemoryCache _cache;

    public HomeController(IMemoryCache memoryCache)
    {
        _cache = memoryCache;
    }
}
```

`IMemoryCache` requer o pacote NuGet [Extensions](#).

`IMemoryCache` requer o pacote NuGet [Extensions](#), que está disponível na [Microsoft.AspNetCore.All metapacote](#).

`IMemoryCache` requer o pacote NuGet [Extensions](#), que está disponível na [metapacote Microsoft](#).

O seguinte código usa [TryGetValue](#) para verificar se um horário está no cache. Se não estiver armazenado em cache um tempo, uma nova entrada é criada e adicionada ao cache com [definir](#).

```

public static class CacheKeys
{
    public static string Entry { get { return "_Entry"; } }
    public static string CallbackEntry { get { return "_Callback"; } }
    public static string CallbackMessage { get { return "_CallbackMessage"; } }
    public static string Parent { get { return "_Parent"; } }
    public static string Child { get { return "_Child"; } }
    public static string DependentMessage { get { return "_DependentMessage"; } }
    public static string DependentCTS { get { return "_DependentCTS"; } }
    public static string Ticks { get { return "_Ticks"; } }
    public static string CancelMsg { get { return "_CancelMsg"; } }
    public static string CancellationTokenSource { get { return "_CancelTokenSource"; } }
}

```

```

public IActionResult CacheTryGetValueSet()
{
    DateTime cacheEntry;

    // Look for cache key.
    if (!_cache.TryGetValue(CacheKeys.Entry, out cacheEntry))
    {
        // Key not in cache, so get data.
        cacheEntry = DateTime.Now;

        // Set cache options.
        var cacheEntryOptions = new MemoryCacheEntryOptions()
            // Keep in cache for this time, reset time if accessed.
            .SetSlidingExpiration(TimeSpan.FromSeconds(3));

        // Save data in cache.
        _cache.Set(CacheKeys.Entry, cacheEntry, cacheEntryOptions);
    }

    return View("Cache", cacheEntry);
}

```

A hora atual e o tempo em cache são exibidos:

```

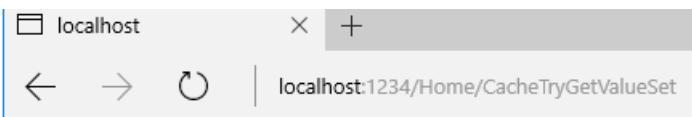
@model DateTime?

<div>
    <h2>Actions</h2>
    <ul>
        <li><a href="#" asp-controller="Home" asp-action="CacheTryGetValueSet">TryGetValue and Set</a></li>
        <li><a href="#" asp-controller="Home" asp-action="CacheGet">Get</a></li>
        <li><a href="#" asp-controller="Home" asp-action="CacheGetOrCreate">GetOrCreate</a></li>
        <li><a href="#" asp-controller="Home" asp-action="CacheGetOrCreateAsync">GetOrCreateAsync</a></li>
        <li><a href="#" asp-controller="Home" asp-action="CacheRemove">Remove</a></li>
    </ul>
</div>

<h3>Current Time: @DateTime.Now.ToString()</h3>
<h3>Cached Time: @(Model == null ? "No cached entry found" : Model.Value.ToString())</h3>

```

O valor `DateTime` em cache permanecerá no cache enquanto houver solicitações dentro do tempo limite (e nenhuma remoção devido à pressão de memória). A imagem abaixo mostra a hora atual e uma hora mais antiga recuperada do cache:



## Scenarios

- [Basic cache operations](#)
- [Cache entry with eviction callback](#)
- [Dependent cache entries](#)

## Actions

- [TryGetValue and Set](#)
- [Get](#)
- [GetOrCreate](#)
- [GetOrCreateAsync](#)
- [Remove](#)

**Current Time:** 17:04:01.1913080

**Cached Time:** 17:03:39.9454218

O código a seguir usa [GetOrCreate](#) e [GetOrCreateAsync](#) para fazer o cache dos dados.

```
public IActionResult CacheGetOrCreate()
{
    var cacheEntry = _cache.GetOrCreate(CacheKeys.Entry, entry =>
    {
        entry SlidingExpiration = TimeSpan.FromSeconds(3);
        return DateTime.Now;
    });

    return View("Cache", cacheEntry);
}

public async Task<IActionResult> CacheGetOrCreateAsync()
{
    var cacheEntry = await
        _cache.GetOrCreateAsync(CacheKeys.Entry, entry =>
    {
        entry SlidingExpiration = TimeSpan.FromSeconds(3);
        return Task.FromResult(DateTime.Now);
    });

    return View("Cache", cacheEntry);
}
```

O código a seguir chama o método [Get](#) para buscar a hora em cache:

```
public IActionResult CacheGet()
{
    var cacheEntry = _cache.Get<DateTime?>(CacheKeys.Entry);
    return View("Cache", cacheEntry);
}
```

[GetOrCreate](#) , [GetOrCreateAsync](#), e [Obtenha](#) fazem parte de métodos de extensão do [CacheExtensions](#) classe que estende a capacidade do [IMemoryCache](#). Ver [métodos IMemoryCache](#) e [CacheExtensions](#) métodos para obter uma descrição dos outros métodos de cache.

## MemoryCacheEntryOptions

O exemplo a seguir:

- Define o tempo de expiração absoluta. Isso é o tempo máximo que a entrada pode ser armazenado em cache e impede que o item se torne muito desatualizado quando a expiração deslizante continuamente é renovada.
- Define um tempo de expiração deslizante. Solicitações que acessam esse item em cache redefinirão o relógio de expiração deslizante.
- Define a prioridade de cache para `CacheItemPriority.NeverRemove`.
- Define uma `PostEvictionDelegate` que será chamado depois que a entrada é removida do cache. O retorno de chamada é executado em um thread diferente do código que remove o item do cache.

```
public IActionResult CreateCallbackEntry()
{
    var cacheEntryOptions = new MemoryCacheEntryOptions()
        // Pin to cache.
        .SetPriority(CacheItemPriority.NeverRemove)
        // Add eviction callback
        .RegisterPostEvictionCallback(callback: EvictionCallback, state: this);

    _cache.Set(CacheKeys.CallbackEntry, DateTime.Now, cacheEntryOptions);

    return RedirectToAction("GetCallbackEntry");
}

public IActionResult GetCallbackEntry()
{
    return View("Callback", new CallbackViewModel
    {
        CachedTime = _cache.Get<DateTime?>(CacheKeys.CallbackEntry),
        Message = _cache.Get<string>(CacheKeys.CallbackMessage)
    });
}

public IActionResult RemoveCallbackEntry()
{
    _cache.Remove(CacheKeys.CallbackEntry);
    return RedirectToAction("GetCallbackEntry");
}

private static void EvictionCallback(object key, object value,
    EvictionReason reason, object state)
{
    var message = $"Entry was evicted. Reason: {reason}.";
    ((HomeController)state)._cache.Set(CacheKeys.CallbackMessage, message);
}
```

## Usar SetSize, tamanho e SizeLimit para limitar o tamanho do cache

Um `MemoryCache` instância pode, opcionalmente, especificar e impor um limite de tamanho. O limite de tamanho de memória não tem uma unidade de medida definida porque o cache não tem nenhum mecanismo para medir o tamanho de entradas. Se o limite de tamanho de memória de cache for definido, todas as entradas devem especificar o tamanho. O tamanho especificado está em unidades que o desenvolvedor optar por.

Por exemplo:

- Se o aplicativo web foi principalmente cache de cadeias de caracteres, cada tamanho de entrada de cache pode ser o comprimento da cadeia de caracteres.
- O aplicativo pode especificar o tamanho de todas as entradas como 1 e o limite de tamanho é a contagem de entradas.

O código a seguir cria um sem unidade tamanho fixo `MemoryCache` acessível pelo [injeção de dependência](#):

```
// using Microsoft.Extensions.Caching.Memory;
public class MyMemoryCache
{
    public MemoryCache Cache { get; set; }
    public MyMemoryCache()
    {
        Cache = new MemoryCache(new MemoryCacheOptions
        {
            SizeLimit = 1024
        });
    }
}
```

`SizeLimit` não tem unidades. As entradas em cache devem especificar o tamanho em qualquer unidade que consideram mais apropriados se o tamanho da memória de cache tiver sido definido. Todos os usuários de uma instância de cache devem usar a mesma unidade de sistema. Uma entrada não será armazenada, se a soma dos tamanhos da entrada armazenada em cache excede o valor especificado pelo `SizeLimit`. Se nenhum limite de tamanho do cache for definido, o tamanho do cache definida na entrada será ignorado.

O código a seguir registra `MyMemoryCache` com o [injeção de dependência](#) contêiner.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    services.AddSingleton<MyMemoryCache>();
}
```

`MyMemoryCache` é criado como um cache de memória independentes para os componentes que reconhecem esse cache de tamanho limitado e saber como definir o tamanho do cache de entrada adequadamente.

O seguinte código usa `MyMemoryCache`:

```

public class AboutModel : PageModel
{
    private MemoryCache _cache;
    public static readonly string MyKey = "_MyKey";

    public AboutModel(MyMemoryCache memoryCache)
    {
        _cache = memoryCache.Cache;
    }

    [TempData]
    public string DateTime_Now { get; set; }

    public IActionResult OnGet()
    {
        if (!_cache.TryGetValue(MyKey, out string cacheEntry))
        {
            // Key not in cache, so get data.
            cacheEntry = DateTime.Now.TimeOfDay.ToString();

            var cacheEntryOptions = new MemoryCacheEntryOptions()
                // Set cache entry size by extension method.
                .SetSize(1)
                // Keep in cache for this time, reset time if accessed.
                .SetSlidingExpiration(TimeSpan.FromSeconds(3));

            // Set cache entry size via property.
            // cacheEntryOptions.Size = 1;

            // Save data in cache.
            _cache.Set(MyKey, cacheEntry, cacheEntryOptions);
        }

        DateTime_Now = cacheEntry;

        return RedirectToPage("./Index");
    }
}

```

O tamanho da entrada de cache pode ser definido [tamanho](#) ou o [SetSize](#) método de extensão:

```
public IActionResult OnGet()
{
    if (!_cache.TryGetValue(MyKey, out string cacheEntry))
    {
        // Key not in cache, so get data.
        cacheEntry = DateTime.Now.ToString();

        var cacheEntryOptions = new MemoryCacheEntryOptions()
            // Set cache entry size by extension method.
            .SetSize(1)
            // Keep in cache for this time, reset time if accessed.
            .SetSlidingExpiration(TimeSpan.FromSeconds(3));

        // Set cache entry size via property.
        // cacheEntryOptions.Size = 1;

        // Save data in cache.
        _cache.Set(MyKey, cacheEntry, cacheEntryOptions);
    }

    DateTime_Now = cacheEntry;

    return RedirectToPage("./Index");
}
```

## Dependências de cache

O exemplo a seguir mostra como expirar uma entrada de cache, se uma entrada dependente expira. Um `CancellationToken` é adicionado ao item em cache. Quando `Cancel` é chamado de `CancellationTokenSource`, ambas as entradas de cache são removidas.

```

public IActionResult CreateDependentEntries()
{
    var cts = new CancellationTokenSource();
    _cache.Set(CacheKeys.DependentCTS, cts);

    using (var entry = _cache.CreateEntry(CacheKeys.Parent))
    {
        // expire this entry if the dependant entry expires.
        entry.Value = DateTime.Now;
        entry.RegisterPostEvictionCallback(DependentEvictionCallback, this);

        _cache.Set(CacheKeys.Child,
            DateTime.Now,
            new CancellationChangeToken(cts.Token));
    }

    return RedirectToAction("GetDependentEntries");
}

public IActionResult GetDependentEntries()
{
    return View("Dependent", new DependentViewModel
    {
        ParentCachedTime = _cache.Get<DateTime?>(CacheKeys.Parent),
        ChildCachedTime = _cache.Get<DateTime?>(CacheKeys.Child),
        Message = _cache.Get<string>(CacheKeys.DependentMessage)
    });
}

public IActionResult RemoveChildEntry()
{
    _cache.Get<CancellationTokenSource>(CacheKeys.DependentCTS).Cancel();
    return RedirectToAction("GetDependentEntries");
}

private static void DependentEvictionCallback(object key, object value,
    EvictionReason reason, object state)
{
    var message = $"Parent entry was evicted. Reason: {reason}.";;
    ((HomeController)state)._cache.Set(CacheKeys.DependentMessage, message);
}

```

Usando um `CancellationTokenSource` permite que várias entradas de cache a ser removido como um grupo. Com o `using` padrão no código acima, as entradas de cache criado dentro de `using` bloco herdará as configurações de expiração e gatilhos.

## Observações adicionais

- Ao usar um retorno de chamada para preencher novamente um item de cache:
  - Várias solicitações podem localizar o valor da chave em cache vazio porque o retorno de chamada não foi concluído.
  - Isso pode resultar em vários threads repopulação o item em cache.
- Quando uma entrada de cache é usada para criar outro, o filho copia a entrada de pai tokens de expiração e as configurações de expiração com base no tempo. O filho não está expirada pela remoção manual ou a atualização da entrada pai.
- Use [PostEvictionCallbacks](#) para definir os retornos de chamada que serão acionados depois que a entrada de cache é removida do cache.

## Recursos adicionais

- O cache no ASP.NET Core distribuído
- Detectar alterações com tokens de alteração no ASP.NET Core
- Cache de resposta no ASP.NET Core
- Middleware no ASP.NET Core de cache de resposta
- Auxiliar de Marca de Cache no ASP.NET Core MVC
- Auxiliar de Marca de Cache Distribuído no ASP.NET Core

# O cache no ASP.NET Core distribuído

31/10/2018 • 13 minutes to read • [Edit Online](#)

Por [Steve Smith](#) e [Luke Latham](#)

Um cache distribuído é um cache compartilhado por vários servidores de aplicativo, normalmente é mantidos como um serviço externo para os servidores de aplicativo que acessá-lo. Um cache distribuído pode melhorar o desempenho e escalabilidade de um aplicativo ASP.NET Core, especialmente quando o aplicativo é hospedado por um serviço de nuvem ou um farm de servidores.

Um cache distribuído tem várias vantagens sobre outros cenários de cache onde os dados armazenados em cache são armazenados em servidores de aplicativos individuais.

Quando os dados armazenados em cache são distribuídos, os dados:

- Está *coerente* (consistente) em todas as solicitações para vários servidores.
- Sobrevive a reinicializações do servidor e as implantações de aplicativo.
- Não use memória local.

Configuração de cache distribuído é específico da implementação. Este artigo descreve como configurar o SQL Server e distribuídos caches Redis. Implementações de terceiros também estão disponíveis, como [NCache \(NCache no GitHub\)](#). Independentemente de qual implementação for selecionada, o aplicativo interage com o cache usando o [IDistributedCache](#) interface.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Pré-requisitos

Para usar um SQL Server distributed cache, a referência a [metapacote do Microsoft](#) ou adicionar uma referência de pacote para o [Microsoft.Extensions.Caching.SqlServer](#) pacote.

Para usar um Redis distributed cache, a referência a [metapacote do Microsoft](#) e adicione uma referência de pacote para o [Microsoft.Extensions.Caching.Redis](#) pacote. O pacote do Redis não está incluído no [Microsoft.AspNetCore.App](#) empacotar, portanto, você deve referenciar o pacote de Redis separadamente no seu arquivo de projeto.

Para usar um SQL Server distributed cache, a referência a [metapacote Microsoft.AspNetCore.All](#) ou adicionar uma referência de pacote para o [Microsoft.Extensions.Caching.SqlServer](#) pacote.

Para usar um Redis distributed cache, a referência a [metapacote Microsoft.AspNetCore.All](#) ou adicionar uma referência de pacote para o [Microsoft.Extensions.Caching.Redis](#) pacote. O pacote de Redis está incluído no [Microsoft.AspNetCore.All](#) empacotar, para que você não precisa fazer referência ao pacote Redis separadamente no seu arquivo de projeto.

Para usar um SQL Server distributed cache, adicione uma referência de pacote para o [Microsoft.Extensions.Caching.SqlServer](#) pacote.

Para usar um Redis cache distribuído, adicione uma referência de pacote para o [Microsoft.Extensions.Caching.Redis](#) pacote.

## Interface IDistributedCache

O [IDistributedCache](#) interface fornece os seguintes métodos para manipular itens na implementação de

cache distribuído:

- [Get](#), [GetAsync](#) – Aceita uma chave de cadeia de caracteres e recupera um item em cache como um `byte[]` matriz se encontrada no cache.
- [Set](#), [SetAsync](#) – Adiciona um item (como `byte[]` matriz) para o cache usando uma chave de cadeia de caracteres.
- [Refresh](#), [RefreshAsync](#) – Atualiza um item no cache com base em sua chave, redefinindo seu tempo limite de expiração deslizante (se houver).
- [Remove](#), [RemoveAsync](#) – Remove um item de cache com base em sua chave de cadeia de caracteres.

## Estabeleça serviços de cache distribuídos

Registrar uma implementação de [IDistributedCache](#) em `Startup.ConfigureServices`. Implementações fornecidos pela estrutura descritas neste tópico incluem:

- [Cache de memória distribuída](#)
- [Cache distribuído do SQL Server](#)
- [Cache distribuído do Redis](#)

### Cache de memória distribuída

O Cache de memória distribuída ([AddDistributedMemoryCache](#)) é uma implementação fornecidos pela estrutura de [IDistributedCache](#) que armazena os itens na memória. O Cache de memória distribuída não é um cache distribuído real. Itens armazenados em cache são armazenados por instância do aplicativo no servidor onde o aplicativo está sendo executado.

O Cache de memória distribuída é uma implementação útil:

- No desenvolvimento e cenários de teste.
- Quando um único servidor é usado em consumo de memória e de produção não é um problema. Implementar os resumos de Cache distribuído memória, armazenamento de dados em cache. Ele permite para implementar que uma verdadeira distribuída a solução de cache no futuro se vários nós ou tolerância a falhas que se tornar necessário.

O aplicativo de exemplo utiliza o Cache de memória distribuída quando o aplicativo é executado no ambiente de desenvolvimento:

```
public void ConfigureServices(IServiceCollection services)
{
    if (_hostContext.IsDevelopment())
    {
        services.AddDistributedMemoryCache();
    }
    else
    {
        services.AddDistributedSqlServerCache(options =>
        {
            options.ConnectionString =
                _config["DistCache_ConnectionString"];
            options.SchemaName = "dbo";
            options.TableName = "TestCache";
        });
    }
}
```

### Cache distribuído do SQL Server

A implementação de Cache distribuído do SQL Server ([AddDistributedSqlServerCache](#)) permite que o cache distribuído usar um banco de dados do SQL Server como seu armazenamento de backup. Para criar

uma tabela de item em cache do SQL Server em uma instância do SQL Server, você pode usar o `sql-cache` ferramenta. A ferramenta cria uma tabela com o nome e o esquema que você especificar.

Adicione `SqlConfig.Tools` para o `<ItemGroup>` elemento do arquivo de projeto e execute `dotnet restore`.

```
<ItemGroup>
  <DotNetCliToolReference Include="Microsoft.Extensions.Caching.SqlConfig.Tools"
    Version="2.0.2" />
</ItemGroup>
```

Crie uma tabela no SQL Server executando o `sql-cache create` comando. Fornecer a instância do SQL Server (`Data Source`), banco de dados (`Initial Catalog`), esquema (por exemplo, `dbo`) e o nome da tabela (por exemplo, `TestCache`):

```
dotnet sql-cache create "Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=DistCache;Integrated Security=True;" dbo TestCache
```

Uma mensagem é registrada para indicar que a ferramenta foi bem-sucedida:

```
Table and index were created successfully.
```

A tabela criada pelo `sql-cache` ferramenta tem o esquema a seguir:

	Name	Data Type	Allow Nulls
1	Id	nvarchar(449)	<input type="checkbox"/>
2	Value	varbinary(MAX)	<input type="checkbox"/>
3	ExpiresAtTime	datetimeoffset(7)	<input type="checkbox"/>
4	SlidingExpirationInSeconds	bigint	<input checked="" type="checkbox"/>
5	AbsoluteExpiration	datetimeoffset(7)	<input checked="" type="checkbox"/>

#### NOTE

Um aplicativo deve manipular valores de cache usando uma instância de [IDistributedCache](#), e não um [SqlServerCache](#).

O aplicativo de exemplo implementa [SqlServerCache](#) em um ambiente de desenvolvimento não:

```
public void ConfigureServices(IServiceCollection services)
{
    if (_hostContext.IsDevelopment())
    {
        services.AddDistributedMemoryCache();
    }
    else
    {
        services.AddDistributedSqlServerCache(options =>
        {
            options.ConnectionString =
                _config["DistCache_ConnectionString"];
            options.SchemaName = "dbo";
            options.TableName = "TestCache";
        });
    }
}
```

## NOTE

Um `ConnectionString` (e, opcionalmente, `SchemaName` e `TableName`) normalmente são armazenados fora do controle do código-fonte (por exemplo, são armazenados pela `Secret Manager` ou na `appSettings.json` / `appsettings.json`). A cadeia de caracteres de conexão pode conter credenciais que devem ser mantidas fora de sistemas de controle do código-fonte.

## Cache Redis distribuído

Redis é um repositório de dados na memória do código-fonte aberto, que geralmente é usado como um cache distribuído. Você pode usar o Redis localmente, e você pode configurar uma [Cache Redis do Azure](#) para um aplicativo ASP.NET Core hospedados no Azure. Um aplicativo configura a implementação de cache usando um `RedisCache` instância (`AddDistributedRedisCache`):

```
services.AddDistributedRedisCache(options =>
{
    options.Configuration = "localhost";
    options.InstanceName = "SampleInstance";
});
```

Para instalar o Redis em seu computador local:

- Instalar o [Redis Chocolatey pacote](#).
- Executar `redis-server` em um prompt de comando.

## Usar o cache distribuído

Para usar o `IDistributedCache` interface, solicitar uma instância de `IDistributedCache` de qualquer construtor no aplicativo. A instância é fornecida pela [injeção de dependência \(DI\)](#).

Quando o aplicativo é iniciado, `IDistributedCache` é injetado no `Startup.Configure`. A hora atual é armazenado em cache usando `IApplicationLifetime` (para obter mais informações, consulte [Host da Web: interface IApplicationLifetime](#)):

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
    IApplicationLifetime lifetime, IDistributedCache cache)
{
    lifetime.ApplicationStarted.Register(() =>
    {
        var currentTimeUTC = DateTime.UtcNow.ToString();
        byte[] encodedcurrentTimeUTC = Encoding.UTF8.GetBytes(currentTimeUTC);
        var options = new DistributedCacheEntryOptions()
            .SetSlidingExpiration(TimeSpan.FromSeconds(20));
        cache.Set("cachedTimeUTC", encodedcurrentTimeUTC, options);
    });
}
```

O aplicativo de exemplo injeta `IDistributedCache` para o `IndexModel` para uso pela página de índice.

Cada vez que a página de índice é carregada, o cache é verificado para o tempo em cache em `OnGetAsync`. Se ainda não tiver expirado o tempo em cache, a hora é exibida. Se 20 segundos decorridos desde a última vez em que o tempo em cache foi acessado (a última vez que esta página foi carregada), a página exibe *armazenado em cache tempo expirado*.

Atualizar imediatamente o tempo em cache para a hora atual, selecionando o **redefinição de tempo em cache** botão. Os gatilhos de botão a `OnPostResetCachedTime` método do manipulador.

```

public class IndexModel : PageModel
{
    private readonly IDistributedCache _cache;

    public IndexModel(IDistributedCache cache)
    {
        _cache = cache;
    }

    public string CachedTimeUTC { get; set; }

    public async Task OnGetAsync()
    {
        CachedTimeUTC = "Cached Time Expired";
        var encodedCachedTimeUTC = await _cache.GetAsync("cachedTimeUTC");

        if (encodedCachedTimeUTC != null)
        {
            CachedTimeUTC = Encoding.UTF8.GetString(encodedCachedTimeUTC);
        }
    }

    public async Task<IActionResult> OnPostResetCachedTime()
    {
        var currentTimeUTC = DateTime.UtcNow.ToString();
        byte[] encodedcurrentTimeUTC = Encoding.UTF8.GetBytes(currentTimeUTC);
        var options = new DistributedCacheEntryOptions()
            .SetSlidingExpiration(TimeSpan.FromSeconds(20));
        await _cache.SetAsync("cachedTimeUTC", encodedcurrentTimeUTC, options);

        return RedirectToPage();
    }
}

```

#### NOTE

Não é necessário usar um tempo de vida Singleton ou com escopo para `IDistributedCache` instâncias (pelo menos para as implementações internas).

Você também pode criar uma `IDistributedCache` da instância onde você pode precisar de uma em vez de usar a DI, mas a criação de uma instância no código pode tornar seu código mais difícil de testar e viola o [princípio de dependências explícitas](#).

## Recomendações

Ao decidir qual implementação de `IDistributedCache` é melhor para seu aplicativo, considere o seguinte:

- Infraestrutura existente
- Requisitos de desempenho
- Custo
- Experiência da equipe

Soluções de cache normalmente se baseiam em armazenamento na memória para fornecer recuperação rápida de dados armazenados em cache, mas a memória é um recurso limitado e caro expandir. Loja apenas dados usados normalmente em um cache.

Em geral, um cache Redis fornece maior taxa de transferência e latência mais baixa que o cache de SQL Server. No entanto, é geralmente necessário para determinar as características de desempenho de estratégias de cache de benchmark.

Quando o SQL Server é usado como um armazenamento de backup de cache distribuído, usar o mesmo banco de dados para o cache e armazenamento de dados comum do aplicativo e recuperação pode afetar negativamente o desempenho de ambos. É recomendável usar uma instância dedicada do SQL Server para o cache distribuído do repositório de backup.

## Recursos adicionais

- [Redis Cache no Azure](#)
- [Banco de dados SQL no Azure](#)
- [Provedor de IDistributedCache para NCache nos Farms da Web do ASP.NET Core \(NCache no GitHub\)](#)
- [Memória de cache no ASP.NET Core](#)
- [Detectar alterações com tokens de alteração no ASP.NET Core](#)
- [Cache de resposta no ASP.NET Core](#)
- [Middleware no ASP.NET Core de cache de resposta](#)
- [Auxiliar de Marca de Cache no ASP.NET Core MVC](#)
- [Auxiliar de Marca de Cache Distribuído no ASP.NET Core](#)
- [Hospedar o ASP.NET Core em um web farm](#)

# Middleware no ASP.NET Core de cache de resposta

30/10/2018 • 13 minutes to read • [Edit Online](#)

Por [Luke Latham](#) e [John Luo](#)

[Exibir ou baixar um código de exemplo \(como baixar\).](#)

Este artigo explica como configurar o Middleware de cache de resposta em um aplicativo ASP.NET Core. O middleware determina quando as respostas são armazenáveis em cache, as respostas de repositórios e respostas de servir do cache. Para obter uma introdução ao cache de HTTP e o `ResponseCache` atributo, consulte [cache de resposta](#).

## Pacote

Referência a [metapacote do Microsoft](#) ou adicionar uma referência de pacote para o `Microsoft.AspNetCore.ResponseCaching` pacote.

Referência a [metapacote Microsoft.AspNetCore.All](#) ou adicionar uma referência de pacote para o `Microsoft.AspNetCore.ResponseCaching` pacote.

Adicionar uma referência de pacote para o `Microsoft.AspNetCore.ResponseCaching` pacote.

## Configuração

No `Startup.ConfigureServices`, adicione o middleware para a coleção de serviço.

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddResponseCaching();
    services.AddMvc()
        .SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
}
```

Configurar o aplicativo para usar o middleware com o `UseResponseCaching` método de extensão, que adiciona o middleware ao pipeline de processamento da solicitação. O aplicativo de exemplo adiciona uma `Cache-Control` cabeçalho à resposta que armazena em cache respostas armazenáveis em cache por até 10 segundos. O exemplo envia uma `Vary` cabeçalho para configurar o middleware para servir a uma resposta em cache somente se o `Accept-Encoding` cabeçalho das solicitações subsequentes corresponde da solicitação original. No exemplo de código a seguir, `CacheControlHeaderValue` e `HeaderNames` exigir uma `using` instrução para o `Microsoft.Net.Http.Headers` namespace.

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseCookiePolicy();

    app.UseResponseCaching();

    app.Use(async (context, next) =>
    {
        // For GetTypedHeaders, add: using Microsoft.AspNetCore.Http;
        context.Response.GetTypedHeaders().CacheControl =
            new Microsoft.Net.Http.Headers.CacheControlHeaderValue()
        {
            Public = true,
            MaxAge = TimeSpan.FromSeconds(10)
        };
        context.Response.Headers[Microsoft.Net.Http.Headers.HeaderNames.Vary] =
            new string[] { "Accept-Encoding" };

        await next();
    });
}

app.UseMvc();
}

```

Middleware de cache de resposta só armazena em cache as respostas do servidor que resultam em um código de status 200 (Okey). Outras respostas, incluindo [páginas de erro](#), são ignorados pelo middleware.

#### **WARNING**

Respostas que contêm o conteúdo para clientes autenticados devem ser marcadas como não armazenável em cache para impedir que o middleware de armazenamento e que atende a essas respostas. Ver [condições para armazenar em cache](#) para obter detalhes sobre como o middleware determina se uma resposta é armazenável em cache.

## Opções

O middleware oferece três opções para controlar o cache de resposta.

OPÇÃO	DESCRIÇÃO
UseCaseSensitivePaths	Determina se as respostas são armazenadas em cache em caminhos diferencia maiusculas de minúsculas. O valor padrão é <code>false</code> .
MaximumBodySize	O maior tamanho armazenáveis em cache para o corpo de resposta em bytes. O valor padrão é <code>64 * 1024 * 1024</code> (64 MB).

OPÇÃO	DESCRIÇÃO
SizeLimit	O limite de tamanho para o middleware de cache de resposta em bytes. O valor padrão é <code>100 * 1024 * 1024</code> (100 MB).

O exemplo a seguir configura o middleware para:

- Respostas em cache menores ou iguais a 1.024 bytes.
- Store as respostas por caminhos diferencia maiusculas de minúsculas (por exemplo, `/page1` e `/Page1` são armazenadas separadamente).

```
services.AddResponseCaching(options =>
{
    options.UseCaseSensitivePaths = true;
    options.MaximumBodySize = 1024;
});
```

## VaryByQueryKeys

Ao usar controladores de API da Web do MVC ou modelos de página de páginas do Razor, o `ResponseCache` atributo especifica os parâmetros necessários para definir os cabeçalhos apropriados para o cache de resposta. O único parâmetro do `ResponseCache` atributo que estritamente requer o middleware é `VaryByQueryKeys`, que não corresponde a um cabeçalho HTTP real. Para obter mais informações, consulte [o atributo ResponseCache](#).

Quando não estiver usando o `ResponseCache` atributo, cache de resposta pode ser variado com o `VaryByQueryKeys` recurso. Use o `ResponseCachingFeature` diretamente do `IFeatureCollection` da `HttpContext`:

```
var responseCachingFeature = context.HttpContext.Features.Get<IResponseCachingFeature>();
if (responseCachingFeature != null)
{
    responseCachingFeature.VaryByQueryKeys = new[] { "MyKey" };
}
```

Usando um único valor igual a `*` em `VaryByQueryKeys` o cache de varia de acordo com todos os parâmetros de consulta de solicitação.

## Cabeçalhos HTTP usados pelo Middleware de cache de resposta

Cache de resposta pelo middleware é configurado usando cabeçalhos HTTP.

CABEÇALHO	DETALHES
Autorização	A resposta não é armazenado em cache se o cabeçalho existe.

CABEÇALHO	DETALHES
Cache-Control	<p>O middleware considera somente o cache de respostas marcadas com o <code>public</code> diretiva de cache. Controlar o cache com os seguintes parâmetros:</p> <ul style="list-style-type: none"> <li>• idade máxima</li> <li>• <code>max-stale</code><sup>†</sup></li> <li>• nova min</li> <li>• deve revalidar</li> <li>• sem cache</li> <li>• Nenhum repositório</li> <li>• somente se-armazenado em cache</li> <li>• particulares</li> <li>• públicos</li> <li>• período máximo s</li> <li>• <code>proxy-revalidate</code><sup>‡</sup></li> </ul> <p><sup>†</sup>Se nenhum limite é especificado para <code>max-stale</code>, o middleware não toma nenhuma ação.  <sup>‡</sup> <code>proxy-revalidate</code> tem o mesmo efeito que <code>must-revalidate</code>.</p> <p>Para obter mais informações, consulte <a href="#">RFC 7231: solicitação de diretivas de Cache-Control</a>.</p>
Pragma	Um <code>Pragma: no-cache</code> cabeçalho na solicitação produz o mesmo efeito que <code>Cache-Control: no-cache</code> . Esse cabeçalho é substituído por diretivas relevantes no <code>Cache-Control</code> cabeçalho, se presente. Considerado para compatibilidade com versões anteriores com HTTP 1.0.
Set-Cookie	A resposta não é armazenado em cache se o cabeçalho existe. Qualquer middleware no pipeline de processamento de solicitação que define um ou mais cookies impede que o Middleware de cache de resposta de cache a resposta (por exemplo, o <a href="#">provedor de TempData baseado em cookie</a> ).
Variar	O <code>Vary</code> cabeçalho é usado para variar a resposta em cache por outro cabeçalho. Por exemplo, armazenar em cache respostas de codificação, incluindo <code>Vary: Accept-Encoding</code> cabeçalho, que armazena em cache respostas para solicitações com cabeçalhos <code>Accept-Encoding: gzip</code> e <code>Accept-Encoding: text/plain</code> separadamente. Uma resposta com um valor de cabeçalho de <code>*</code> nunca é armazenada.
Expira	Uma resposta considerada obsoleta por esse cabeçalho não é armazenada ou recuperada, a menos que substituído por outro <code>Cache-Control</code> cabeçalhos.
If-None-Match	A resposta completa for atendida do cache se o valor não for <code>*</code> e o <code>ETag</code> da resposta não coincide com nenhum dos valores fornecidos. Caso contrário, uma resposta 304 (não modificado) é atendido.

CABEÇALHO	DETALHES
If-Modified-Since	Se o <code>If-None-Match</code> cabeçalho não estiver presente, uma resposta completa for atendida do cache se a data de resposta em cache é mais recente do que o valor fornecido. Caso contrário, uma resposta 304 (não modificado) é atendido.
Date	Quando atendendo do cache, o <code>Date</code> cabeçalho é definido pelo middleware se ele não foi fornecido na resposta original.
Tamanho do conteúdo	Quando atendendo do cache, o <code>Content-Length</code> cabeçalho é definido pelo middleware se ele não foi fornecido na resposta original.
Idade	O <code>Age</code> cabeçalho enviado na resposta original será ignorado. O middleware calcula um novo valor ao oferecer uma resposta em cache.

## Armazenamento em cache respeita as diretivas de solicitação de Cache-Control

O middleware respeita as regras da [especificação de cache do HTTP 1.1](#). As regras exigem um cache para honrar válido `Cache-Control` cabeçalho enviado pelo cliente. Sob a especificação, um cliente pode fazer solicitações com um `no-cache` que força o servidor gere uma nova resposta para cada solicitação e o valor do cabeçalho. Atualmente, não há nenhum controle do desenvolvedor sobre o comportamento de cache ao usar o middleware porque o middleware segue a especificação oficial do cache.

Para obter mais controle sobre o comportamento de cache, explore outros recursos de cache do ASP.NET Core. Confira os seguintes tópicos:

- [Memória de cache no ASP.NET Core](#)
- [O cache no ASP.NET Core distribuído](#)
- [Auxiliar de Marca de Cache no ASP.NET Core MVC](#)
- [Auxiliar de Marca de Cache Distribuído no ASP.NET Core](#)

## Solução de problemas

Se o comportamento de cache não é conforme o esperado, confirme que as respostas sejam armazenável em cache e capaz de serem servidas do cache. Examine os cabeçalhos de entrada da solicitação e cabeçalhos de saída da resposta. Habilite [registro em log](#) para ajudar na depuração.

Ao testar e solucionar problemas de comportamento de cache, um navegador pode definir cabeçalhos de solicitação que afetam o cache de maneira indesejada. Por exemplo, um navegador pode definir a `Cache-Control` cabeçalho `no-cache` ou `max-age=0` durante a atualização de uma página. As ferramentas a seguir podem definir explicitamente os cabeçalhos de solicitação e são preferenciais para testes de armazenamento em cache:

- [Fiddler](#)
- [Postman](#)

### Condições para armazenar em cache

- A solicitação deve resultar em uma resposta do servidor com um código de status 200 (OK).

- O método de solicitação deve ser GET ou HEAD.
- Middleware de terminal, como [Middleware de arquivo estático](#), não deve processar a resposta antes do Middleware de cache de resposta.
- O `Authorization` cabeçalho não deverá estar presente.
- `Cache-Control` parâmetros do cabeçalho devem ser válidos, e a resposta deve ser marcada `public` e não marcada `private`.
- O `Pragma: no-cache` cabeçalho não deverá estar presente se o `Cache-Control` cabeçalho não estiver presente, como o `Cache-Control` cabeçalho substitui o `Pragma` cabeçalho quando presentes.
- O `Set-Cookie` cabeçalho não deverá estar presente.
- `Vary` parâmetros do cabeçalho devem ser válido e não é igual a `*`.
- O `Content-Length` valor de cabeçalho (se definido) deve corresponder ao tamanho do corpo da resposta.
- O [IHttpSendFileFeature](#) não é usado.
- A resposta não deve ser obsoleta conforme especificado pelo `Expires` cabeçalho e o `max-age` e `s-maxage` diretivas de cache.
- Buffer de resposta deve ser bem-sucedida e o tamanho da resposta deve ser menor do que o configurado ou padrão `SizeLimit`.
- A resposta deve ser armazenáveis em cache de acordo com o [RFC 7234](#) especificações. Por exemplo, o `no-store` diretiva não deve existir nos campos de cabeçalho de solicitação ou resposta. Ver [seção 3: armazenar respostas em Caches](#) dos [RFC 7234](#) para obter detalhes.

#### **NOTE**

O sistema Antifalsificação para gerar tokens de seguras para evitar a falsificação de solicitação entre sites (CSRF) attacks conjuntos a `Cache-Control` e `Pragma` cabeçalhos para `no-cache` para que as respostas não são armazenadas em cache. Para obter informações sobre como desabilitar tokens antifalsificação para elementos de formulário HTML, consulte [configuração antifalsificação do ASP.NET Core](#).

## Recursos adicionais

- [Inicialização de aplicativo no ASP.NET Core](#)
- [Middleware do ASP.NET Core](#)
- [Memória de cache no ASP.NET Core](#)
- [O cache no ASP.NET Core distribuído](#)
- [Detectar alterações com tokens de alteração no ASP.NET Core](#)
- [Cache de resposta no ASP.NET Core](#)
- [Auxiliar de Marca de Cache no ASP.NET Core MVC](#)
- [Auxiliar de Marca de Cache Distribuído no ASP.NET Core](#)

# Compactação de resposta no ASP.NET Core

08/01/2019 • 23 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

[Exibir ou baixar código de exemplo \(como baixar\)](#)

Largura de banda de rede é um recurso limitado. Reduzindo o tamanho da resposta geralmente aumenta a capacidade de resposta de um aplicativo, muitas vezes drasticamente. É uma maneira de reduzir os tamanhos do conteúdo compactar respostas do aplicativo.

## Quando usar o Middleware de compactação de resposta

Use as tecnologias de compactação de resposta com base em servidor no IIS, Apache ou Nginx. O desempenho do middleware provavelmente não corresponderá dos módulos de servidor. [Servidor HTTP.sys](#) servidor e [Kestrel](#) server atualmente não oferecem suporte à compactação interna.

Use o Middleware de compactação de resposta quando você estiver:

- Não é possível usar as seguintes tecnologias de compactação baseada em servidor:
  - [Módulo de compactação dinâmica do IIS](#)
  - [Módulo do Apache mod\\_deflate](#)
  - [Nginx compactação e descompactação](#)
- Hospedagem diretamente em:
  - [Servidor HTTP.sys](#) (anteriormente chamado WebListener)
  - [Servidor kestrel](#)

## Compactação de resposta

Geralmente, nenhuma resposta compactada nativamente não pode se beneficiar da compactação de resposta. As respostas não nativamente compactadas normalmente incluem: CSS, JavaScript, HTML, XML e JSON. Você não deve compactar os ativos nativamente compactados, como arquivos PNG. Se você tentar compactar ainda mais uma resposta compactada nativamente, qualquer redução adicional pequena em tempo de tamanho e a transmissão será provavelmente ser superada pelo tempo levado para processar a compactação. Não compacte arquivos menores do que cerca de 150 e 1000 bytes (dependendo do conteúdo do arquivo e a eficiência da compactação). A sobrecarga de compactação de arquivos pequenos pode produzir um arquivo compactado maior do que o arquivo descompactado.

Quando um cliente pode processar o conteúdo compactado, o cliente deve informar o servidor de seus recursos, enviando o `Accept-Encoding` cabeçalho com a solicitação. Quando um servidor envia o conteúdo compactado, ele deve incluir informações no `Content-Encoding` cabeçalho em como a resposta compactada é codificada. Conteúdo designações de codifica com suporte pelo middleware são mostradas na tabela a seguir.

ACCEPT-ENCODING	VALORES DE CABEÇALHO	MIDDLEWARE COM SUPORTE	DESCRIÇÃO
<code>br</code>		Sim (padrão)	<a href="#">Formato de dados compactados Brotli</a>
<code>deflate</code>		Não	<a href="#">Formato de dados compactados DEFLATE</a>

ACCEPT-ENCODING	VALORES DE CABEÇALHO	MIDDLEWARE COM SUPORTE	DESCRIÇÃO
<code>exi</code>		Não	Intercâmbio de eficiente de XML do W3C
<code>gzip</code>		Sim	Formato de arquivo gzip
<code>identity</code>		Sim	Identificador "Nenhuma codificação": A resposta não deve ser codificada.
<code>pack200-gzip</code>		Não	Formato de transferência de rede para arquivos mortos de Java
<code>*</code>		Sim	Qualquer conteúdo disponível não codificação explicitamente solicitada

ACCEPT-ENCODING	VALORES DE CABEÇALHO	MIDDLEWARE COM SUPORTE	DESCRIÇÃO
<code>br</code>		Não	Formato de dados compactados Brotli
<code>deflate</code>		Não	Formato de dados compactados DEFLATE
<code>exi</code>		Não	Intercâmbio de eficiente de XML do W3C
<code>gzip</code>		Sim (padrão)	Formato de arquivo gzip
<code>identity</code>		Sim	Identificador "Nenhuma codificação": A resposta não deve ser codificada.
<code>pack200-gzip</code>		Não	Formato de transferência de rede para arquivos mortos de Java
<code>*</code>		Sim	Qualquer conteúdo disponível não codificação explicitamente solicitada

Para obter mais informações, consulte o [IANA lista oficial de conteúdo de codificação](#).

O middleware permite que você adicione provedores de compactação adicional para custom `Accept-Encoding` valores de cabeçalho. Para obter mais informações, consulte [provedores personalizados](#) abaixo.

O middleware é capaz de reagir a valor de qualidade (qvalue, `q`) quando enviado pelo cliente para priorizar os esquemas de compactação de ponderação. Para obter mais informações, consulte [RFC 7231: Codificação aceita](#).

Algoritmos de compactação estão sujeitos a uma compensação entre a velocidade de compactação e a eficiência da compactação. *Eficácia* neste contexto refere-se ao tamanho da saída após a compactação. O menor tamanho é obtido com a maioria *ideal* compactação.

Cabeçalhos envolvidos na solicitação, enviar, armazenamento em cache e receber conteúdo compactado são descritas na tabela a seguir.

CABEÇALHO	FUNÇÃO
-----------	--------

CABEÇALHO	FUNÇÃO
Accept-Encoding	Enviada do cliente para o servidor para indicar a codificação esquemas aceitáveis para o cliente de conteúdo.
Content-Encoding	Enviados do servidor para o cliente para indicar a codificação do conteúdo na carga.
Content-Length	Quando ocorre a compressão, a Content-Length cabeçalho for removido, desde que as alterações de conteúdo do corpo quando a resposta é compactada.
Content-MD5	Quando ocorre a compressão, a Content-MD5 cabeçalho for removido, uma vez que o conteúdo do corpo foi alterado e o hash não é mais válido.
Content-Type	Especifica o tipo MIME do conteúdo. Cada resposta deve especificar seu Content-Type. O middleware verifica esse valor para determinar se a resposta deve ser compactada. O middleware Especifica um conjunto de <a href="#">padrão de tipos MIME</a> que ele pode codificar, mas você pode substituir ou adicionar tipos MIME.
Vary	Quando enviadas pelo servidor com um valor de Accept-Encoding para clientes e proxies, o Vary cabeçalho indica para o cliente ou um proxy que ele deve armazenar em cache (variar) respostas com base no valor da Accept-Encoding cabeçalho da solicitação. O resultado de retorno de conteúdo com o Vary: Accept-Encoding cabeçalho é que ambos compactados e descompactadas respostas são armazenadas em cache separadamente.

Explore os recursos do Middleware de compactação de resposta com o [aplicativo de exemplo](#). O exemplo ilustra:

- A compactação de respostas do aplicativo usando o Gzip e provedores de compactação personalizado.
- Como adicionar um tipo de MIME para a lista padrão de tipos MIME para compactação.

## Pacote

Para incluir o middleware em um projeto, adicione uma referência para o [metapacote do Microsoft](#), que inclui o [Microsoft.AspNetCore.ResponseCompression](#) pacote.

Para incluir o middleware em um projeto, adicione uma referência para o [metapacote Microsoft.AspNetCore.All](#), que inclui o [Microsoft.AspNetCore.ResponseCompression](#) pacote.

Para incluir o middleware em um projeto, adicione uma referência para o [Microsoft.AspNetCore.ResponseCompression](#) pacote.

## Configuração

O código a seguir mostra como habilitar o Middleware de compactação de resposta para provedores de compactação e tipos MIME padrão (Brotli e Gzip):

O código a seguir mostra como habilitar o Middleware de compactação de resposta para tipos MIME padrão e o provedor de compactação Gzip:

```

public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddResponseCompression();
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.UseResponseCompression();
    }
}

```

Notas:

- `app.UseResponseCompression` deve ser chamado antes de `app.UseMvc`.
- Usar uma ferramenta como [Fiddler](#), [Firebug](#), ou [Postman](#) para definir o `Accept-Encoding` cabeçalho de solicitação e estudar os cabeçalhos de resposta, o tamanho e o corpo.

Enviar uma solicitação para o aplicativo de exemplo sem o `Accept-Encoding` cabeçalho e observe que a resposta é descompactada. O `Content-Encoding` e `Vary` cabeçalhos não estiverem presentes na resposta.

The screenshot shows a Fiddler session with the following details:

- Request Headers:**
  - Client: User-Agent: Fiddler
  - Transport: Host: localhost:5000
- Response Headers:**
  - HTTP/1.1 200 OK
  - Date: Wed, 11 Jan 2017 18:30:11 GMT
  - Content-Type: text/plain
  - Server: Kestrel
  - Content-Length: 2032
- Response Body:**

The body contains a large amount of placeholder text (Lorem ipsum) which is displayed in its entirety due to the lack of compression. A red box highlights the entire text block.

Enviar uma solicitação para o aplicativo de exemplo com o `Accept-Encoding: br` cabeçalho (a compactação Brotli) e observe que a resposta é compactada. O `Content-Encoding` e `Vary` cabeçalhos estão presentes na resposta.

Headers | TextView | SyntaxView | WebForms | HexView | Auth | Cookies | Raw | JSON | XML |

### Request Headers

GET / HTTP/1.1

#### Client

Accept-Encoding: br  
User-Agent: Fiddler

#### Transport

Host: localhost:5000

Transformer | Headers | TextView | SyntaxView | ImageView | HexView | WebView | Auth | Caching | Cookies | Raw

```
HTTP/1.1 200 OK
Date: Wed, 03 Oct 2018 17:16:21 GMT
Content-Type: text/plain
Server: Kestrel
Transfer-Encoding: chunked
Content-Encoding: br
Vary: Accept-Encoding
```

3c1

... (Large amount of compressed binary data)

Enviar uma solicitação para o aplicativo de exemplo com o `Accept-Encoding: gzip` cabeçalho e observe que a resposta é compactada. O `Content-Encoding` e `Vary` cabeçalhos estão presentes na resposta.

Headers | TextView | WebForms | HexView | Auth | Cookies | Raw | JSON | XML |

**Request Headers**

GET / HTTP/1.1

**Client**

Accept-Encoding: gzip  
User-Agent: Fiddler

**Transport**

Host: localhost:5000

Response body is encoded. Click to decode.

Get SyntaxView Transformer Headers TextView ImageView HexView WebView Auth Caching Cookies Raw

```
HTTP/1.1 200 OK
Date: Wed, 11 Jan 2017 18:24:08 GMT
Content-Type: text/plain
Server: Kestrel
Transfer-Encoding: chunked
Content-Encoding: gzip
Vary: Accept-Encoding
```

a  
00000000  
384  
0U.9  
00}00d0200000^00000G]K\*000rUW0Y0n0G05000  
0-000  
XH00V000Z \ycI\@0500Z00  
00\$00000"0000000 X0QY00m\0000vNL0Bk80D00Q0200je000V0000R00000  
0`0;000\*0u\_0a00000:@000{0  
[0<0000\*%\_0j000\$~0r00u80Fu00a00F}0SXPY0(PG000/\h0000N00(Sgy7000120n00500I000s" | 000\*#0z0~0009`o0tC00  
0000;iGP\*[000w[, [00000a0#0010\*fD0cJ>~000是000Vn@0\_0VX000  
00i+f000^00=M3n0  
000x0!000.ж0&F0 00>YG00300Pdy0[000&(0' 000vg0R0EY 000z000k000\*0lyI0000~t0yE0X00+v00j00v0i00S>00b0  
000do'np0q0vt0(010Nz/0000Y000#  
004w00S0000G;00=050080000, 0&0000@  
0Yn?00yY0G0x00000\_0F0j0000ET0rXY.00V0?^00sX01/ш60&000K7ST0000d00;0000000f0ut00[000s0m0eL^0000+0k0+qu0  
0000K0000i000q0rM0ZUX}0000)Q0<L0^<0t0]0h0:7vG?0Q\_ofF00u0~S0id0Q000z1hn0 00  
0

## Provedores

### Provedor de compactação Brotli

Use o `BrotliCompressionProvider` para compactar respostas com o formato de dados compactados Brotli.

Se nenhum provedor de compactação é adicionado explicitamente a `CompressionProviderCollection`:

- O provedor de compactação Brotli é adicionado por padrão para a matriz de provedores de compactação juntamente com o [provedor de compactação Gzip](#).
- Padrões de compactação para a compactação Brotli quando o formato de dados compactados Brotli é suportado pelo cliente. Se Brotli não é suportada pelo cliente, a compactação padrão Gzip quando o cliente oferece suporte à compactação Gzip.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCompression();
}
```

O provedor de compactação Brotli devem ser adicionado ao quaisquer provedores de compactação são adicionados explicitamente:

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCompression(options =>
    {
        options.Providers.Add<BrotliCompressionProvider>();
        options.Providers.Add<GzipCompressionProvider>();
        options.Providers.Add<CustomCompressionProvider>();
        options.MimeTypes =
            ResponseCompressionDefaults.MimeTypes.Concat(
                new[] { "image/svg+xml" });
    });
}

```

Definir a compactação de nível com `BrotliCompressionProviderOptions`. O provedor de compactação Brotli assume como padrão o nível de compactação mais rápido (`CompressionLevel.Fastest`), que não pode produzir a compactação mais eficiente. Se a compactação mais eficiente é desejada, configure o middleware de compactação ideal.

NÍVEL DE COMPACTAÇÃO	DESCRIÇÃO
<code>CompressionLevel.Fastest</code>	A compactação deve ser concluída assim que possível, mesmo se a saída resultante não é compactada da maneira ideal.
<code>CompressionLevel.NoCompression</code>	Nenhuma compactação deve ser executada.
<code>CompressionLevel.Optimal</code>	As respostas devem ser compactadas ideal, mesmo se a compactação leva mais tempo para concluir.

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCompression();

    services.Configure<BrotliCompressionProviderOptions>(options =>
    {
        options.Level = CompressionLevel.Fastest;
    });
}

```

## Provedor de compactação gzip

Use o `GzipCompressionProvider` para compactar respostas com o formato de arquivo `Gzip`.

Se nenhum provedor de compactação é adicionado explicitamente a `CompressionProviderCollection`:

- O provedor de compactação Gzip é adicionado por padrão para a matriz de provedores de compactação juntamente com o [provedor de compactação Brotli](#).
- Padrões de compactação para a compactação Brotli quando o formato de dados compactados Brotli é suportado pelo cliente. Se Brotli não é suportado pelo cliente, a compactação padrão Gzip quando o cliente oferece suporte à compactação Gzip.
- O provedor de compactação Gzip é adicionado por padrão para a matriz de provedores de compactação.
- Padrões de compactação como Gzip, quando o cliente oferece suporte à compactação Gzip.

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCompression();
}

```

O provedor de compactação Gzip devem ser adicionado ao quaisquer provedores de compactação são adicionados explicitamente:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCompression(options =>
    {
        options.Providers.Add<BrotliCompressionProvider>();
        options.Providers.Add<GzipCompressionProvider>();
        options.Providers.Add<CustomCompressionProvider>();
        options.MimeTypes =
            ResponseCompressionDefaults.MimeTypes.Concat(
                new[] { "image/svg+xml" });
    });
}
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCompression(options =>
    {
        options.Providers.Add<GzipCompressionProvider>();
        options.Providers.Add<CustomCompressionProvider>();
        options.MimeTypes =
            ResponseCompressionDefaults.MimeTypes.Concat(
                new[] { "image/svg+xml" });
    });
}
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCompression(options =>
    {
        options.Providers.Add<GzipCompressionProvider>();
        options.Providers.Add<CustomCompressionProvider>();
        options.MimeTypes =
            ResponseCompressionDefaults.MimeTypes.Concat(
                new[] { "image/svg+xml" });
    });
}
```

Definir a compactação de nível com [GzipCompressionProviderOptions](#). O provedor de compactação Gzip assume como padrão o nível de compactação mais rápido ([CompressionLevel.Fastest](#)), que não pode produzir a compactação mais eficiente. Se a compactação mais eficiente é desejada, configure o middleware de compactação ideal.

NÍVEL DE COMPACTAÇÃO	DESCRIÇÃO
<a href="#">CompressionLevel.Fastest</a>	A compactação deve ser concluída assim que possível, mesmo se a saída resultante não é compactada da maneira ideal.
<a href="#">CompressionLevel.NoCompression</a>	Nenhuma compactação deve ser executada.
<a href="#">CompressionLevel.Optimal</a>	As respostas devem ser compactadas ideal, mesmo se a compactação leva mais tempo para concluir.

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCompression();

    services.Configure<GzipCompressionProviderOptions>(options =>
    {
        options.Level = CompressionLevel.Fastest;
    });
}

```

## Provedores personalizados

Criar implementações de compactação personalizado com [ICompressionProvider](#). O [EncodingName](#) representa o conteúdo de codificação que este [ICompressionProvider](#) produz. O middleware usa essas informações para escolher o provedor de acordo com a lista especificada no [Accept-Encoding](#) cabeçalho da solicitação.

Usando o aplicativo de exemplo, o cliente envia uma solicitação com o [Accept-Encoding: mycustomcompression](#) cabeçalho. O middleware usa a implementação de compactação personalizado e retorna a resposta com um [Content-Encoding: mycustomcompression](#) cabeçalho. O cliente deve ser capaz de descompactar a codificação personalizada para que uma implementação de compactação personalizado trabalhar.

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCompression(options =>
    {
        options.Providers.Add<BrotliCompressionProvider>();
        options.Providers.Add<GzipCompressionProvider>();
        options.Providers.Add<CustomCompressionProvider>();
        options.MimeTypes =
            ResponseCompressionDefaults.MimeTypes.Concat(
                new[] { "image/svg+xml" });
    });
}

```

```

public class CustomCompressionProvider : ICompressionProvider
{
    public string EncodingName => "mycustomcompression";
    public bool SupportsFlush => true;

    public Stream CreateStream(Stream outputStream)
    {
        // Create a custom compression stream wrapper here
        return outputStream;
    }
}

```

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCompression(options =>
    {
        options.Providers.Add<GzipCompressionProvider>();
        options.Providers.Add<CustomCompressionProvider>();
        options.MimeTypes =
            ResponseCompressionDefaults.MimeTypes.Concat(
                new[] { "image/svg+xml" });
    });
}

```

```
public class CustomCompressionProvider : ICompressionProvider
{
    public string EncodingName => "mycustomcompression";
    public bool SupportsFlush => true;

    public Stream CreateStream(Stream outputStream)
    {
        // Create a custom compression stream wrapper here
        return outputStream;
    }
}
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCompression(options =>
    {
        options.Providers.Add<GzipCompressionProvider>();
        options.Providers.Add<CustomCompressionProvider>();
        options.MimeTypes =
            ResponseCompressionDefaults.MimeTypes.Concat(
                new[] { "image/svg+xml" });
    });
}
```

```
public class CustomCompressionProvider : ICompressionProvider
{
    public string EncodingName => "mycustomcompression";
    public bool SupportsFlush => true;

    public Stream CreateStream(Stream outputStream)
    {
        // Create a custom compression stream wrapper here
        return outputStream;
    }
}
```

Enviar uma solicitação para o aplicativo de exemplo com o `Accept-Encoding: mycustomcompression` cabeçalho e observar os cabeçalhos de resposta. O `Vary` e `Content-Encoding` cabeçalhos estão presentes na resposta. O corpo de resposta (não mostrado) não será compactado pelo exemplo. Não existe uma implementação da compactação no `CustomCompressionProvider` classe do exemplo. No entanto, o exemplo mostra onde você poderia implementar tal um algoritmo de compactação.

The screenshot shows the Fiddler tool interface. In the 'Request Headers' section, under 'Client', 'Accept-Encoding: mycustomcompression' is highlighted with a red box. Under 'Transport', 'Host: localhost:5000' is listed. In the 'Response Headers' section, under 'Cache', 'Date: Thu, 19 Jan 2017 22:06:50 GMT' and 'Vary: Accept-Encoding' are highlighted with red boxes. Under 'Entity', 'Content-Encoding: mycustomcompression' and 'Content-Type: text/plain' are listed. Under 'Miscellaneous', 'Server: Kestrel' is listed. Under 'Transport', 'Transfer-Encoding: chunked' is listed.

## tipos MIME

O middleware Especifica um conjunto de tipos MIME para a compactação padrão:

- `application/javascript`
- `application/json`
- `application/xml`
- `text/css`
- `text/html`
- `text/json`
- `text/plain`
- `text/xml`

Substituir ou acrescentar os tipos MIME com as opções de Middleware de compactação de resposta. Observe que curinga MIME tipos, como `text/*` não são suportados. O aplicativo de exemplo adiciona um tipo MIME `image/svg+xml` e compacta e serve o ASP.NET Core a imagem da faixa (`banner.svg`).

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCompression(options =>
    {
        options.Providers.Add<BrotliCompressionProvider>();
        options.Providers.Add<GzipCompressionProvider>();
        options.Providers.Add<CustomCompressionProvider>();
        options.MimeTypes =
            ResponseCompressionDefaults.MimeTypes.Concat(
                new[] { "image/svg+xml" });
    });
}
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCompression(options =>
    {
        options.Providers.Add<GzipCompressionProvider>();
        options.Providers.Add<CustomCompressionProvider>();
        options.MimeTypes =
            ResponseCompressionDefaults.MimeTypes.Concat(
                new[] { "image/svg+xml" });
    });
}
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCompression(options =>
    {
        options.Providers.Add<GzipCompressionProvider>();
        options.Providers.Add<CustomCompressionProvider>();
        options.MimeTypes =
            ResponseCompressionDefaults.MimeTypes.Concat(
                new[] { "image/svg+xml" });
    });
}
```

## Compactação com protocolo seguro

Respostas compactadas através de conexões seguras podem ser controladas com o `EnableForHttps` opção, que é desabilitada por padrão. Usar a compactação com páginas geradas dinamicamente pode levar a problemas de segurança, como o [CRIME](#) e [violação](#) ataques.

## Adicionando o cabeçalho Vary

Quando a compactação de respostas com base no `Accept-Encoding` cabeçalho, há potencialmente várias versões compactadas de resposta e uma versão não compactada. Para instruir os caches de cliente e o proxy que várias versões existirem e devem ser armazenadas, o `Vary` cabeçalho é adicionado com um `Accept-Encoding` valor. No ASP.NET Core 2.0 ou posterior, o middleware adiciona o `Vary` cabeçalho automaticamente quando a resposta é compactada.

Quando a compactação de respostas com base no `Accept-Encoding` cabeçalho, há potencialmente várias versões compactadas de resposta e uma versão não compactada. Para instruir os caches de cliente e o proxy que várias versões existirem e devem ser armazenadas, o `Vary` cabeçalho é adicionado com um `Accept-Encoding` valor. No ASP.NET Core 1.x, adicionando o `Vary` cabeçalho à resposta é realizado manualmente:

```
// ONLY REQUIRED FOR ASP.NET CORE 1.x APPS
private void ManageVaryHeader(HttpContext context)
{
    // If the Accept-Encoding header is present, add the Vary header
    var accept = context.Request.Headers[HeaderNames.AcceptEncoding];
    if (!StringValues.IsNullOrEmpty(accept))
    {
        context.Response.Headers.Append(HeaderNames.Vary, HeaderNames.AcceptEncoding);
    }
}
```

## Problema de middleware quando atrás de um proxy reverso do Nginx

Quando uma solicitação é transmitida por proxy pelo Nginx, o `Accept-Encoding` cabeçalho é removido. Remoção do `Accept-Encoding` cabeçalho impede que o middleware de compactação de resposta. Para obter mais informações, consulte [NGINX: Compactação e descompactação](#). Esse problema é acompanhado pelo [descobrir a compactação de passagem do Nginx \(aspnet/BasicMiddleware #123\)](#).

## Trabalhando com a compactação dinâmica do IIS

Se você tiver um Active Directory dinâmico compactação de módulo do IIS configurado no nível do servidor que você deseja desabilitar para um aplicativo, desabilite o módulo com uma adição à `Web.config` arquivo. Para obter mais informações, consulte [Desabilitando módulos do IIS](#).

## Solução de problemas

Use uma ferramenta como [Fiddler](#), [Firebug](#), ou [Postman](#), que permitem que você defina o `Accept-Encoding` cabeçalho de solicitação e estudar os cabeçalhos de resposta, o tamanho e o corpo. Por padrão, o Middleware de compactação de resposta compacta as respostas que atendem às seguintes condições:

- O `Accept-Encoding` cabeçalho está presente com um valor de `br`, `gzip`, `*`, ou codificação personalizada que corresponde a um provedor de compactação personalizado estabelecida por você. O valor não deve ser `identity` ou tem um valor de qualidade (qvalue, `q`) configuração de 0 (zero).
- O tipo MIME ( `Content-Type` ) deve ser definido e deve corresponder a um tipo MIME configurado no [ResponseCompressionOptions](#).
- A solicitação não deve incluir o `Content-Range` cabeçalho.
- A solicitação deve usar protocolo inseguro (http), a menos que o protocolo seguro (https) é configurado nas opções de Middleware de compactação de resposta. *Observe o perigo descritos acima ao habilitar a compactação de conteúdo segura.*
- O `Accept-Encoding` cabeçalho está presente com um valor de `gzip`, `*`, ou codificação personalizada que corresponde a um provedor de compactação personalizado estabelecida por você. O valor não deve ser `identity` ou tem um valor de qualidade (qvalue, `q`) configuração de 0 (zero).
- O tipo MIME ( `Content-Type` ) deve ser definido e deve corresponder a um tipo MIME configurado no [ResponseCompressionOptions](#).
- A solicitação não deve incluir o `Content-Range` cabeçalho.
- A solicitação deve usar protocolo inseguro (http), a menos que o protocolo seguro (https) é configurado nas opções de Middleware de compactação de resposta. *Observe o perigo descritos acima ao habilitar a compactação de conteúdo segura.*

## Recursos adicionais

- [Inicialização de aplicativo no ASP.NET Core](#)
- [Middleware do ASP.NET Core](#)
- [Rede de desenvolvedor do Mozilla: Codificação aceita](#)
- [RFC 7231 seção 3.1.2.1: Conteúdo Codings](#)
- [RFC 7230 seção 4.2.3: A codificação gzip](#)
- [Versão de especificação de formato de arquivo GZIP 4.3](#)

# Ferramentas de diagnóstico de desempenho

08/01/2019 • 5 minutes to read • [Edit Online](#)

Por [Mike Rousos](#)

Este artigo lista as ferramentas para diagnosticar problemas de desempenho no ASP.NET Core.

## Ferramentas de diagnóstico do Visual Studio

O [ferramentas de criação de perfil e diagnóstico](#) incorporado ao Visual Studio são um bom lugar para começar a investigar problemas de desempenho. Essas ferramentas são poderosas e conveniente usar o ambiente de desenvolvimento do Visual Studio. As ferramentas permite que a análise de uso de CPU, uso de memória e eventos de desempenho em aplicativos ASP.NET Core. Interno que está sendo torna a criação de perfil fácil em tempo de desenvolvimento.

Mais informações estão disponíveis no [documentação do Visual Studio](#).

## Informações do aplicativo

[Application Insights](#) fornece dados de desempenho detalhados para seu aplicativo. Application Insights coleta automaticamente dados em taxas de resposta, taxas de falha, os tempos de resposta de dependência e muito mais. Application Insights dá suporte ao registro em log eventos personalizados e métricas específicas para seu aplicativo.

O Azure Application Insights fornece várias maneiras de fornecer informações nos aplicativos monitorados:

- [Mapa do aplicativo](#) – ajuda a identificar gargalos de desempenho ou falha pontos em todos os componentes de aplicativos distribuídos.
- [Folha métricas no portal do Application Insights](#) mostra valores de medida e contagens de eventos.
- [Folha de desempenho no portal do Application Insights](#):
  - Mostra detalhes de desempenho para operações diferentes no aplicativo monitorado.
  - Permite que o drill down em uma única operação para verificar todas as partes/dependências que contribuem para um longo tempo.
  - Profiler pode ser invocado a partir daqui para coletar rastreamentos de desempenho sob demanda.
- [De do Azure Application Insights Profiler](#) permite regular e sob demanda de criação de perfil de aplicativos .NET. Mostra portal do Azure capturado rastreamentos de desempenho com as pilhas de chamadas e caminhos de acesso. Os arquivos de rastreamento também podem ser baixados para análise mais profunda com o PerfView.

Application Insights podem ser usados em ambientes de uma variedade:

- Otimizado para funcionar no Azure.
- Funciona em produção, desenvolvimento e preparo.
- Funciona localmente a partir [Visual Studio](#) ou em outros ambientes de hospedagem.

Para obter mais informações, veja [Application Insights para ASP.NET Core](#).

## PerfView

O [PerfView](#) é uma ferramenta de análise de desempenho criada pela equipe do .NET especificamente para

diagnosticar problemas de desempenho do .NET. O PerfView permite que a análise do CPU uso, memória e GC comportamento, eventos de desempenho e a hora do relógio.

Você pode aprender mais sobre como começar com o PerfView [tutoriais em vídeo](#) [PerfView](#) ou ao ler o guia do usuário disponível na ferramenta ou [no GitHub](#).

## Windows Performance Toolkit

[Windows Performance Toolkit](#) (WPT) consiste em dois componentes: Windows Performance Recorder (WPR) e o Windows Performance Analyzer (WPA). As ferramentas produzem perfis detalhados de desempenho de aplicativos e sistemas de operacionais do Windows. WPT tem maneiras mais ricas de visualização de dados, mas suas coletas de dados são menos eficientes do que do PerfView.

## PerfCollect

Enquanto o PerfView é uma ferramenta de análise de desempenho úteis para cenários do .NET, ele só é executado no Windows para que você não pode usá-lo para coletar rastreamentos de aplicativos do ASP.NET Core em execução em ambientes Linux.

[PerfCollect](#) é um script bash que usa as ferramentas de criação de perfil de Linux nativo ([Perf](#) e [LTng](#)) para coletar rastreamentos no Linux que pode ser analisado por PerfView. PerfCollect é útil quando problemas de desempenho aparecem em ambientes Linux em que o PerfView não pode ser usado diretamente. Em vez disso, PerfCollect pode coletar rastreamentos de aplicativos .NET Core que, em seguida, são analisados em um computador Windows usando o PerfView.

Para obter mais informações sobre como instalar e começar a trabalhar com PerfCollect estão disponíveis [no GitHub](#).

## Outras ferramentas de desempenho de terceiros

O exemplo a seguir lista algumas ferramentas de desempenho de terceiros que são úteis na investigação de desempenho de aplicativos .NET Core.

- [MiniProfiler](#)
- o dotTrace e dotMemory da JetBrains
- VTune da Intel

# ASP.NET Core teste de estresse e carregar

11/01/2019 • 4 minutes to read • [Edit Online](#)

Teste de carga e testes de estresse são importantes para garantir que um aplicativo web é eficaz e escalonável. Suas metas são diferentes, mesmo que eles compartilham muitas vezes testes semelhantes.

**Testes de carga:** Testa se o aplicativo pode lidar com uma carga especificada de usuários para um determinado cenário e ainda assim satisfazer a meta de resposta. O aplicativo é executado em condições normais.

**Testes de estresse:** Estabilidade do aplicativo de testes ao executar sob condições extremas e muitas vezes um longo período de tempo:

- Carga de usuário com altos – picos ou aumentando gradualmente.
- Recursos de computação limitados.

Sob carga excessiva, pode o aplicativo se recuperar de falha e normalmente retornar ao comportamento esperado? Sob carga excessiva, o aplicativo está *não* executado sob condições normais.

## Ferramentas do Visual Studio

Visual Studio permite aos usuários criar, desenvolver e depurar testes de carga e desempenho na web. Uma opção está disponível para criar testes gravando ações em um navegador da web.

[Início Rápido: Criar um projeto de teste de carga](#) mostra como criar, configurar e executar um teste de carga projetos usando o Visual Studio 2017.

Consulte [Recursos adicionais](#) para obter mais informações.

Testes de carga podem ser configurados para executar no local ou executados na nuvem usando o DevOps do Azure.

## DevOps do Azure

Execuções de teste de carga podem ser iniciadas usando o [planos de teste do Azure DevOps](#) service.

The screenshot shows the Azure DevOps interface for 'CustomerSamples'. On the left, there's a sidebar with icons for Overview, Boards, Repos, Pipelines, Test Plans, Parameters, Configurations, Runs, Load test (which is selected), and Artifacts. The main area has a header: 'Create and run high-scale load tests, analyze results – all using the browser! Learn more. During preview, this feature...' with a 'New' button and a search bar. Below this is a list of test types: Visual Studio test, HTTP Archive based test\*, URL based test, and Apache JMeter test. Underneath these are several load test runs listed: LoadTest1.loadtest, PrimacyLoadTests, Rate Service Performance Test Update..., and TestAPI.loadtest. At the bottom right of the main area, there are buttons for Run test, Open, Stop, and Run Type.

O serviço suporta os seguintes tipos de formato de teste:

- Teste do Visual Studio – teste da web criado no Visual Studio.
- Teste com base em arquivo HTTP – tráfego HTTP capturado dentro do arquivo morto é reproduzida durante o teste.
- **Teste com base na URL** – permite especificar URLs para carregar testes, tipos de solicitação, cabeçalhos e cadeias de caracteres de consulta. Executar a configuração de parâmetros, como duração, padrão de carga, o número de usuários, etc., pode ser configurado.
- **Apache JMeter** de teste.

## Portal do Azure

[Portal do Azure](#) permite configurar e executar testes de carga de aplicativos Web, diretamente a partir da guia de desempenho do serviço de aplicativo no portal do Azure.

The screenshot shows the Azure Portal interface. On the left, there's a sidebar with icons for Change App Service plan, Development Tools (Clone App, Console, Advanced Tools, App Service Editor (Preview), Performance test, Resource explorer, Testing in production), and other options like Set Organization. The 'Performance test' option is selected and highlighted in blue. The main area has a search bar at the top. Below it, there's a 'Recent runs' section with a table header 'NAME' and a message 'No performance test found. C'. There are also sections for 'Logs' and 'Metrics'.

O teste pode ser um teste manual com uma URL especificada, ou um arquivo de teste do Visual Studio Web, que pode testar várias URLs.

The screenshot shows the 'Configure test using' dialog for a new performance test. On the left, under 'CONFIGURE TEST USING', the 'Test type: ManualTest 1 Url' option is selected. On the right, the 'TEST TYPE' dropdown is set to 'Manual Test'. The 'URL' field contains 'http://studymodulechooser.azurewebsites.net'. Other settings include 'NAME' (PerfTest01), 'GENERATE LOAD FROM' (Central US (Web app Location)), and 'USER LOAD' (250).

No final do teste, os relatórios são gerados para mostrar as características de desempenho do aplicativo.

Estatísticas de exemplo incluem:

- Tempo médio de resposta
- Taxa de transferência máxima: solicitações por segundo
- Percentual de falha

## Ferramentas de terceiros

A lista a seguir contém as ferramentas de desempenho da web de terceiros com vários conjuntos de recursos:

- [Apache JMeter](#) : Pacote de destaque completo de ferramentas de teste de carga. Limite de thread: precisa de um thread por usuário.
- [AB - servidor HTTP Apache ferramenta de benchmark](#)
- [Gatling](#) : Ferramenta da área de trabalho com gravadores de GUI e de teste. Mais eficaz do que o JMeter.
- [Locust.IO](#) : Não é limitado por threads.

## Recursos adicionais

[Série de blogs de teste de carga](#) por Charles Sterling. Com data, mas a maioria dos tópicos ainda é relevante.

# Globalização e localização no ASP.NET Core

13/11/2018 • 32 minutes to read • [Edit Online](#)

Por [Rick Anderson](#), [Damien Bowden](#), [Bart Calixto](#), [Nadeem Afana](#) e [Hisham Bin Ateya](#)

A criação de um site multilíngue com o ASP.NET Core permitirá que seu site alcance um público maior. O ASP.NET Core fornece serviços e middleware para localização em diferentes idiomas e culturas.

A internacionalização envolve [Globalização](#) e [Localização](#). Globalização é o processo de criação de aplicativos que dão suporte a diferentes culturas. A globalização adiciona suporte para entrada, exibição e saída de um conjunto definido de scripts de idiomas relacionados a áreas geográficas específicas.

Localização é o processo de adaptar um aplicativo globalizado, que você já processou para possibilidade de localização, a determinada cultura/localidade. Para obter mais informações, consulte [Termos de globalização e localização](#) próximo ao final deste documento.

A localização de aplicativos envolve o seguinte:

1. Tornar o conteúdo do aplicativo localizável
2. Fornecer recursos localizados para as culturas e os idiomas aos quais você dá suporte
3. Implementar uma estratégia para selecionar o idioma e a cultura para cada solicitação

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Tornar o conteúdo do aplicativo localizável

Introduzidos no ASP.NET Core `IStringLocalizer` e `IStringLocalizer<T>` foram projetados para melhorar a produtividade ao desenvolver aplicativos localizados. `IStringLocalizer` usa o `ResourceManager` e o `ResourceReader` para fornecer recursos específicos a uma cultura em tempo de execução. A interface simples tem um indexador e um `IEnumerable` para retornar cadeias de caracteres localizadas. `IStringLocalizer` não exige o armazenamento das cadeias de caracteres de idioma padrão em um arquivo de recurso. Você pode desenvolver um aplicativo direcionado à localização e não precisa criar arquivos de recurso no início do desenvolvimento. O código abaixo mostra como encapsular a cadeia de caracteres "About Title" para localização.

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Localization;

namespace Localization.Controllers
{
    [Route("api/[controller]")]
    public class AboutController : Controller
    {
        private readonly IStringLocalizer<AboutController> _localizer;

        public AboutController(IStringLocalizer<AboutController> localizer)
        {
            _localizer = localizer;
        }

        [HttpGet]
        public string Get()
        {
            return _localizer["About Title"];
        }
    }
}
```

No código acima, a implementação `IStringLocalizer<T>` é obtida da [Injeção de Dependência](#). Se o valor localizado de "About Title" não é encontrado, a chave do indexador é retornada, ou seja, a cadeia de caracteres "About Title". Deixe as cadeias de caracteres literais de idioma padrão no aplicativo e encapsule-as no localizador, de modo que você possa se concentrar no desenvolvimento do aplicativo. Você pode desenvolver seu aplicativo com o idioma padrão e prepará-lo para a etapa de localização sem primeiro criar um arquivo de recurso padrão. Como alternativa, você pode usar a abordagem tradicional e fornecer uma chave para recuperar a cadeia de caracteres de idioma padrão. Para muitos desenvolvedores, o novo fluxo de trabalho de não ter um arquivo .resx de idioma padrão e simplesmente encapsular os literais de cadeia de caracteres pode reduzir a sobrecarga de localizar um aplicativo. Outros desenvolvedores preferirão o fluxo de trabalho tradicional, pois ele pode facilitar o trabalho com literais de cadeia de caracteres mais longas e a atualização de cadeias de caracteres localizadas.

Use a implementação `IHtmlLocalizer<T>` para recursos que contêm HTML. O `IHtmlLocalizer` codifica em HTML os argumentos que são formatados na cadeia de caracteres de recurso, mas não codifica em HTML a cadeia de caracteres de recurso em si. Na amostra realçada abaixo, apenas o valor do parâmetro `name` é codificado em HTML.

```

using System;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Localization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Localization;

namespace Localization.Controllers
{
    public class BookController : Controller
    {
        private readonly IHtmlLocalizer<BookController> _localizer;

        public BookController(IHtmlLocalizer<BookController> localizer)
        {
            _localizer = localizer;
        }

        public IActionResult Hello(string name)
        {
            ViewData["Message"] = _localizer["<b>Hello</b><i> {0}</i>", name];

            return View();
        }
    }
}

```

**Observação:** em geral, recomendamos localizar somente o texto e não o HTML.

No nível mais baixo, você pode obter `IStringLocalizerFactory` com a [Injeção de Dependência](#):

```

{
    public class TestController : Controller
    {
        private readonly IStringLocalizer _localizer;
        private readonly IStringLocalizer _localizer2;

        public TestController(IStringLocalizerFactory factory)
        {
            var type = typeof(SharedResource);
            var assemblyName = new AssemblyName(type.GetTypeInfo().Assembly.FullName);
            _localizer = factory.Create(type);
            _localizer2 = factory.Create("SharedResource", assemblyName.Name);
        }

        public IActionResult About()
        {
            ViewData["Message"] = _localizer["Your application description page."]
                + " loc 2: " + _localizer2["Your application description page."];
        }
    }
}

```

O código acima demonstra cada um dos dois métodos `Create` de alocador.

Você pode particionar as cadeias de caracteres localizadas por controlador, área ou ter apenas um contêiner. No aplicativo de exemplo, uma classe fictícia chamada `SharedResource` é usada para recursos compartilhados.

```

// Dummy class to group shared resources

namespace Localization
{
    public class SharedResource
    {
    }
}

```

Alguns desenvolvedores usam a classe `Startup` para conter cadeias de caracteres globais ou compartilhadas. Na amostra abaixo, os localizadores `InfoController` e `SharedResource` são usados:

```
public class InfoController : Controller
{
    private readonly IStringLocalizer<InfoController> _localizer;
    private readonly IStringLocalizer<SharedResource> _sharedLocalizer;

    public InfoController(IStringLocalizer<InfoController> localizer,
                          IStringLocalizer<SharedResource> sharedLocalizer)
    {
        _localizer = localizer;
        _sharedLocalizer = sharedLocalizer;
    }

    public string TestLoc()
    {
        string msg = "Shared resx: " + _sharedLocalizer["Hello!"] +
                    " Info resx " + _localizer["Hello!"];
        return msg;
    }
}
```

## Localização de exibição

O serviço `IViewLocalizer` fornece cadeias de caracteres localizadas para uma [exibição](#). A classe `ViewLocalizer` implementa essa interface e encontra o local do recurso no caminho do arquivo de exibição. O seguinte código mostra como usar a implementação padrão de `IViewLocalizer`:

```
@using Microsoft.AspNetCore.Mvc.Localization

@inject IViewLocalizer Localizer

 @{
    ViewData["Title"] = Localizer["About"];
}
<h2>@ViewData["Title"].</h2>
<h3>@ViewData["Message"]</h3>

<p>@Localizer["Use this area to provide additional information."]</p>
```

A implementação padrão de `IViewLocalizer` encontra o arquivo de recurso com base no nome de arquivo da exibição. Não há nenhuma opção para usar um arquivo de recurso compartilhado global. `ViewLocalizer` implementa o localizador usando `IHtmlLocalizer` e, portanto, o Razor não codifica em HTML a cadeia de caracteres localizada. Parametrize cadeias de recurso e o `IViewLocalizer` codificará em HTML os parâmetros, mas não a cadeia de caracteres de recurso. Considere a seguinte marcação do Razor:

```
@Localizer["<i>Hello</i> <b>{0}</b>", UserManager.GetUserName(User)]
```

Um arquivo de recurso em francês pode conter o seguinte:

CHAVE	VALOR
<i>Hello</i> <b>{0}</b>	<i>Bonjour</i> <b>{0}</b>

A exibição renderizada contém a marcação HTML do arquivo de recurso.

**Observação:** em geral, recomendamos localizar somente o texto e não o HTML.

Para usar um arquivo de recurso compartilhado em uma exibição, injete `IHtmlLocalizer<T>`:

```
@using Microsoft.AspNetCore.Mvc.Localization  
@using Localization.Services  
  
@inject IViewLocalizer Localizer  
@inject IHtmlLocalizer<SharedResource> SharedLocalizer  
  
{@  
    ViewData["Title"] = Localizer["About"];  
}  
<h2>@ViewData["Title"].</h2>  
  
<h1>@SharedLocalizer["Hello!"]</h1>
```

## Localização de DataAnnotations

As mensagens de erro de DataAnnotations são localizadas com `IStringLocalizer<T>`. Usando a opção `ResourcesPath = "Resources"`, as mensagens de erro em `RegisterViewModel` podem ser armazenadas em um dos seguintes caminhos:

- `Resources/ViewModels.Account.RegisterViewModel.fr.resx`
- `Resources/ViewModels/Account/RegisterViewModel.fr.resx`

```
public class RegisterViewModel  
{  
    [Required(ErrorMessage = "The Email field is required.")]  
    [EmailAddress(ErrorMessage = "The Email field is not a valid email address.")]  
    [Display(Name = "Email")]  
    public string Email { get; set; }  
  
    [Required(ErrorMessage = "The Password field is required.")]  
    [StringLength(8, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]  
    [DataType(DataType.Password)]  
    [Display(Name = "Password")]  
    public string Password { get; set; }  
  
    [DataType(DataType.Password)]  
    [Display(Name = "Confirm password")]  
    [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]  
    public string ConfirmPassword { get; set; }  
}
```

No ASP.NET Core MVC 1.1.0 e superior, atributos que não sejam de validação são localizados. O ASP.NET Core MVC 1.0 **não** pesquisa cadeias de caracteres localizadas para atributos que não sejam de validação.

### Usando uma cadeia de caracteres de recurso para várias classes

O seguinte código mostra como usar uma cadeia de caracteres de recurso para atributos de validação com várias classes:

```
public void ConfigureServices(IServiceCollection services)  
{  
    services.AddMvc()  
        .AddDataAnnotationsLocalization(options => {  
            options.DataAnnotationLocalizerProvider = (type, factory) =>  
                factory.Create(typeof(SharedResource));  
        });  
}
```

No código anterior, `SharedResource` é a classe correspondente ao resx em que as mensagens de validação são armazenadas. Com essa abordagem, DataAnnotations usará apenas `SharedResource`, em vez de o recurso para cada classe.

## Fornecer recursos localizados para as culturas e os idiomas aos quais você dá suporte

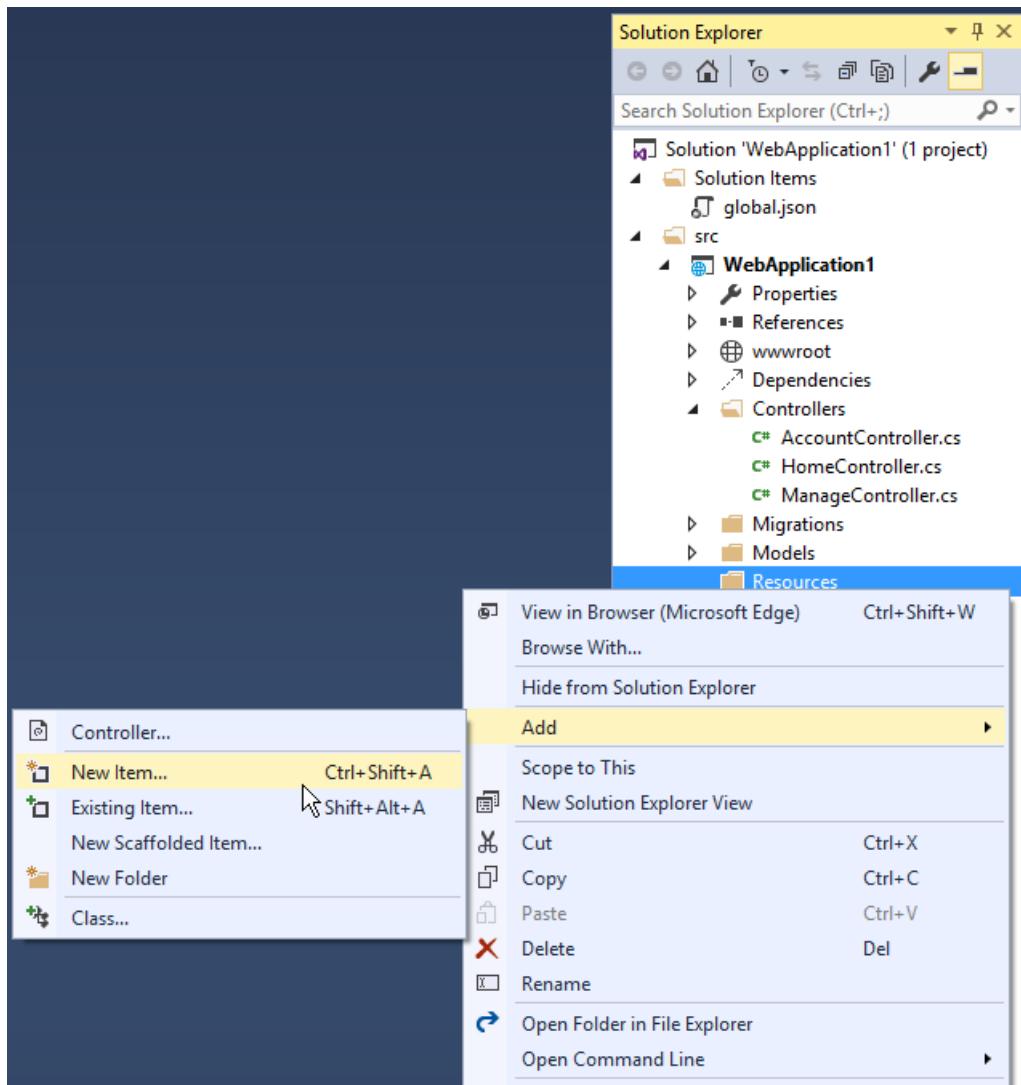
### **SupportedCultures** e **SupportedUICultures**

O ASP.NET Core permite que você especifique dois valores de cultura, `SupportedCultures` e `SupportedUICultures`. O objeto `CultureInfo` para `SupportedCultures` determina os resultados das funções dependentes de cultura, como data, hora, número e formatação de moeda. `SupportedCultures` também determina a ordem de classificação de texto, convenções de uso de maiúsculas e comparações de cadeia de caracteres. Consulte `CultureInfo.CurrentCulture` para obter mais informações sobre como o servidor obtém a Cultura. O `SupportedUICultures` determina quais cadeias de caracteres traduzidas (de arquivos .resx) são pesquisadas pelo `ResourceManager`. O `ResourceManager` apenas pesquisa cadeias de caracteres específicas a uma cultura determinadas por `CurrentUICulture`. Cada thread no .NET tem objetos `CurrentCulture` e `CurrentUICulture`. O ASP.NET Core inspeciona esses valores durante a renderização de funções dependentes de cultura. Por exemplo, se a cultura do thread atual estiver definida como "en-US" (inglês, Estados Unidos), `DateTime.Now.ToString("F")` exibirá "Thursday, February 18, 2016", mas se `CurrentCulture` estiver definida como "es-ES" (espanhol, Espanha), o resultado será "jueves, 18 de febrero de 2016".

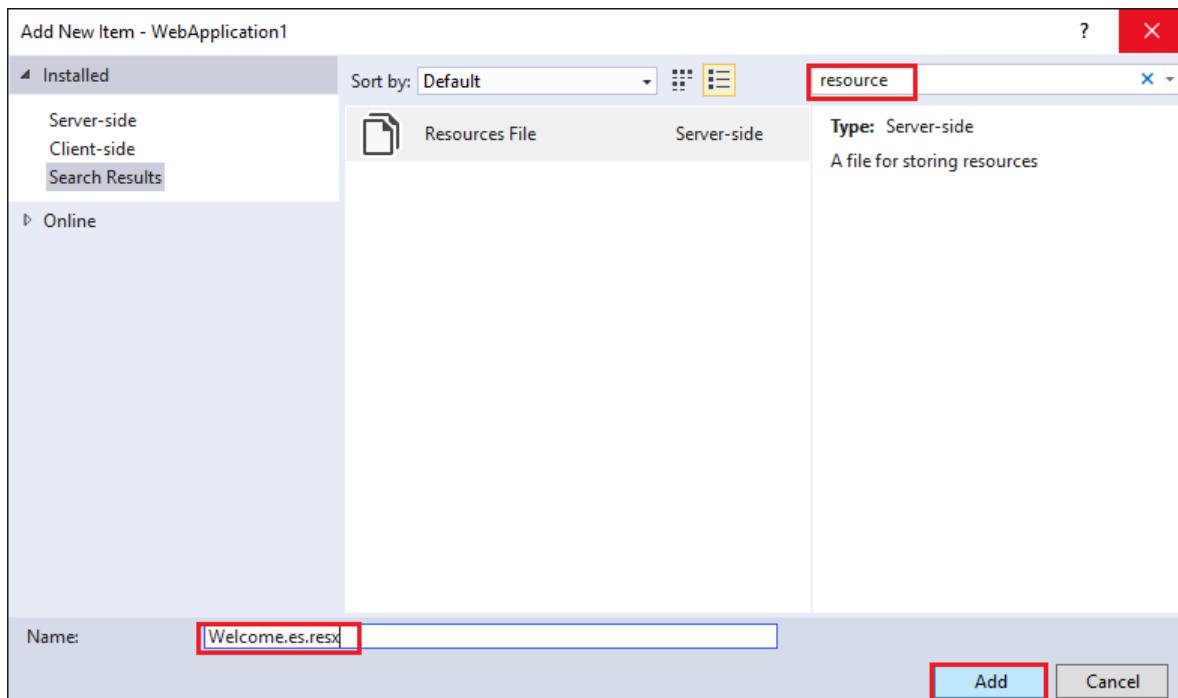
## Arquivos de recurso

Um arquivo de recurso é um mecanismo útil para separar cadeias de caracteres localizáveis do código. Cadeias de caracteres traduzidas para o idioma não padrão são arquivos de recurso .resx isolados. Por exemplo, talvez você queira criar um arquivo de recurso em espanhol chamado *Welcome.es.resx* contendo cadeias de caracteres traduzidas. "es" são o código de idioma para o espanhol. Para criar esse arquivo de recurso no Visual Studio:

1. No **Gerenciador de Soluções**, clique com o botão direito do mouse na pasta que conterá o arquivo de recurso > **Adicionar** > **Novo Item**.



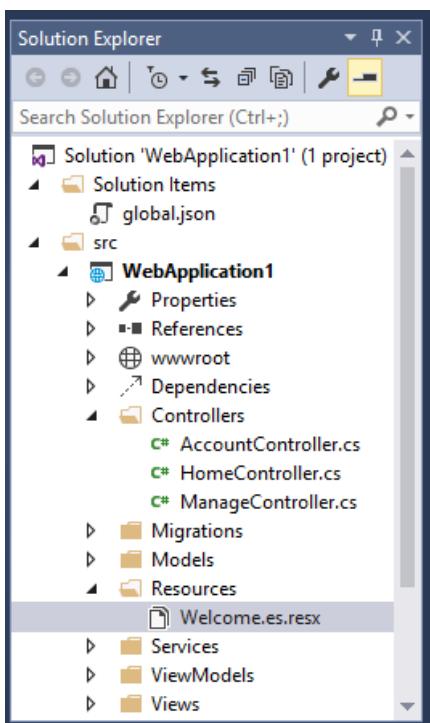
2. Na caixa **Pesquisar modelos instalados**, insira "recurso" e nomeie o arquivo.



3. Insira o valor da chave (cadeia de caracteres nativa) na coluna **Nome** e a cadeia de caracteres traduzida na coluna **Valor**.

Name	Value	Comment
Hello	Hola	
*		

O Visual Studio mostra o arquivo `Welcome.es.resx`.



## Nomenclatura do arquivo de recurso

Os recursos são nomeados com o nome completo do tipo de sua classe menos o nome do assembly. Por exemplo, um recurso em francês em um projeto cujo assembly principal é `LocalizationWebsite.Web.dll` para a classe `LocalizationWebsite.Web.Startup` será nomeado `Startup.fr.resx`. Um recurso para a classe `LocalizationWebsite.Web.Controllers.HomeController` será nomeado `Controllers.HomeControllerfr.resx`. Se o namespace da classe de destino não for o mesmo que o nome do assembly, você precisará do nome completo do tipo. Por exemplo, no projeto de exemplo, um recurso para o tipo `ExtraNamespace.Tools` será nomeado `ExtraNamespace.Tools.fr.resx`.

No projeto de exemplo, o método `ConfigureServices` define o `ResourcesPath` como "Resources", de modo que o caminho relativo do projeto para o arquivo de recurso em francês do controlador principal seja `Resources/Controllers.HomeController.fr.resx`. Como alternativa, você pode usar pastas para organizar arquivos de recurso. Para o controlador principal, o caminho será `Resources/Controllers/HomeController.fr.resx`. Se você não usar a opção `ResourcesPath`, o arquivo `.resx` entrará no diretório base do projeto. O arquivo de recurso para `HomeController` será nomeado `Controllers.HomeController.fr.resx`. A opção de usar a convenção de nomenclatura de ponto ou caminho depende de como você deseja organizar os arquivos de recurso.

NOME DO RECURSO	NOMENCLATURA DE PONTO OU CAMINHO
Resources/Controllers.HomeController.fr.resx	Ponto
Resources/Controllers/HomeController.fr.resx	Caminho

Os arquivos de recurso que usam `@inject IViewLocalizer` em exibições do Razor seguem um padrão semelhante. O arquivo de recurso de uma exibição pode ser nomeado usando a nomenclatura de ponto ou de caminho. Os arquivos de recurso da exibição do Razor simulam o caminho de seu arquivo de exibição associado. Supondo que definimos o `ResourcesPath` como "Resources", o arquivo de recurso em francês associado à exibição `Views/Home/About.cshtml` pode ser um dos seguintes:

- Resources/Views/Home/About.fr.resx
- Resources/Views.Home.About.fr.resx

Se você não usar a opção `ResourcesPath`, o arquivo `.resx` de uma exibição estará localizado na mesma pasta da exibição.

### RootNamespaceAttribute

O atributo `RootNamespace` fornece o namespace raiz de um assembly quando o namespace raiz de um assembly é diferente do nome do assembly.

Se o namespace raiz de um assembly é diferente do nome do assembly:

- A localização não funciona por padrão.
- A localização falha devido à maneira como os recursos são pesquisados dentro do assembly. `RootNamespace` é um valor de tempo de build que não está disponível para o processo em execução.

Se o `RootNamespace` é diferente de `AssemblyName`, inclua o seguinte no `AssemblyInfo.cs` (com valores de parâmetro substituídos pelos valores reais):

```
using System.Reflection;
using Microsoft.Extensions.Localization;

[assembly: ResourceLocation("Resource Folder Name")]
[assembly: RootNamespace("App Root Namespace")]
```

O código anterior permite a resolução bem-sucedida de arquivos `resx`.

## Comportamento de fallback da cultura

Ao procurar por um recurso, a localização participa do "fallback de cultura". Começando com a cultura solicitada, se ela não for encontrada, ela será revertida para a cultura pai dessa cultura. Além disso, a propriedade `CultureInfo.Parent` representa a cultura pai. Isso normalmente (mas nem sempre) significa a remoção do significante do ISO. Por exemplo, o dialeto do espanhol falado no México é "es-MX". Ele tem o pai "es", ou seja, espanhol, não específico de nenhum país.

Imagine que seu site receba uma solicitação de um recurso "Boas-vindas" usando a cultura "fr-CA". O sistema de localização procura os seguintes recursos, em ordem, e seleciona a primeira correspondência:

- `Welcome.fr-CA.resx`
- `Welcome.fr.resx`
- `Welcome.resx` (se o `NeutralResourcesLanguage` for "fr-CA")

Por exemplo, se você remover o designador de cultura "fr" e tiver a cultura definida como francês, o arquivo de recurso padrão será lido e as cadeias de caracteres serão localizadas. O Gerenciador de Recursos designa um padrão ou um recurso de fallback para quando nenhuma opção atende à cultura solicitada. Se você deseja apenas retornar a chave quando um recurso está ausente para a cultura solicitada, não deve ter um arquivo de recurso padrão.

### Gerar arquivos de recurso com o Visual Studio

Se você criar um arquivo de recurso no Visual Studio sem uma cultura no nome do arquivo (por exemplo, *Welcome.resx*), o Visual Studio criará uma classe do C# com uma propriedade para cada cadeia de caracteres. Geralmente isso não é o desejado com o ASP.NET Core. Normalmente você não tem um arquivo de recurso *.resx* padrão (um arquivo *.resx* sem o nome de cultura). Sugerimos que você crie o arquivo *.resx* com um nome de cultura (por exemplo *Welcome.fr.resx*). Quando você criar um arquivo *.resx* com um nome de cultura, o Visual Studio não gerará o arquivo de classe. A nossa previsão é que muitos desenvolvedores não criariam um arquivo de recurso de idioma padrão.

### Adicionar outras culturas

Cada combinação de idioma e cultura (que não seja o idioma padrão) exige um arquivo de recurso exclusivo. Crie arquivos de recurso para diferentes culturas e localidades criando novos arquivos de recurso, nos quais os códigos ISO e de idioma fazem parte do nome do arquivo (por exemplo, **en-us**, **fr-ca** e **en-gb**). Esses códigos ISO são colocados entre o nome do arquivo e a extensão de arquivo *.resx*, como em *Welcome.es-MX.resx* (espanhol/México).

## Implementar uma estratégia para selecionar o idioma e a cultura para cada solicitação

### Configurar a localização

A localização é configurada no método `Startup.ConfigureServices`:

```
services.AddLocalization(options => options.ResourcesPath = "Resources");

services.AddMvc()
    .AddViewLocalization(LanguageViewLocationExpanderFormat.Suffix)
    .AddDataAnnotationsLocalization();
```

- `AddLocalization` Adiciona os serviços de localização ao contêiner de serviços. O código acima também define o caminho de recursos como "Resources".
- `AddViewLocalization` Adiciona suporte para os arquivos de exibição localizados. Nesta amostra, a localização de exibição se baseia no sufixo do arquivo de exibição. Por exemplo, "fr" no arquivo *Index.fr.cshtml*.
- `AddDataAnnotationsLocalization` Adiciona suporte para as mensagens de validação `DataAnnotations` localizadas por meio de abstrações `IStringLocalizer`.

### Middleware de localização

A cultura atual em uma solicitação é definida no [Middleware](#) de localização. O middleware de localização é habilitado no método `Startup.Configure`. O middleware de localização precisa ser configurado antes de qualquer middleware que possa verificar a cultura de solicitação (por exemplo, `app.UseMvcWithDefaultRoute()`).

```

var supportedCultures = new[]
{
    new CultureInfo("en-US"),
    new CultureInfo("fr"),
};

app.UseRequestLocalization(new RequestLocalizationOptions
{
    DefaultRequestCulture = new RequestCulture("en-US"),
    // Formatting numbers, dates, etc.
    SupportedCultures = supportedCultures,
    // UI strings that we have localized.
    SupportedUICultures = supportedCultures
});

app.UseStaticFiles();
// To configure external authentication,
// see: http://go.microsoft.com/fwlink/?LinkID=532715
app.UseAuthentication();
app.UseMvcWithDefaultRoute();

```

`UseRequestLocalization` inicializa um objeto `RequestLocalizationOptions`. Em cada solicitação, a lista de `RequestCultureProvider` em `RequestLocalizationOptions` é enumerada e o primeiro provedor que pode determinar com êxito a cultura de solicitação é usado. Os provedores padrão são obtidos da classe `RequestLocalizationOptions`:

1. `QueryStringRequestCultureProvider`
2. `CookieRequestCultureProvider`
3. `AcceptLanguageHeaderRequestCultureProvider`

A lista padrão é apresentada do mais específico ao menos específico. Mais adiante neste artigo, veremos como você pode alterar a ordem e até mesmo adicionar um provedor de cultura personalizado. Se nenhum dos provedores pode determinar a cultura de solicitação, o `DefaultRequestCulture` é usado.

### QueryStringRequestCultureProvider

Alguns aplicativos usarão uma cadeia de caracteres de consulta para definir a [cultura e a cultura da interface do usuário](#). Para aplicativos que usam a abordagem do cabeçalho Accept-Language ou do cookie, a adição de uma cadeia de caracteres de consulta à URL é útil para depurar e testar o código. Por padrão, o

`QueryStringRequestCultureProvider` é registrado como o primeiro provedor de localização na lista `RequestCultureProvider`. Passe os parâmetros `culture` e `ui-culture` da cadeia de caracteres de consulta. O seguinte exemplo define a cultura específica (idioma e região) como espanhol/México:

```
http://localhost:5000/?culture=es-MX&ui-culture=es-MX
```

Se você passar somente uma das duas (`culture` ou `ui-culture`), o provedor da cadeia de caracteres de consulta definirá os dois valores usando aquela que foi passada. Por exemplo, a definição apenas da cultura definirá a `Culture` e a `UICulture`:

```
http://localhost:5000/?culture=es-MX
```

### CookieRequestCultureProvider

Em geral, aplicativos de produção fornecerão um mecanismo para definir a cultura com o cookie de cultura do ASP.NET Core. Use o método `MakeCookieValue` para criar um cookie.

O `CookieRequestCultureProvider` `DefaultCookieName` retorna o nome padrão do cookie usado para acompanhar as informações de cultura preferencial do usuário. O nome padrão do cookie é `.AspNetCore.Culture`.

O formato do cookie é `c=%LANGCODE%|uic=%LANGCODE%`, em que `c` é `Culture` e `uic` é `UICulture`, por exemplo:

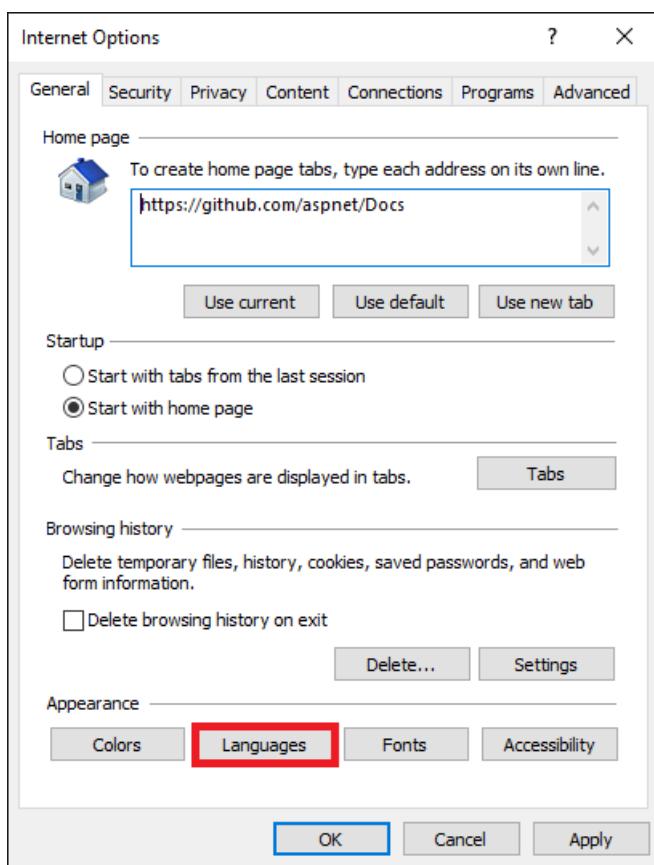
Se você especificar apenas as informações de cultura e a cultura da interface do usuário, a cultura especificada será usada para as informações de cultura e para a cultura da interface do usuário.

### O cabeçalho HTTP Accept-Language

O [cabeçalho Accept-Language](#) é configurável na maioria dos navegadores e originalmente foi criado para especificar o idioma do usuário. Essa configuração indica que o navegador foi definido para enviar ou herdou do sistema operacional subjacente. O cabeçalho HTTP Accept-Language de uma solicitação do navegador não é uma maneira infalível de detectar o idioma preferencial do usuário (consulte [Definindo preferências de idioma em um navegador](#)). Um aplicativo de produção deve incluir uma maneira para que um usuário personalize sua opção de cultura.

### Definir o cabeçalho HTTP Accept-Language no IE

1. No ícone de engrenagem, toque em **Opções da Internet**.
2. Toque em **Idiomas**.



3. Toque em **Definir Preferências de Idioma**.
4. Toque em **Adicionar um idioma**.
5. Adicione o idioma.
6. Toque no idioma e, em seguida, em **Mover Para Cima**.

### Usar um provedor personalizado

Suponha que você deseja permitir que os clientes armazenem seus idiomas e culturas nos bancos de dados. Você pode escrever um provedor para pesquisar esses valores para o usuário. O seguinte código mostra como adicionar um provedor personalizado:

```

private const string enUSCulture = "en-US";

services.Configure<RequestLocalizationOptions>(options =>
{
    var supportedCultures = new[]
    {
        new CultureInfo(enUSCulture),
        new CultureInfo("fr")
    };

    options.DefaultRequestCulture = new RequestCulture(culture: enUSCulture, uiCulture: enUSCulture);
    options.SupportedCultures = supportedCultures;
    options.SupportedUICultures = supportedCultures;

    options.RequestCultureProviders.Insert(0, new CustomRequestCultureProvider(async context =>
    {
        // My custom request culture logic
        return new ProviderCultureResult("en");
    }));
});

```

Use `RequestLocalizationOptions` para adicionar ou remover provedores de localização.

## Definir a cultura de forma programática

Este projeto de exemplo **Localization.StarterWeb** no [GitHub](#) contém a interface do usuário para definir a `Culture`. O arquivo `Views/Shared/_SelectLanguagePartial.cshtml` permite que você selecione a cultura na lista de culturas compatíveis:

```

@using Microsoft.AspNetCore.Builder
@using Microsoft.AspNetCore.Http.Features
@using Microsoft.AspNetCore.Localization
@using Microsoft.AspNetCore.Mvc.Localization
@using Microsoft.Extensions.Options

@inject IViewLocalizer Localizer
@inject IOptions<RequestLocalizationOptions> LocOptions

 @{
    var requestCulture = Context.Features.Get< IRequestCultureFeature>();
    var cultureItems = LocOptions.Value.SupportedUICultures
        .Select(c => new SelectListItem { Value = c.Name, Text = c.DisplayName })
        .ToList();
    var returnUrl = string.IsNullOrEmpty(Context.Request.Path) ? "/" : $"~{Context.Request.Path.Value}";
}

<div title="@Localizer["Request culture provider:"] @requestCulture?.Provider?.GetType().Name">
    <form id="selectLanguage" asp-controller="Home"
        asp-action="SetLanguage" asp-route-returnUrl="@returnUrl"
        method="post" class="form-horizontal" role="form">
        <label asp-for="@requestCulture.RequestCulture.UICulture.Name">@Localizer["Language:"]</label>
        <select name="culture"
            onchange="this.form.submit();"
            asp-for="@requestCulture.RequestCulture.UICulture.Name" asp-items="cultureItems">
            </select>
    </form>
</div>

```

O arquivo `Views/Shared/_SelectLanguagePartial.cshtml` é adicionado à seção `footer` do arquivo de layout para que ele fique disponível para todos os exibições:

```

<div class="container body-content" style="margin-top:60px">
    @RenderBody()
    <hr>
    <footer>
        <div class="row">
            <div class="col-md-6">
                <p>&copy; @System.DateTime.Now.Year - Localization</p>
            </div>
            <div class="col-md-6 text-right">
                @await Html.PartialAsync("_SelectLanguagePartial")
            </div>
        </div>
    </footer>
</div>

```

O método `SetLanguage` define o cookie de cultura.

```

[HttpPost]
public IActionResult SetLanguage(string culture, string returnUrl)
{
    Response.Cookies.Append(
        CookieRequestCultureProvider.DefaultCookieName,
        CookieRequestCultureProvider.MakeCookieValue(new RequestCulture(culture)),
        new CookieOptions { Expires = DateTimeOffset.UtcNow.AddYears(1) }
    );

    return LocalRedirect(returnUrl);
}

```

Não é possível conectar o `_SelectLanguagePartial.cshtml` ao código de exemplo para este projeto. O projeto **Localization.StarterWeb** no [GitHub](#) contém o código para o fluxo do `RequestLocalizationOptions` para uma parcial do Razor por meio do contêiner de [Injeção de Dependência](#).

## Termos de globalização e localização

O processo de localização do aplicativo também exige uma compreensão básica dos conjuntos de caracteres relevantes geralmente usados no desenvolvimento moderno de software e uma compreensão dos problemas associados a eles. Embora todos os computadores armazenem texto como números (códigos), diferentes sistemas armazenam o mesmo texto usando números diferentes. O processo de localização se refere à tradução da interface do usuário do aplicativo para uma cultura/localidade específica.

[Possibilidade de localização](#) é um processo intermediário usado para verificar se um aplicativo globalizado está pronto para localização.

O formato [RFC 4646](#) para o nome de cultura é `<languagecode2>-<country/regioncode2>`, em que `<languagecode2>` é o código de idioma e `<country/regioncode2>` é o código de subcultura. Por exemplo, `es-CL` para Espanhol (Chile), `en-US` para inglês (Estados Unidos) e `en-AU` para inglês (Austrália). O [RFC 4646](#) é uma combinação de um código de cultura de duas letras minúsculas ISO 639 associado a um idioma e um código de subcultura de duas letras maiúsculas ISO 3166 associado a um país ou uma região. Consulte [Nome da cultura de idioma](#).

Muitas vezes, internacionalização é abreviada como "I18N". A abreviação usa a primeira e última letras e o número de letras entre elas e, portanto, 18 significa o número de letras entre o primeiro "I" e o último "N". O mesmo se aplica a Globalização (G11N) e Localização (L10N).

Termos:

- Globalização (G11N): o processo de fazer com que um aplicativo dê suporte a diferentes idiomas e regiões.
- Localização (L10N): o processo de personalizar um aplicativo para determinado idioma e região.

- Internacionalização (I18N): descreve a globalização e a localização.
- Cultura: um idioma e, opcionalmente, uma região.
- Cultura neutra: uma cultura que tem um idioma especificado, mas não uma região. (por exemplo, "en", "es")
- Cultura específica: uma cultura que tem um idioma e uma região especificados. (por exemplo, "en-US", "en-GB", "es-CL")
- Cultura pai: a cultura neutra que contém uma cultura específica. (por exemplo, "en" é a cultura pai de "en-US" e "en-GB")
- Localidade: uma localidade é o mesmo que uma cultura.

#### NOTE

Talvez você não consiga inserir vírgulas decimais em campos decimais. Para dar suporte à [validação do jQuery](#) para localidades de idiomas diferentes do inglês que usam uma vírgula (",") para um ponto decimal e formatos de data diferentes do inglês dos EUA, você deve tomar medidas para globalizar o aplicativo. Veja [Problema 4076 do GitHub](#) para obter instruções sobre como adicionar casas decimais.

## Recursos adicionais

- O projeto [Localization.StarterWeb](#) usado no artigo.
- [Globalizando e localizando aplicativos do .NET](#)
- [Recursos em arquivos .resx](#)
- [Kit de Ferramentas de Aplicativo Multilíngue da Microsoft](#)

# Configurar a localização de objeto portátil no ASP.NET Core

30/10/2018 • 12 minutes to read • [Edit Online](#)

Por Sébastien Ros e Scott Addie

Este artigo explica as etapas para usar arquivos PO (Objeto Portátil) em um aplicativo ASP.NET Core com a estrutura [Orchard Core](#).

**Observação:** o Orchard Core não é um produto da Microsoft. Consequentemente, a Microsoft não fornece suporte para esse recurso.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## O que é um arquivo PO?

Os arquivos PO são distribuídos como arquivos de texto que contém cadeias de caracteres traduzidas em determinado idioma. Algumas vantagens do uso de arquivos PO em vez de arquivos .resx incluem:

- Os arquivos PO dão suporte à pluralização, ao contrário dos arquivos .resx.
- Os arquivos PO não são compilados como os arquivos .resx. Dessa forma, não são necessárias ferramentas especializadas nem etapas de build.
- Os arquivos PO funcionam bem com ferramentas de colaboração de edição online.

### Exemplo

Este é um arquivo PO de exemplo que contém a tradução de duas cadeias de caracteres em francês, incluindo uma com sua forma plural:

*fr.po*

```
#: Services/EmailService.cs:29
msgid "Enter a comma separated list of email addresses."
msgstr "Entrez une liste d'emails séparés par une virgule."

#: Views/Email.cshtml:112
msgid "The email address is \"{0}\"."
msgid_plural "The email addresses are \"{0}\"."
msgstr[0] "L'adresse email est \"{0}\"."
msgstr[1] "Les adresses email sont \"{0}\""
```

Este exemplo usa a seguinte sintaxe:

- `#:` : um comentário que indica o contexto da cadeia de caracteres a ser traduzida. A mesma cadeia de caracteres pode ser traduzida de modo diferente, dependendo do local em que ela está sendo usada.
- `msgid` : a cadeia de caracteres não traduzida.
- `msgstr` : a cadeia de caracteres traduzida.

No caso de suporte à pluralização, entradas adicionais podem ser definidas.

- `msgid_plural` : a cadeia de caracteres no plural não traduzida.
- `msgstr[0]` : a cadeia de caracteres traduzida para o caso 0.
- `msgstr[N]` : a cadeia de caracteres traduzida para o caso N.

A especificação de arquivo PO pode ser encontrada [aqui](#).

## Configurando o suporte a arquivos PO no ASP.NET Core

Este exemplo se baseia em um aplicativo ASP.NET Core MVC gerado com base em um modelo de projeto do Visual Studio 2017.

### Referenciando o pacote

Adicione uma referência ao pacote NuGet `OrchardCore.Localization.Core`. Ele está disponível em [MyGet](#) na fonte do pacote a seguir: <https://www.myget.org/F/orchardcore-preview/api/v3/index.json>

O arquivo `.csproj` agora contém uma linha semelhante à seguinte (o número de versão pode variar):

```
<PackageReference Include="OrchardCore.Localization.Core" Version="1.0.0-beta1-3187" />
```

### Registrando o serviço

Adicione os serviços necessários ao método `ConfigureServices` de `Startup.cs`:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc()
        .AddViewLocalization(LanguageViewLocationExpanderFormat.Suffix);

    services.AddPortableObjectLocalization();

    services.Configure<RequestLocalizationOptions>(options =>
    {
        var supportedCultures = new List<CultureInfo>
        {
            new CultureInfo("en-US"),
            new CultureInfo("en"),
            new CultureInfo("fr-FR"),
            new CultureInfo("fr")
        };

        options.DefaultRequestCulture = new RequestCulture("en-US");
        options.SupportedCultures = supportedCultures;
        options.SupportedUICultures = supportedCultures;
    });
}
```

Adicione o middleware necessário ao método `Configure` de `Startup.cs`:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseBrowserLink();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseStaticFiles();

    app.UseRequestLocalization();

    app.UseMvcWithDefaultRoute();
}
```

Adicione o código a seguir à exibição do Razor de sua escolha. *About.cshtml* é usado neste exemplo.

```
@using Microsoft.AspNetCore.Mvc.Localization
@inject IViewLocalizer Localizer

<p>@Localizer["Hello world!"]</p>
```

Uma instância `IViewLocalizer` é injetada e usada para traduzir o texto "Olá, Mundo!".

## Criando um arquivo PO

Crie um arquivo chamado `.po` na pasta raiz do aplicativo. Neste exemplo, o nome do arquivo é `fr.po` porque o idioma francês é usado:

```
msgid "Hello world!"
msgstr "Bonjour le monde!"
```

Esse arquivo armazena a cadeia de caracteres a ser traduzida e a cadeia de caracteres traduzida do francês. As traduções são revertidas para a cultura pai, se necessário. Neste exemplo, o arquivo `fr.po` é usado se a cultura solicitada é `fr-FR` ou `fr-CA`.

## Testando o aplicativo

Execute o aplicativo e navegue para a URL `/Home/About`. O texto **Olá, Mundo!** é exibido.

Navegue para a URL `/Home/About?culture=fr-FR`. O texto **Bonjour le monde!** é exibido.

## Pluralização

Os arquivos PO dão suporte a formas de pluralização, que são úteis quando a mesma cadeia de caracteres precisa ser traduzida de modo diferente de acordo com uma cardinalidade. Essa tarefa torna-se complicada pelo fato de que cada idioma define regras personalizadas para selecionar qual cadeia de caracteres será usada de acordo com a cardinalidade.

O pacote de Localização do Orchard fornece uma API para invocar essas diferentes formas plurais automaticamente.

## Criando arquivos PO de pluralização

Adicione o seguinte conteúdo ao arquivo `fr.po` mencionado anteriormente:

```
msgid "There is one item."
msgid_plural "There are {0} items."
msgstr[0] "Il y a un élément."
msgstr[1] "Il y a {0} éléments."
```

Consulte [O que é um arquivo PO?](#) para obter uma explicação do que representa cada entrada neste exemplo.

## Adicionando um idioma usando diferentes formas de pluralização

Cadeias de caracteres em inglês e francês foram usadas no exemplo anterior. O inglês e o francês têm apenas duas formas de pluralização e compartilham as mesmas regras de forma, o que significa que uma cardinalidade de um é mapeada para a primeira forma plural. Qualquer outra cardinalidade é mapeada para a segunda forma plural.

Nem todos os idiomas compartilham as mesmas regras. Isso é ilustrado com o idioma tcheco, que tem três formas plurais.

Crie o arquivo `cs.po` da seguinte maneira e observe como a pluralização precisa de três traduções diferentes:

```
msgid "Hello world!"
msgstr "Ahoj světe!!"

msgid "There is one item."
msgid_plural "There are {0} items."
msgstr[0] "Existuje jedna položka."
msgstr[1] "Existují {0} položky."
msgstr[2] "Existuje {0} položek."
```

Para aceitar localizações para o tcheco, adicione `"cs"` à lista de culturas com suporte no método

`ConfigureServices`:

```
var supportedCultures = new List<CultureInfo>
{
    new CultureInfo("en-US"),
    new CultureInfo("en"),
    new CultureInfo("fr-FR"),
    new CultureInfo("fr"),
    new CultureInfo("cs")
};
```

Edito o arquivo `Views/Home/About.cshtml` para renderizar cadeias de caracteres localizadas no plural para várias cardinalidades:

```
<p>@Localizer.Plural(1, "There is one item.", "There are {0} items.")</p>
<p>@Localizer.Plural(2, "There is one item.", "There are {0} items.")</p>
<p>@Localizer.Plural(5, "There is one item.", "There are {0} items.")</p>
```

**Observação:** em um cenário do mundo real, uma variável é usada para representar a contagem. Aqui, repetimos o mesmo código com três valores diferentes para expor um caso muito específico.

Ao mudar as culturas, o seguinte é observado:

Para `/Home/About`:

```
There is one item.
There are 2 items.
There are 5 items.
```

Para `/Home/About?culture=fr`:

```
Il y a un élément.  
Il y a 2 éléments.  
Il y a 5 éléments.
```

Para `/Home/About?culture=cs`:

```
Existuje jedna položka.  
Existují 2 položky.  
Existuje 5 položek.
```

Observe que, para a cultura do tcheco, as três traduções são diferentes. As culturas do francês e do inglês compartilham a mesma construção para as duas últimas cadeias de caracteres traduzidas.

## Tarefas avançadas

### Contextualizando cadeias de caracteres

Os aplicativos costumam conter as cadeias de caracteres a serem traduzidas em vários locais. A mesma cadeia de caracteres pode ter uma tradução diferente em determinados locais em um aplicativo (exibições do Razor ou arquivos de classe). Um arquivo PO dá suporte à noção de um contexto de arquivo, que pode ser usado para categorizar a cadeia de caracteres que está sendo representada. Usando um contexto de arquivo, uma cadeia de caracteres pode ser traduzida de forma diferente, dependendo do contexto de arquivo (ou de sua ausência).

Os serviços de localização de PO usam o nome da classe completa ou da exibição que é usado ao traduzir uma cadeia de caracteres. Isso é feito definindo o valor na entrada `msgctxt`.

Considere uma adição mínima ao exemplo anterior de `fr.po`. Uma exibição do Razor localizada em `Views/Home/About.cshtml` pode ser definida com o contexto de arquivo definindo o valor da entrada `msgctxt` reservada:

```
msgctxt "Views.Home.About"  
msgid "Hello world!"  
msgstr "Bonjour le monde!"
```

Com o `msgctxt` definido assim, a tradução de texto ocorre durante a navegação para `/Home/About?culture=fr-FR`. A tradução não ocorre durante a navegação para `/Home/Contact?culture=fr-FR`.

Quando não é encontrada a correspondência de nenhuma entrada específica com um contexto de arquivo fornecido, o mecanismo de fallback do Orchard Core procura um arquivo PO apropriado sem contexto. Supondo que não haja nenhum contexto de arquivo específico definido para `Views/Home/Contact.cshtml`, a navegação para `/Home/Contact?culture=fr-FR` carrega um arquivo PO, como:

```
msgid "Hello world!"  
msgstr "Bonjour le monde!"
```

### Alterando o local dos arquivos PO

A localização padrão dos arquivos PO pode ser alterada em `ConfigureServices`:

```
services.AddPortableObjectLocalization(options => options.ResourcesPath = "Localization");
```

Neste exemplo, os arquivos PO são carregados da pasta *Localization*.

## Implementando uma lógica personalizada para encontrar arquivos de localização

Quando uma lógica mais complexa é necessária para localizar arquivos PO, a interface

`OrchardCore.Localization.PortableObject.ILocalizationFileLocationProvider` pode ser implementada e registrada como um serviço. Isso é útil quando arquivos PO podem ser armazenados em locais variáveis ou quando os arquivos precisam ser encontrados em uma hierarquia de pastas.

## Usando um idioma pluralizado padrão diferente

O pacote inclui um método de extensão `Plural` específico a duas formas plurais. Para idiomas que exigem mais formas plurais, crie um método de extensão. Com um método de extensão, você não precisará fornecer nenhum arquivo de localização para o idioma padrão – as cadeias de caracteres originais já estão disponíveis diretamente no código.

Use a sobrecarga `Plural(int count, string[] pluralForms, params object[] arguments)` mais genérica que aceita uma matriz de cadeia de caracteres de traduções.

# Middleware de Reconfiguração de URL no ASP.NET Core

10/01/2019 • 31 minutes to read • [Edit Online](#)

Por [Luke Latham](#) e [Mikael Mengistu](#)

Para obter a versão 1.1 deste tópico, baixe [Middleware de Reconfiguração de URL no ASP.NET Core \(versão 1.1, PDF\)](#).

Este documento apresenta a reconfiguração de URL com instruções sobre como usar o Middleware de Reconfiguração de URL em aplicativos ASP.NET Core.

A reconfiguração de URL é o ato de modificar URLs de solicitação com base em uma ou mais regras predefinidas. A reconfiguração de URL cria uma abstração entre locais de recursos e seus endereços, de forma que os locais e os endereços não estejam totalmente vinculados. A reconfiguração de URL é útil em vários cenários para:

- Mover ou substituir recursos de servidor temporária ou permanentemente e manter localizadores estáveis para esses recursos.
- Dividir o processamento de solicitação entre diferentes aplicativos ou áreas de um aplicativo.
- Remover, adicionar ou reorganizar segmentos de URL em solicitações de entrada.
- Otimizar URLs públicas para SEO (Otimização do Mecanismo de Pesquisa).
- Permitir o uso de URLs públicas amigáveis para ajudar os visitantes a prever o conteúdo retornado ao solicitar um recurso.
- Redirecionar solicitações não seguras para pontos de extremidade seguros.
- Impedir o hotlink, em que um site externo usa um ativo estático hospedado em outro site vinculando o ativo ao seu próprio conteúdo.

## NOTE

A reconfiguração de URL pode reduzir o desempenho de um aplicativo. Sempre que possível, limite o número e a complexidade das regras.

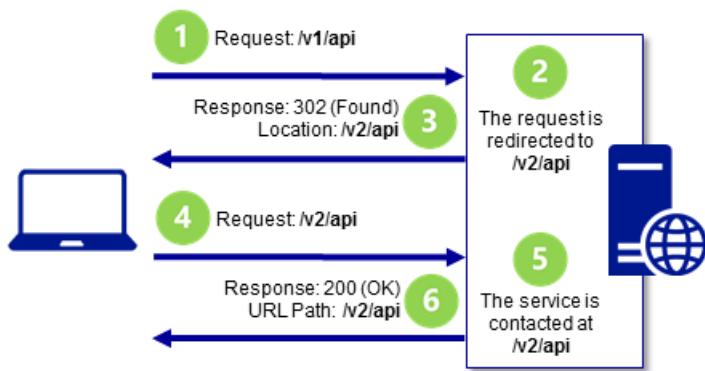
[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Redirecionamento e reconfiguração de URL

A diferença entre os termos *redirecionamento de URL* e *reconfiguração de URL* é sutil, mas tem implicações importantes no fornecimento de recursos aos clientes. O Middleware de Reconfiguração de URL do ASP.NET Core pode atender às necessidades de ambos.

Um *redirecionamento de URL* envolve uma operação do lado do cliente, em que o cliente é instruído a acessar um recurso em um endereço diferente do que ele solicitou originalmente. Isso exige uma viagem de ida e volta para o servidor. A URL de redirecionamento retornada para o cliente é exibida na barra de endereços do navegador quando o cliente faz uma nova solicitação para o recurso.

Se `/resource` é redirecionado para `/different-resource`, o servidor responde que o cliente deve obter o recurso em `/different-resource` com um código de status indicando que o redirecionamento é temporário ou permanente.



Ao redirecionar solicitações para uma URL diferente, indique se o redirecionamento é permanente ou temporário especificando o código de status com a resposta:

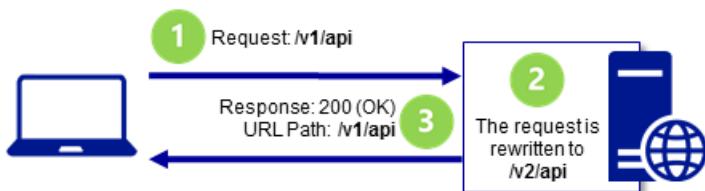
- O código de status *301 – Movido Permanentemente* é usado quando o recurso tem uma nova URL permanente e você deseja instruir o cliente que todas as solicitações futuras para o recurso devem usar a nova URL. *O cliente pode armazenar a resposta em cache e reutilizá-la quando um código de status 301 é recebido.*
- O código de status *302 – Encontrado* é usado quando o redirecionamento é temporário ou está geralmente sujeito a alterações. O código de status 302 indica para o cliente não armazenar a URL e usá-la no futuro.

Para obter mais informações sobre códigos de status, confira [RFC 2616: definições de código de status](#).

Uma *reconfiguração de URL* é uma operação do servidor que fornece um recurso de um endereço de recurso diferente do que o solicitado pelo cliente. A reconfiguração de uma URL não exige uma viagem de ida e volta para o servidor. A URL reconfigurada não é retornada para o cliente e não é exibida na barra de endereços do navegador.

Se `/resource` é reconfigurado como `/different-resource`, o servidor efetua fetch *internamente* e retorna o recurso em `/different-resource`.

Embora o cliente possa ter a capacidade de recuperar o recurso na URL reconfigurada, ele não é informado de que o recurso existe na URL reconfigurada ao fazer a solicitação e receber a resposta.



## Aplicativo de exemplo de reconfiguração de URL

Explore as funcionalidades do Middleware de Reconfiguração de URL com o [aplicativo de exemplo](#). O aplicativo aplica as regras de redirecionamento e reconfiguração e mostra a URL redirecionada ou reconfigurada para vários cenários.

## Quando usar o Middleware de Reconfiguração de URL

Use o Middleware de Reconfiguração de URL quando não for possível usar as seguintes abordagens:

- [Módulo de Reconfiguração de URL com o IIS no Windows Server](#)
- [Módulo mod\\_rewrite do Apache no Apache Server](#)
- [Reconfiguração de URL no Nginx](#)

Além disso, use o middleware quando o aplicativo estiver hospedado no [servidor HTTP.sys](#) (anteriormente chamado WebListener).

As principais razões para usar as tecnologias de reconfiguração de URL baseada em servidor no IIS, no Apache e no Nginx são:

- O middleware não dá suporte às funcionalidades completas desses módulos.

Algumas funcionalidades dos módulos de servidor não funcionam com projetos ASP.NET Core, como as restrições `IsFile` e `IsDirectory` do módulo de Reconfiguração do IIS. Nesses cenários, use o middleware.

- O desempenho do middleware provavelmente não corresponde ao desempenho dos módulos.

Os parâmetros de comparação são a única maneira de saber com certeza qual abordagem degrada mais o desempenho ou se a degradação de desempenho é insignificante.

## Pacote

Para incluir o middleware em seu projeto, adicione uma referência de pacote ao [metapacote Microsoft.AspNetCore.App](#) no arquivo de projeto, que contém o pacote [Microsoft.AspNetCore.Rewrite](#).

Quando não estiver usando o metapacote `Microsoft.AspNetCore.App`, adicione uma referência de projeto ao pacote `Microsoft.AspNetCore.Rewrite`.

## Extensão e opções

Estabeleça regras de reconfiguração e redirecionamento de URL criando uma instância da classe [RewriteOptions](#) com métodos de extensão para cada uma das regras de reconfiguração. Encadeie várias regras na ordem em que deseja que sejam processadas. As `RewriteOptions` são passadas para o Middleware de Reconfiguração de URL, conforme ele é adicionado ao pipeline de solicitação com [UseRewriter](#):

```
public void Configure(IApplicationBuilder app)
{
    using (StreamReader apacheModRewriteStreamReader =
        File.OpenText("ApacheModRewrite.txt"))
    using (StreamReader iisUrlRewriteStreamReader =
        File.OpenText("IISUrlRewrite.xml"))
    {
        var options = new RewriteOptions()
            .AddRedirect("redirect-rule/(.*)", "redirected/$1")
            .AddRewrite(@"^rewrite-rule/(\d+)/(\d+)", "rewritten?var1=$1&var2=$2",
                skipRemainingRules: true)
            .AddApacheModRewrite(apacheModRewriteStreamReader)
            .AddIISUrlRewrite(iisUrlRewriteStreamReader)
            .Add(MethodRules.RedirectXmlFileRequests)
            .Add(MethodRules.RewriteTextFileRequests)
            .Add(new RedirectImageRequests(".png", "/png-images"))
            .Add(new RedirectImageRequests(".jpg", "/jpg-images"));

        app.UseRewriter(options);
    }

    app.UseStaticFiles();

    app.Run(context => context.Response.WriteAsync(
        $"Rewritten or Redirected Url: " +
        $"{context.Request.Path + context.Request.QueryString}"));
}
```

## Redirecionar não www para www

Três opções permitem que o aplicativo redirecione solicitações não `www` para `www`:

- `AddRedirectToWwwPermanent` – Redirecionar permanentemente a solicitação para o subdomínio `www` se a solicitação não for `www`. Redireciona com um código de status `Status308PermanentRedirect`.
- `AddRedirectToWww` – Redirecionar a solicitação para o subdomínio `www` se a solicitação de entrada não for `www`. Redireciona com um código de status `Status307TemporaryRedirect`. Uma sobrecarga permite que você forneça o código de status para a resposta. Use um campo da classe `StatusCodes` para uma atribuição de código de status.

## Redirecionamento de URL

Use `AddRedirect` para redirecionar as solicitações. O primeiro parâmetro contém o regex para correspondência no caminho da URL de entrada. O segundo parâmetro é a cadeia de caracteres de substituição. O terceiro parâmetro, se presente, especifica o código de status. Se você não especificar o código de status, ele usará como padrão `302 – Encontrado`, que indica que o recurso foi substituído ou movido temporariamente.

```
public void Configure(IApplicationBuilder app)
{
    using (StreamReader apacheModRewriteStreamReader =
        File.OpenText("ApacheModRewrite.txt"))
    using (StreamReader iisUrlRewriteStreamReader =
        File.OpenText("IISUrlRewrite.xml"))
    {
        var options = new RewriteOptions()
            .AddRedirect("redirect-rule/(.*)", "redirected/$1")
            .AddRewrite(@"^rewrite-rule/(\d+)/(\d+)", "rewritten?var1=$1&var2=$2",
                skipRemainingRules: true)
            .AddApacheModRewrite(apacheModRewriteStreamReader)
            .AddIISUrlRewrite(iisUrlRewriteStreamReader)
            .Add(MethodRules.RedirectXmlFileRequests)
            .Add(MethodRules.RewriteTextFileRequests)
            .Add(new RedirectImageRequests(".png", "/png-images"))
            .Add(new RedirectImageRequests(".jpg", "/jpg-images"));

        app.UseRewriter(options);
    }

    app.UseStaticFiles();

    app.Run(context => context.Response.WriteAsync(
        $"Rewritten or Redirected Url: " +
        $"{context.Request.Path + context.Request.QueryString}"));
}
```

Em um navegador com as ferramentas para desenvolvedores habilitadas, faça uma solicitação para o aplicativo de exemplo com o caminho `/redirect-rule/1234/5678`. O regex corresponde ao caminho de solicitação em `redirect-rule/(.*)` e o caminho é substituído por `/redirected/1234/5678`. A URL de redirecionamento é enviada novamente ao cliente com um código de status `302 – Encontrado`. O navegador faz uma nova solicitação na URL de redirecionamento, que é exibida na barra de endereços do navegador. Como não há nenhuma correspondência de regras no aplicativo de exemplo na URL de redirecionamento:

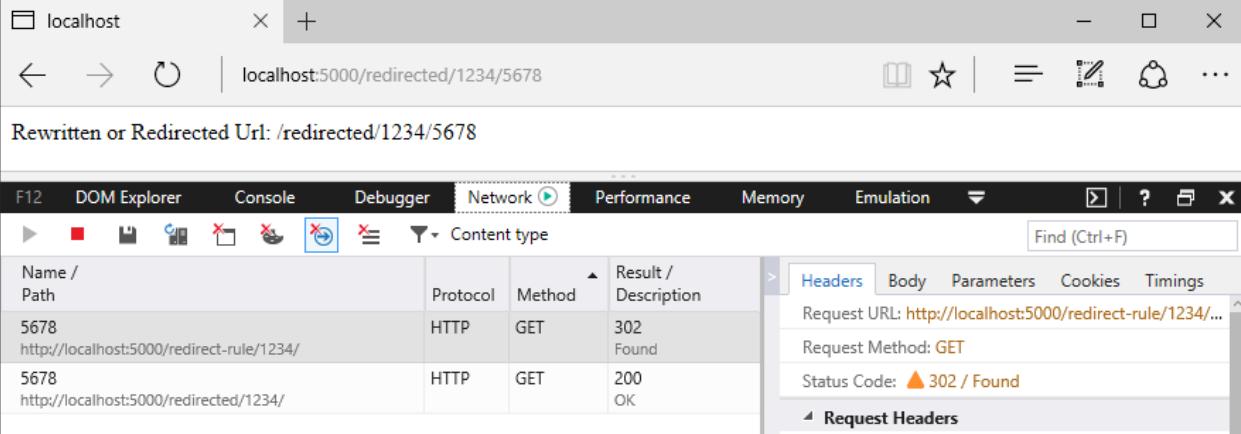
- A segunda solicitação recebe uma resposta `200 – OK` do aplicativo.
- O corpo da resposta mostra a URL de redirecionamento.

Uma viagem de ida e volta é feita para o servidor quando uma URL é *redirecionada*.

## WARNING

Tenha cuidado ao estabelecer regras de redirecionamento. As regras de redirecionamento são avaliadas em cada solicitação para o aplicativo, incluindo após um redirecionamento. É fácil criar acidentalmente um *loop de redirecionamentos infinitos*.

Solicitação original: /redirect-rule/1234/5678



The screenshot shows the Microsoft Edge DevTools Network tab. The address bar displays 'localhost:5000/redirected/1234/5678'. The main content area says 'Rewritten or Redirected Url: /redirected/1234/5678'. The Network tab shows two entries:

Name / Path	Protocol	Method	Result / Description
5678 http://localhost:5000/redirect-rule/1234/	HTTP	GET	302 Found
5678 http://localhost:5000/redirected/1234/	HTTP	GET	200 OK

Details pane on the right shows Request URL: http://localhost:5000/redirect-rule/1234..., Request Method: GET, Status Code: 302 / Found, and Request Headers.

A parte da expressão contida nos parênteses é chamada um *grupo de captura*. O ponto (.) da expressão significa *corresponder a qualquer caractere*. O asterisco (\*) indica *corresponder ao caractere zero precedente ou mais vezes*. Portanto, os dois últimos segmentos de caminho da URL, 1234/5678, são capturados pelo grupo de captura (.\*) . Qualquer valor que você fornecer na URL de solicitação após redirect-rule/ é capturado por esse único grupo de captura.

Na cadeia de caracteres de substituição, os grupos capturados são injetados na cadeia de caracteres com o cifrão (\$) seguido do número de sequência da captura. O primeiro valor de grupo de captura é obtido com \$1 , o segundo com \$2 e eles continuam em sequência para os grupos de captura no regex. Há apenas um grupo capturado no regex da regra de redirecionamento no aplicativo de exemplo, para que haja apenas um grupo injetado na cadeia de caracteres de substituição, que é \$1 . Quando a regra é aplicada, a URL se torna /redirected/1234/5678 .

## Redirecionamento de URL para um ponto de extremidade seguro

Use [AddRedirectToHttps](#) para redirecionar solicitações HTTP para o mesmo host e caminho usando o protocolo HTTPS. Se o código de status não for fornecido, o middleware usará como padrão 302 – *Encontrado*. Se a porta não for fornecida:

- O middleware usará como padrão null .
- O esquema será alterado para https (protocolo HTTPS), e o cliente acessará o recurso na porta 443.

O exemplo a seguir mostra como definir o código de status como 301 – *Movido Permanentemente* e alterar a porta para 5001.

```
public void Configure(IApplicationBuilder app)
{
    var options = new RewriteOptions()
        .AddRedirectToHttps(301, 5001);

    app.UseRewriter(options);
}
```

Use [AddRedirectToHttpsPermanent](#) para redirecionar solicitações não seguras para o mesmo host e caminho com o protocolo HTTPS seguro na porta 443. O middleware define o código de status como 301 – *Movido*

Permanentemente.

```
public void Configure(IApplicationBuilder app)
{
    var options = new RewriteOptions()
        .AddRedirectToHttpsPermanent();

    app.UseRewriter(options);
}
```

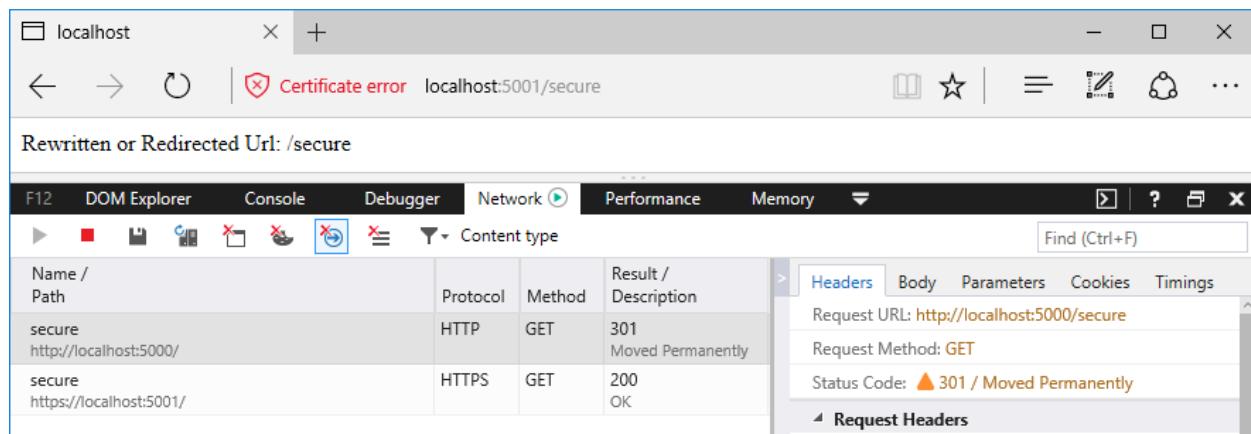
#### NOTE

Ao redirecionar para um ponto de extremidade seguro sem a necessidade de regras de redirecionamento adicionais, recomendamos o uso do Middleware de Redirecionamento HTTPS. Para obter mais informações, veja o tópico [Importar HTTPS](#).

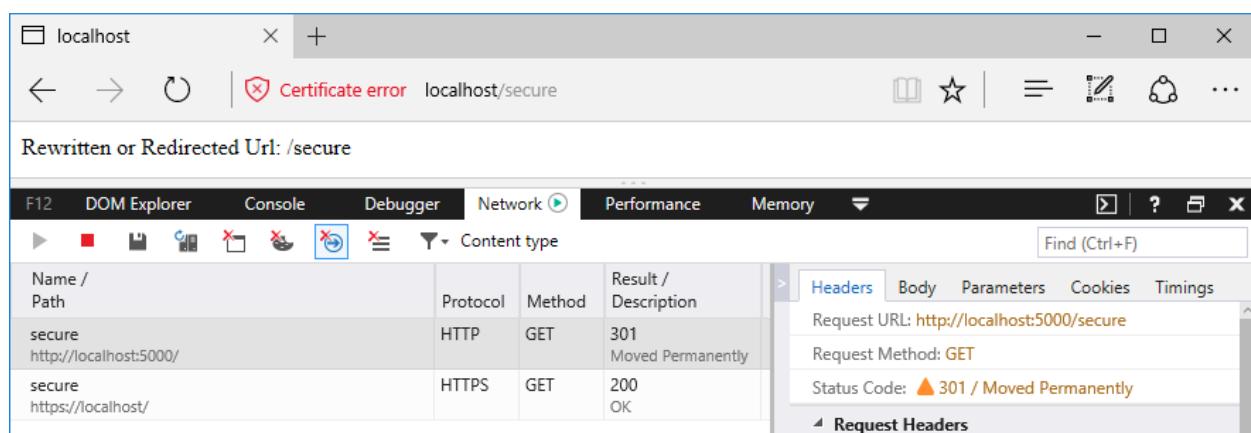
O aplicativo de exemplo pode demonstrar como usar `AddRedirectToHttps` ou `AddRedirectToHttpsPermanent`.

Adicione o método de extensão às `RewriteOptions`. Faça uma solicitação não segura para o aplicativo em qualquer URL. Ignore o aviso de segurança do navegador de que o certificado autoassinado não é confiável ou crie uma exceção para confiar no certificado.

Solicitação original usando `AddRedirectToHttps(301, 5001)` : `http://localhost:5000/secure`



Solicitação original usando `AddRedirectToHttpsPermanent` : `http://localhost:5000/secure`



## Reconfiguração de URL

Use `AddRewrite` para criar uma regra para a reconfiguração de URLs. O primeiro parâmetro contém o regex para correspondência no caminho da URL de entrada. O segundo parâmetro é a cadeia de caracteres de substituição. O terceiro parâmetro, `skipRemainingRules: {true|false}`, indica para o middleware se ele deve ou não ignorar regras de reconfiguração adicionais se a regra atual é aplicada.

```

public void Configure(IApplicationBuilder app)
{
    using (StreamReader apacheModRewriteStreamReader =
        File.OpenText("ApacheModRewrite.txt"))
    using (StreamReader iisUrlRewriteStreamReader =
        File.OpenText("IISUrlRewrite.xml"))
    {
        var options = new RewriteOptions()
            .AddRedirect("redirect-rule/(.*)", "redirected/$1")
            .AddRewrite(@"^rewrite-rule/(\d+)/(\d+)", "rewritten?var1=$1&var2=$2",
                skipRemainingRules: true)
            .AddApacheModRewrite(apacheModRewriteStreamReader)
            .AddIISUrlRewrite(iisUrlRewriteStreamReader)
            .Add(MethodRules.RedirectXmlFileRequests)
            .Add(MethodRules.RewriteTextFileRequests)
            .Add(new RedirectImageRequests(".png", "/png-images"))
            .Add(new RedirectImageRequests(".jpg", "/jpg-images"));

        app.UseRewriter(options);
    }

    app.UseStaticFiles();

    app.Run(context => context.Response.WriteAsync(
        $"Rewritten or Redirected Url: " +
        $"{context.Request.Path + context.Request.QueryString}"));
}

```

Solicitação original: /rewrite-rule/1234/5678

Name / Path	Protocol	Method	Result / Description
5678 http://localhost:5000/rewrite-rule/1234/	HTTP	GET	200 OK

Request URL: http://localhost:5000/rewrite-rule/1234/...  
 Request Method: GET  
 Status Code: 200 / OK  
 ▲ Request Headers

O acento circunflexo (^) no início da expressão significa que a correspondência começa no início do caminho da URL.

No exemplo anterior com a regra de redirecionamento, redirect-rule/(.\*), não há nenhum acento circunflexo (^) no início do regex. Portanto, qualquer caractere pode preceder redirect-rule/ no caminho para uma correspondência com êxito.

CAMINHO	CORRESPONDER A
/redirect-rule/1234/5678	Sim
/my-cool-redirect-rule/1234/5678	Sim
/anotherredirect-rule/1234/5678	Sim

A regra de reconfiguração, ^rewrite-rule/(\d+)/(\d+), corresponde apenas a caminhos se eles são iniciados

com `rewrite-rule/`. Na tabela a seguir, observe a diferença na correspondência.

CAMINHO	CORRESPONDER A
<code>/rewrite-rule/1234/5678</code>	Sim
<code>/my-cool-rewrite-rule/1234/5678</code>	Não
<code>/anotherrewrite-rule/1234/5678</code>	Não

Após a parte `^rewrite-rule/` da expressão, há dois grupos de captura, `(\d+)(\d+)`. O `\d` significa *corresponder a um dígito (número)*. O sinal de adição (`+`) significa *corresponder a um ou mais caracteres anteriores*. Portanto, a URL precisa conter um número seguido de uma barra "/" seguida de outro número. Esses grupos de captura são injetados na URL reconfigurada como `$1` e `$2`. A cadeia de caracteres de substituição da regra de reconfiguração coloca os grupos capturados na cadeia de consulta. O caminho solicitado de `/rewrite-rule/1234/5678` foi reconfigurado para obter o recurso em `/rewritten?var1=1234&var2=5678`. Se uma cadeia de consulta estiver presente na solicitação original, ela será preservada quando a URL for reconfigurada.

Não há nenhuma viagem de ida e volta para o servidor para obtenção do recurso. Se o recurso existir, ele será buscado e retornado para o cliente com um código de status *200 – OK*. Como o cliente não é redirecionado, a URL na barra de endereços do navegador não é alterada. Os clientes não conseguem detectar que uma operação de reconfiguração de URL ocorreu no servidor.

#### NOTE

Use `skipRemainingRules: true` sempre que possível, porque as regras de correspondência são computacionalmente caras e aumentam o tempo de resposta do aplicativo. Para a resposta mais rápida do aplicativo:

- Ordene as regras de reconfiguração da regra com correspondência mais frequente para a regra com correspondência menos frequente.
- Ignore o processamento das regras restantes quando ocorrer uma correspondência e nenhum processamento de regra adicional for necessário.

## mod\_rewrite do Apache

Aplique as regras do mod\_rewrite do Apache com [AddApacheModRewrite](#). Verifique se o arquivo de regras foi implantado com o aplicativo. Para obter mais informações e exemplos de regras de mod\_rewrite, consulte [mod\\_rewrite do Apache](#).

Um [StreamReader](#) é usado para ler as regras do arquivo de regras *ApacheModRewrite.txt*:

```

public void Configure(IApplicationBuilder app)
{
    using (StreamReader apacheModRewriteStreamReader =
        File.OpenText("ApacheModRewrite.txt"))
    using (StreamReader iisUrlRewriteStreamReader =
        File.OpenText("IISUrlRewrite.xml"))
    {
        var options = new RewriteOptions()
            .AddRedirect("redirect-rule/(.*)", "redirected/$1")
            .AddRewrite(@"^rewrite-rule/(\d+)/(\d+)", "rewritten?var1=$1&var2=$2",
                skipRemainingRules: true)
            .AddApacheModRewrite(apacheModRewriteStreamReader)
            .AddIISUrlRewrite(iisUrlRewriteStreamReader)
            .Add(MethodRules.RedirectXmlFileRequests)
            .Add(MethodRules.RewriteTextFileRequests)
            .Add(new RedirectImageRequests(".png", "/png-images"))
            .Add(new RedirectImageRequests(".jpg", "/jpg-images"));

        app.UseRewriter(options);
    }

    app.UseStaticFiles();

    app.Run(context => context.Response.WriteAsync(
        $"Rewritten or Redirected Url: " +
        $"{context.Request.Path + context.Request.QueryString}"));
}

```

O aplicativo de exemplo redireciona solicitações de `/apache-mod-rules-redirect/(.*)` para `/redirected?id=$1`. O código de status da resposta é *302 – Encontrado*.

```

# Rewrite path with additional sub directory
RewriteRule ^/apache-mod-rules-redirect/(.*) /redirected?id=$1 [L,R=302]

```

Solicitação original: `/apache-mod-rules-redirect/1234`

The screenshot shows the Microsoft Edge DevTools Network tab. The request URL is `localhost:5000/redirected?id=1234`. The response status is `Rewritten or Redirected Url: /redirected?id=1234`. The table below shows two entries:

Name / Path	Protocol	Method	Result / Description
<code>1234</code> <code>http://localhost:5000/apache-mod-rules-redirect/</code>	HTTP	GET	302 Found
<code>redirected?id=1234</code> <code>http://localhost:5000/</code>	HTTP	GET	200 OK

Details pane on the right shows:

- Request URL: `http://localhost:5000/apache-mod-rules...`
- Request Method: `GET`
- Status Code: `302 / Found`

O middleware dá suporte às seguintes variáveis de servidor do mod\_rewrite do Apache:

- `CONN_REMOTE_ADDR`
- `HTTP_ACCEPT`
- `HTTP_CONNECTION`
- `HTTP_COOKIE`
- `HTTP_FORWARDED`
- `HTTP_HOST`
- `HTTP_REFERER`

- HTTP\_USER\_AGENT
- HTTPS
- IPV6
- QUERY\_STRING
- REMOTE\_ADDR
- REMOTE\_PORT
- REQUEST\_FILENAME
- REQUEST\_METHOD
- REQUEST\_SCHEME
- REQUEST\_URI
- SCRIPT\_FILENAME
- SERVER\_ADDR
- SERVER\_PORT
- SERVER\_PROTOCOL
- TIME
- TIME\_DAY
- TIME\_HOUR
- TIME\_MIN
- TIME\_MON
- TIME\_SEC
- TIME\_WDAY
- TIME\_YEAR

### **Regras do Módulo de Reconfiguração de URL do IIS**

Para usar o mesmo conjunto de regras que se aplica ao Módulo de Reconfiguração de URL do IIS, use [AddIISUrlRewrite](#). Verifique se o arquivo de regras foi implantado com o aplicativo. Não instrua o middleware a usar o arquivo *web.config* do aplicativo quando ele estiver em execução no IIS do Windows Server. Com o IIS, essas regras devem ser armazenadas fora do arquivo *web.config* do aplicativo para evitar conflitos com o módulo de Reconfiguração do IIS. Para obter mais informações e exemplos de regras do Módulo de Reconfiguração de URL do IIS, consulte [Usando o Módulo de Reconfiguração de URL 2.0](#) e [Referência de configuração do Módulo de Reconfiguração de URL](#).

Um [StreamReader](#) é usado para ler as regras do arquivo de regras *IISUrlRewrite.xml*:

```

public void Configure(IApplicationBuilder app)
{
    using (StreamReader apacheModRewriteStreamReader =
        File.OpenText("ApacheModRewrite.txt"))
    using (StreamReader iisUrlRewriteStreamReader =
        File.OpenText("IISUrlRewrite.xml"))
    {
        var options = new RewriteOptions()
            .AddRedirect("redirect-rule/(.*)", "redirected/$1")
            .AddRewrite(@"^rewrite-rule/(\d+)/(\d+)", "rewritten?var1=$1&var2=$2",
                skipRemainingRules: true)
            .AddApacheModRewrite(apacheModRewriteStreamReader)
            .AddIISUrlRewrite(iisUrlRewriteStreamReader)
            .Add(MethodRules.RedirectXmlFileRequests)
            .Add(MethodRules.RewriteTextFileRequests)
            .Add(new RedirectImageRequests(".png", "/png-images"))
            .Add(new RedirectImageRequests(".jpg", "/jpg-images"));

        app.UseRewriter(options);
    }

    app.UseStaticFiles();

    app.Run(context => context.Response.WriteAsync(
        $"Rewritten or Redirected Url: " +
        $"{context.Request.Path + context.Request.QueryString}"));
}

```

O aplicativo de exemplo reconfigura as solicitações de `/iis-rules-rewrite/(.*)` para `/rewritten?id=$1`. A resposta é enviada ao cliente com um código de status `200 – OK`.

```

<rewrite>
  <rules>
    <rule name="Rewrite segment to id querystring" stopProcessing="true">
      <match url="^iis-rules-rewrite/(.*)$" />
      <action type="Rewrite" url="rewritten?id={R:1}" appendQueryString="false"/>
    </rule>
  </rules>
</rewrite>

```

Solicitação original: `/iis-rules-rewrite/1234`

Name / Path	Protocol	Method	Result / Description
1234 http://localhost:5000/iis-rules-rewrite/	HTTP	GET	200 OK

Caso você tenha um Módulo de Reconfiguração do IIS ativo com regras no nível do servidor configuradas que poderiam afetar o aplicativo de maneiras indesejadas, desabilite o Módulo de Reconfiguração do IIS em um aplicativo. Para obter mais informações, consulte [Desabilitando módulos do IIS](#).

#### Recursos sem suporte

O middleware liberado com o ASP.NET Core 2.x não dá suporte aos seguintes recursos do Módulo de

## Reconfiguração de URL do IIS:

- Regras de saída
- Variáveis de servidor personalizadas
- Curingas
- LogRewrittenUrl

### Variáveis de servidor compatíveis

O middleware dá suporte às seguintes variáveis de servidor do Módulo de Reconfiguração de URL do IIS:

- CONTENT\_LENGTH
- CONTENT\_TYPE
- HTTP\_ACCEPT
- HTTP\_CONNECTION
- HTTP\_COOKIE
- HTTP\_HOST
- HTTP\_REFERER
- HTTP\_URL
- HTTP\_USER\_AGENT
- HTTPS
- LOCAL\_ADDR
- QUERY\_STRING
- REMOTE\_ADDR
- REMOTE\_PORT
- REQUEST\_FILENAME
- REQUEST\_URI

#### NOTE

Também obtenha um [IFileProvider](#) por meio de um [PhysicalFileProvider](#). Essa abordagem pode fornecer maior flexibilidade para o local dos arquivos de regras de reconfiguração. Verifique se os arquivos de regras de reconfiguração são implantados no servidor no caminho fornecido.

```
PhysicalFileProvider fileProvider = new PhysicalFileProvider(Directory.GetCurrentDirectory());
```

## Regra baseada em método

Use [Add](#) para implementar sua própria lógica de regra em um método. [Add](#) expõe o [RewriteContext](#), que disponibiliza o [HttpContext](#) para uso no método. O [RewriteContext.Result](#) determina como o processamento adicional de pipeline é feito. Defina o valor como um dos campos [RuleResult](#) descritos na tabela a seguir.

REWRITECONTEXT.RESULT	AÇÃO
RuleResult.ContinueRules (padrão)	Continuar aplicando regras.
RuleResult.EndResponse	Parar de aplicar regras e enviar a resposta.
RuleResult.SkipRemainingRules	Parar de aplicar regras e enviar o contexto para o próximo middleware.

```

public void Configure(IApplicationBuilder app)
{
    using (StreamReader apacheModRewriteStreamReader =
        File.OpenText("ApacheModRewrite.txt"))
    using (StreamReader iisUrlRewriteStreamReader =
        File.OpenText("IISUrlRewrite.xml"))
    {
        var options = new RewriteOptions()
            .AddRedirect("redirect-rule/(.*)", "redirected/$1")
            .AddRewrite(@"^rewrite-rule/(\d+)/(d+)", "rewritten?var1=$1&var2=$2",
                skipRemainingRules: true)
            .AddApacheModRewrite(apacheModRewriteStreamReader)
            .AddIISUrlRewrite(iisUrlRewriteStreamReader)
            .Add(MethodRules.RedirectXmlFileRequests)
            .Add(MethodRules.RewriteTextFileRequests)
            .Add(new RedirectImageRequests(".png", "/png-images"))
            .Add(new RedirectImageRequests(".jpg", "/jpg-images"));

        app.UseRewriter(options);
    }

    app.UseStaticFiles();

    app.Run(context => context.Response.WriteAsync(
        $"Rewritten or Redirected Url: " +
        $"{context.Request.Path + context.Request.QueryString}"));
}

```

O aplicativo de exemplo demonstra um método que redireciona as solicitações para caminhos que terminam com `.xml`. Se uma solicitação for feita para `/file.xml`, ela será redirecionada para `/xmlfiles/file.xml`. O código de status é definido como *301 – Movido Permanentemente*. Quando o navegador faz uma nova solicitação para `/xmlfiles/file.xml`, o Middleware de Arquivo Estático fornece o arquivo para o cliente por meio da pasta `wwwroot/xmlfiles`. Para um redirecionamento, defina explicitamente o código de status da resposta. Caso contrário, um código de status *200 – OK* será retornado e o redirecionamento não ocorrerá no cliente.

*RewriteRules.cs:*

```

public static void RedirectXmlFileRequests(RewriteContext context)
{
    var request = context.HttpContext.Request;

    // Because the client is redirecting back to the same app, stop
    // processing if the request has already been redirected.
    if (request.Path.StartsWithSegments(new PathString("/xmlfiles")))
    {
        return;
    }

    if (request.Path.Value.EndsWith(".xml", StringComparison.OrdinalIgnoreCase))
    {
        var response = context.HttpContext.Response;
        response.StatusCode = StatusCodes.Status301MovedPermanently;
        context.Result = RuleResult.EndResponse;
        response.Headers[HeaderNames.Location] =
            "/xmlfiles" + request.Path + request.QueryString;
    }
}

```

Essa abordagem também pode reconfigurar as solicitações. O aplicativo de exemplo demonstra a reconfiguração do caminho de qualquer solicitação de arquivo de texto para fornecer o arquivo de texto `file.txt` por meio da pasta `wwwroot`. O Middleware de Arquivo Estático fornece o arquivo com base no caminho de solicitação atualizado:

```

public void Configure(IApplicationBuilder app)
{
    using (StreamReader apacheModRewriteStreamReader =
        File.OpenText("ApacheModRewrite.txt"))
    using (StreamReader iisUrlRewriteStreamReader =
        File.OpenText("IISUrlRewrite.xml"))
    {
        var options = new RewriteOptions()
            .AddRedirect("redirect-rule/(.*)", "redirected/$1")
            .AddRewrite(@"^rewrite-rule/(\d+)/(\d+)", "rewritten?var1=$1&var2=$2",
                skipRemainingRules: true)
            .AddApacheModRewrite(apacheModRewriteStreamReader)
            .AddIISUrlRewrite(iisUrlRewriteStreamReader)
            .Add(MethodRules.RedirectXmlFileRequests)
            .Add(MethodRules.RewriteTextFileRequests)
            .Add(new RedirectImageRequests(".png", "/png-images"))
            .Add(new RedirectImageRequests(".jpg", "/jpg-images"));

        app.UseRewriter(options);
    }

    app.UseStaticFiles();

    app.Run(context => context.Response.WriteAsync(
        $"Rewritten or Redirected Url: " +
        $"{context.Request.Path + context.Request.QueryString}"));
}

```

#### *RewriteRules.cs:*

```

public static void RewriteTextFileRequests(RewriteContext context)
{
    var request = context.HttpContext.Request;

    if (request.Path.Value.EndsWith(".txt", StringComparison.OrdinalIgnoreCase))
    {
        context.Result = RuleResult.SkipRemainingRules;
        request.Path = "/file.txt";
    }
}

```

#### **Regra baseada em IRule**

Use [Add](#) para usar a lógica de regra em uma classe que implementa a interface [IRule](#). [IRule](#) fornece maior flexibilidade em comparação ao uso da abordagem de regra baseada em método. A classe de implementação pode incluir um construtor, que permite passar parâmetros para o método [ApplyRule](#).

```

public void Configure(IApplicationBuilder app)
{
    using (StreamReader apacheModRewriteStreamReader =
        File.OpenText("ApacheModRewrite.txt"))
    using (StreamReader iisUrlRewriteStreamReader =
        File.OpenText("IISUrlRewrite.xml"))
    {
        var options = new RewriteOptions()
            .AddRedirect("redirect-rule/(.*)", "redirected/$1")
            .AddRewrite(@"^rewrite-rule/(\d+)/(\d+)", "rewritten?var1=$1&var2=$2",
                skipRemainingRules: true)
            .AddApacheModRewrite(apacheModRewriteStreamReader)
            .AddIISUrlRewrite(iisUrlRewriteStreamReader)
            .Add(MethodRules.RedirectXmlFileRequests)
            .Add(MethodRules.RewriteTextFileRequests)
            .Add(new RedirectImageRequests(".png", "/png-images"))
            .Add(new RedirectImageRequests(".jpg", "/jpg-images"));

        app.UseRewriter(options);
    }

    app.UseStaticFiles();

    app.Run(context => context.Response.WriteAsync(
        $"Rewritten or Redirected Url: " +
        $"{context.Request.Path + context.Request.QueryString}"));
}

```

Os valores dos parâmetros no aplicativo de exemplo para a `extension` e o `newPath` são verificados para atender a várias condições. A `extension` precisa conter um valor que precisa ser `.png`, `.jpg` ou `.gif`. Se o `newPath` não é válido, uma [ArgumentException](#) é gerada. Se uma solicitação é feita para `image.png`, a solicitação é redirecionada para `/png-images/image.png`. Se uma solicitação é feita para `image.jpg`, a solicitação é redirecionada para `/jpg-images/image.jpg`. O código de status é definido como `301 – Movido Permanentemente` e o `context.Result` é definida para parar o processamento de regras e enviar a resposta.

```

public class RedirectImageRequests : IRule
{
    private readonly string _extension;
    private readonly PathString _newPath;

    public RedirectImageRequests(string extension, string newPath)
    {
        if (string.IsNullOrEmpty(extension))
        {
            throw new ArgumentException(nameof(extension));
        }

        if (!Regex.IsMatch(extension, @"^\.png|jpg|gif$"))
        {
            throw new ArgumentException("Invalid extension", nameof(extension));
        }

        if (!Regex.IsMatch(newPath, @"^/[A-Za-z0-9]+?"))
        {
            throw new ArgumentException("Invalid path", nameof(newPath));
        }

        _extension = extension;
        _newPath = new PathString(newPath);
    }

    public void ApplyRule(RewriteContext context)
    {
        var request = context.HttpContext.Request;

        // Because we're redirecting back to the same app, stop
        // processing if the request has already been redirected
        if (request.Path.StartsWithSegments(_newPath))
        {
            return;
        }

        if (request.Path.Value.EndsWith(_extension, StringComparison.OrdinalIgnoreCase))
        {
            var response = context.HttpContext.Response;
            response.StatusCode = StatusCodes.Status301MovedPermanently;
            context.Result = RuleResult.EndResponse;
            response.Headers[HeaderNames.Location] =
                _newPath + request.Path + request.QueryString;
        }
    }
}

```

Solicitação original: /image.png

The screenshot shows a Microsoft Edge browser window. The address bar displays "localhost:5000/png-images/image.png". Below the address bar, the status bar says "Rewritten or Redirected Url: /png-images/image.png". The browser interface includes standard navigation buttons (back, forward, refresh), a search/address bar, and a toolbar with icons for file operations.

The main content area of the browser shows the Network tab of the developer tools. The table under "Content type" lists two entries:

Name / Path	Protocol	Method	Result / Description
image.png http://localhost:5000/	HTTP	GET	301 Moved Permanently
image.png http://localhost:5000/png-images/	HTTP	GET	200 OK

Details for the first row (Status 301):

- Request URL: http://localhost:5000/image.png
- Request Method: GET
- Status Code: ▲ 301 / Moved Permanently

Details for the second row (Status 200):

- Request Headers

Solicitação original: /image.jpg

Rewritten or Redirected Url: [/jpg-images/image.jpg](#)

Name / Path	Protocol	Method	Result / Description
image.jpg http://localhost:5000/	HTTP	GET	301 Moved Permanently
image.jpg http://localhost:5000/jpg-images/	HTTP	GET	200 OK

Request URL: <http://localhost:5000/image.jpg>  
 Request Method: GET  
 Status Code: ▲ 301 / Moved Permanently

## Exemplos do regex

GOAL	CADEIA DE CARACTERES DO REGEX & EXEMPLO DE CORRESPONDÊNCIA	CADEIA DE CARACTERES DE SUBSTITUIÇÃO & EXEMPLO DE SAÍDA
Reconfigurar o caminho na cadeia de consulta	<code>^path/(.*)(.*)</code> <code>/path/abc/123</code>	<code>path?var1=\$1&amp;var2=\$2</code> <code>/path?var1=abc&amp;var2=123</code>
Barra "/" à direita da faixa	<code>(.*)/\$</code> <code>/path/</code>	<code>\$1</code> <code>/path</code>
Impor barra "/" à direita	<code>(.*[^/])\$</code> <code>/path</code>	<code>\$1/</code> <code>/path/</code>
Evitar a reconfiguração de solicitações específicas	<code>^(.*)(?&lt;!\.axd\$)</code> ou <code>^(?!.*\\.axd\$)(.*\$)</code> Sim: <code>/resource.htm</code> Não: <code>/resource.axd</code>	<code>rewritten/\$1</code> <code>/rewritten/resource.htm</code> <code>/resource.axd</code>
Reorganizar segmentos de URL	<code>path/(.*)(.*)(.*)</code> <code>path/1/2/3</code>	<code>path/\$3/\$2/\$1</code> <code>path/3/2/1</code>
Substituir um segmento de URL	<code>^(.*)/segment2/(.*)</code> <code>/segment1/segment2/segment3</code>	<code>\$1/replaced/\$2</code> <code>/segment1/replaced/segment3</code>

## Recursos adicionais

- [Inicialização de aplicativos](#)
- [Middleware](#)
- [Expressões regulares no .NET](#)
- [Linguagem de expressão regular – referência rápida](#)
- [mod\\_rewrite do Apache](#)
- [Usando o Módulo de Reconfiguração de URL 2.0 \(para IIS\)](#)
- [Referência de configuração do Módulo de Reconfiguração de URL](#)
- [Fórum do Módulo de Reconfiguração de URL do IIS](#)
- [Manter uma estrutura de URL simples](#)
- [10 dicas e truques de reconfiguração de URL](#)
- [Inserir ou não inserir uma barra "/"](#)

# Provedores de arquivos no ASP.NET Core

22/11/2018 • 14 minutes to read • [Edit Online](#)

Por [Steve Smith](#) e [Luke Latham](#)

O ASP.NET Core abstrai o acesso ao sistema de arquivos por meio do uso de provedores de arquivos. Os provedores de arquivos são usados em toda a estrutura do ASP.NET Core:

- [IHostingEnvironment](#) expõe o conteúdo raiz do aplicativo e a raiz da Web como tipos [IFileProvider](#).
- O [middleware de arquivos estáticos](#) usa provedores de arquivos para localizar arquivos estáticos.
- [Razor](#) usa provedores de arquivos para localizar páginas e modos de exibição.
- As ferramentas do .NET Core usam provedores de arquivos e padrões glob para especificar quais arquivos devem ser publicados.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Interfaces de provedor de arquivo

A interface principal é [IFileProvider](#). [IFileProvider](#) expõe métodos para:

- Obter informações sobre o arquivo ([IFileInfo](#)).
- Obter informações sobre o diretório ([IDirectoryContents](#)).
- Configurar notificações de alteração (usando um [IChangeToken](#)).

[IFileInfo](#) fornece métodos e propriedades para trabalhar com arquivos:

- [Exists](#)
- [IsDirectory](#)
- [Nome](#)
- [Length](#) (em bytes)
- Data de [LastModified](#)

Você pode ler o arquivo usando o método [IFileInfo.CreateReadStream](#).

O aplicativo de amostra demonstra como configurar um provedor de arquivos em [Startup.ConfigureServices](#) para uso em todo o aplicativo por meio da [injeção de dependência](#).

## Implementações do provedor de arquivos

Três implementações de [IFileProvider](#) estão disponíveis.

IMPLEMENTAÇÃO	DESCRIÇÃO
<a href="#">PhysicalFileProvider</a>	O provedor físico é usado para acessar os arquivos físicos do sistema.
<a href="#">ManifestEmbeddedFileProvider</a>	O provedor inserido de manifesto é usado para acessar arquivos incorporados em assemblies.
<a href="#">CompositeFileProvider</a>	O provedor composto é usado para fornecer acesso combinado a arquivos e diretórios de um ou mais provedores.

IMPLEMENTAÇÃO	DESCRIÇÃO
PhysicalFileProvider	O provedor físico é usado para acessar os arquivos físicos do sistema.
EmbeddedFileProvider	O provedor inserido é usado para acessar arquivos inseridos em assemblies.
CompositeFileProvider	O provedor composto é usado para fornecer acesso combinado a arquivos e diretórios de um ou mais provedores.

## PhysicalFileProvider

O `PhysicalFileProvider` fornece acesso ao sistema de arquivos físico. O `PhysicalFileProvider` usa o tipo `System.IO.File` (para o provedor físico) e delimita todos os caminhos para um diretório e seus filhos. Esse escopo impede o acesso ao sistema de arquivos fora do diretório especificado e seus filhos. Ao criar uma instância para esse provedor, um caminho de diretório é necessário e serve como o caminho base para todas as solicitações feitas usando o provedor. Você pode criar uma instância de um provedor `PhysicalFileProvider` diretamente, ou pode solicitar um `IFileProvider` em um construtor por meio de uma [injeção de dependência](#).

### Tipos estáticos

O código a seguir mostra como criar um `PhysicalFileProvider` e usá-lo para obter o conteúdo do diretório e as informações do arquivo:

```
var provider = new PhysicalFileProvider(applicationRoot);
var contents = provider.GetDirectoryContents(string.Empty);
var fileInfo = provider.GetFileInfo("wwwroot/js/site.js");
```

Tipos no exemplo anterior:

- `provider` é um `IFileProvider`.
- `contents` é um `IDirectoryContents`.
- `fileInfo` é um `IFileInfo`.

O provedor de arquivos pode ser usado para percorrer o diretório especificado por `applicationRoot` ou chamar `GetFileInfo` para obter as informações de um arquivo. O provedor de arquivos não tem acesso fora do diretório `applicationRoot`.

O aplicativo de amostra cria o provedor na classe `Startup.ConfigureServices` do aplicativo usando `IHostingEnvironment.ContentRootFileProvider`:

```
var physicalProvider = _env.ContentRootFileProvider;
```

### Obter tipos de provedor de arquivos com injeção de dependência

Injetar o provedor em qualquer construtor de classe e atribua-o a um campo local. Use o campo em todos os métodos da classe para acessar arquivos.

No aplicativo de amostra, a classe `IndexModel` recebe uma instância `IFileProvider` para obter o conteúdo do diretório para o caminho base do aplicativo.

`Pages/Index.cshtml.cs`:

```

public class IndexModel : PageModel
{
    private readonly IFileProvider _fileProvider;

    public IndexModel(IFileProvider fileProvider)
    {
        _fileProvider = fileProvider;
    }

    public IDirectoryContents DirectoryContents { get; private set; }

    public void OnGet()
    {
        DirectoryContents = _fileProvider.GetDirectoryContents(string.Empty);
    }
}

```

Os `IDirectoryContents` são iterados na página.

*Pages/Index.cshtml:*

```

<ul>
    @foreach (var item in Model.DirectoryContents)
    {
        if (item.IsDirectory)
        {
            <li><strong>@item.Name</strong></li>
        }
        else
        {
            <li>@item.Name - @item.Length bytes</li>
        }
    }
</ul>

```

No aplicativo de amostra, a classe `HomeController` recebe uma instância `IFileProvider` para obter o conteúdo do diretório para o caminho base do aplicativo.

*Controllers/HomeController.cs:*

```

public class HomeController : Controller
{
    private readonly IFileProvider _fileProvider;

    public HomeController(IFileProvider fileProvider)
    {
        _fileProvider = fileProvider;
    }

    public IActionResult Index()
    {
        var contents = _fileProvider.GetDirectoryContents(string.Empty);

        return View(contents);
    }
}

```

Os `IDirectoryContents` são iterados na exibição.

*Views/Home/Index.cshtml:*

```

<ul>
    @foreach (IFileInfo item in Model)
    {
        if (item.IsDirectory)
        {
            <li><strong>@item.Name</strong></li>
        }
        else
        {
            <li>@item.Name - @item.Length bytes</li>
        }
    }
</ul>

```

## ManifestEmbeddedFileProvider

O [ManifestEmbeddedFileProvider](#) é usado para acessar arquivos inseridos em assemblies. O [ManifestEmbeddedFileProvider](#) usa um manifesto compilado no assembly para reconstruir os caminhos originais dos arquivos inseridos.

### NOTE

O [ManifestEmbeddedFileProvider](#) está disponível no ASP.NET Core 2.1 ou posterior. Para acessar arquivos inseridos em assemblies no ASP.NET Core 2.0 ou anterior, confira a [versão ASP.NET Core 1.x deste tópico](#).

Para gerar um manifesto dos arquivos inseridos, defina a propriedade `<GenerateEmbeddedFilesManifest>` como `true`. Especifique os arquivos para inserir com `<EmbeddedResource>`:

```

<Project Sdk="Microsoft.NET.Sdk.Web">

    <PropertyGroup>
        <TargetFramework>netcoreapp2.1</TargetFramework>
        <GenerateEmbeddedFilesManifest>true</GenerateEmbeddedFilesManifest>
    </PropertyGroup>

    <ItemGroup>
        <PackageReference Include="Microsoft.AspNetCore.App" />
    </ItemGroup>

    <ItemGroup>
        <EmbeddedResource Include="Resource.txt" />
    </ItemGroup>

</Project>

```

Use [padrões glob](#) para especificar um ou mais arquivos a serem inseridos no assembly.

O aplicativo de amostra cria um [ManifestEmbeddedFileProvider](#) e passa o assembly atualmente em execução para seu construtor.

*Startup.cs:*

```

var manifestEmbeddedProvider =
    new ManifestEmbeddedFileProvider(Assembly.GetEntryAssembly());

```

Sobrecargas adicionais permitem:

- Especificar um caminho de arquivo relativo.
- Delimitar os arquivos segundo a data da última modificação.

- Nomear o recurso inserido que contém o manifesto do arquivo inserido.

SOBRECARGA	DESCRIÇÃO
<code>ManifestEmbeddedFileProvider(Assembly, String)</code>	Aceita um parâmetro de caminho relativo <code>root</code> opcional. Especifique o <code>root</code> para delimitar as chamadas para <code>GetDirectoryContents</code> para os recursos no caminho fornecido.
<code>ManifestEmbeddedFileProvider(Assembly, String, DateTimeOffset)</code>	Aceita um parâmetro de caminho relativo <code>root</code> opcional e um parâmetro de data <code>lastModified</code> ( <code>DatetimeOffset</code> ). A data de <code>lastModified</code> tem como escopo a data da última modificação para as instâncias de <code>IFileInfo</code> retornadas pelo <code>IFileProvider</code> .
<code>ManifestEmbeddedFileProvider(Assembly, String, String, DateTimeOffset)</code>	Aceita um caminho relativo <code>root</code> opcional, data de <code>lastModified</code> e parâmetros <code>manifestName</code> . O <code>manifestName</code> representa o nome do recurso inserido que contém o manifesto.

## EmbeddedFileProvider

O `EmbeddedFileProvider` é usado para acessar arquivos inseridos em assemblies. Especifique os arquivos para inserir na propriedade `<EmbeddedResource>` no arquivo do projeto:

```
<ItemGroup>
  <EmbeddedResource Include="Resource.txt" />
</ItemGroup>
```

Use `padrões glob` para especificar um ou mais arquivos a serem inseridos no assembly.

O aplicativo de amostra cria um `EmbeddedFileProvider` e passa o assembly atualmente em execução para seu construtor.

*Startup.cs:*

```
var embeddedProvider = new EmbeddedFileProvider(Assembly.GetEntryAssembly());
```

Recursos inseridos não expõem diretórios. Em vez disso, o caminho para o recurso (por meio de seu namespace) é inserido no nome de arquivo usando separadores `.`. No aplicativo de amostra, o `baseNamespace` é `FileProviderSample..`.

O construtor `EmbeddedFileProvider(Assembly, String)` aceita um parâmetro `baseNamespace` opcional. Especifique o namespace de base para delimitar as chamadas para `GetDirectoryContents` para os recursos no namespace fornecido.

## CompositeFileProvider

O `CompositeFileProvider` combina instâncias de `IFileProvider`, expondo uma interface única para trabalhar com arquivos de vários provedores. Ao criar o `CompositeFileProvider`, passe uma ou mais instâncias de `IFileProvider` para o construtor.

No aplicativo de amostra, um `PhysicalFileProvider` e um `ManifestEmbeddedFileProvider` fornecem arquivos para um `CompositeFileProvider` registrado no contêiner de serviço do aplicativo:

```

var physicalProvider = _env.ContentRootFileProvider;
var manifestEmbeddedProvider =
    new ManifestEmbeddedFileProvider(Assembly.GetEntryAssembly());
var compositeProvider =
    new CompositeFileProvider(physicalProvider, manifestEmbeddedProvider);

services.AddSingleton<IFileProvider>(compositeProvider);

```

No aplicativo de amostra, um `PhysicalFileProvider` e um `EmbeddedFileProvider` fornecem arquivos para um `CompositeFileProvider` registrado no contêiner de serviço do aplicativo:

```

var physicalProvider = _hostingEnvironment.ContentRootFileProvider;
var embeddedProvider = new EmbeddedFileProvider(Assembly.GetEntryAssembly());
var compositeProvider = new CompositeFileProvider(physicalProvider, embeddedProvider);

services.AddSingleton<IFileProvider>(compositeProvider);

```

## Monitorar as alterações

O método `IFileProvider.Watch` proporciona um cenário para monitorar um ou mais arquivos ou diretórios quanto a alterações. `Watch` aceita uma cadeia de caracteres de caminho, que pode usar `padrões glob` para especificar vários arquivos. `Watch` retorna um `IChangeToken`. O token de alteração expõe:

- `HasChanged`: uma propriedade que pode ser inspecionada para determinar se uma alteração ocorreu.
- `RegisterChangeCallback`: chamado quando são detectadas alterações na cadeia de caracteres do caminho especificado. Cada token de alteração chama apenas seu retorno de chamada associado em resposta a uma única alteração. Para permitir o monitoramento constante, use um `TaskCompletionSource` (mostrado abaixo) ou recrie instâncias de `IChangeToken` em resposta a alterações.

No aplicativo de amostra, o aplicativo de console `WatchConsole` é configurado para exibir uma mensagem sempre que um arquivo de texto é modificado:

```

private static PhysicalFileProvider _fileProvider =
    new PhysicalFileProvider(Directory.GetCurrentDirectory());

public static void Main(string[] args)
{
    Console.WriteLine("Monitoring quotes.txt for changes (Ctrl-c to quit)...");

    while (true)
    {
        MainAsync().GetAwaiter().GetResult();
    }
}

private static async Task MainAsync()
{
    IChangeToken token = _fileProvider.Watch("quotes.txt");
    var tcs = new TaskCompletionSource<object>();

    token.RegisterChangeCallback(state =>
        ((TaskCompletionSource<object>)state).TrySetResult(null), tcs);

    await tcs.Task.ConfigureAwait(false);

    Console.WriteLine("quotes.txt changed");
}

```

```

private static PhysicalFileProvider _fileProvider =
    new PhysicalFileProvider(Directory.GetCurrentDirectory());

public static void Main(string[] args)
{
    Console.WriteLine("Monitoring quotes.txt for changes (Ctrl-c to quit)...");

    while (true)
    {
        MainAsync().GetAwaiter().GetResult();
    }
}

private static async Task MainAsync()
{
    IChangeToken token = _fileProvider.Watch("quotes.txt");
    var tcs = new TaskCompletionSource<object>();

    token.RegisterChangeCallback(state =>
        ((TaskCompletionSource<object>)state).TrySetResult(null), tcs);

    await tcs.Task.ConfigureAwait(false);

    Console.WriteLine("quotes.txt changed");
}

```

Alguns sistemas de arquivos, como contêineres do Docker e compartilhamentos de rede, podem não enviar notificações de alteração de forma confiável. Defina a variável de ambiente `DOTNET_USE_POLLING_FILE_WATCHER` como `1` ou `true` para sondar o sistema de arquivos a cada quatro segundos em relação a alterações.

## Padrões glob

Os caminhos do sistema de arquivos usam padrões curinga chamados *padrões glob* (*ou globbing*). Especifique grupos de arquivos com esses padrões. Os dois caracteres curinga são `*` e `**`:

`*`

Corresponde a qualquer coisa no nível da pasta atual, qualquer nome de arquivo ou qualquer extensão de arquivo. As correspondências são terminadas pelos caracteres `/` e `.` no caminho do arquivo.

`**`

Coincide a qualquer coisa em vários níveis de diretório. Pode ser usada para fazer a correspondência recursiva com vários arquivos em uma hierarquia de diretórios.

### Exemplos de padrão glob

`directory/file.txt`

Corresponde a um arquivo específico em um diretório específico.

`directory/*.txt`

Corresponde a todos os arquivos com a extensão `.txt` em um diretório específico.

`directory/*/appsettings.json`

Corresponde a todos os arquivos `appsettings.json` em diretórios que estão exatamente um nível abaixo da pasta *diretório*.

`directory/**/*.*`

Corresponde a todos os arquivos com a extensão `.txt` encontrados em qualquer lugar abaixo da pasta *diretório*.

# Solicitar recursos no ASP.NET Core

25/06/2018 • 5 minutes to read • [Edit Online](#)

Por [Steve Smith](#)

Os detalhes de implementação de servidor Web relacionados a solicitações HTTP e respostas são definidos em interfaces. Essas interfaces são usadas pelas implementações de servidor e pelo middleware para criar e modificar o pipeline de hospedagem do aplicativo.

## Interfaces de recurso

O ASP.NET Core define várias interfaces de recurso HTTP em `Microsoft.AspNetCore.Http.Features`, que são usadas pelos servidores para identificar os recursos para os quais eles dão suporte. As seguintes interfaces de recurso manipulam solicitações e respostas de retorno:

`IHttpRequestFeature` Define a estrutura de uma solicitação HTTP, incluindo o protocolo, o caminho, a cadeia de caracteres de consulta, os cabeçalhos e o corpo.

`IHttpResponseFeature` Define a estrutura de uma resposta HTTP, incluindo o código de status, os cabeçalhos e o corpo da resposta.

`IHttpAuthenticationFeature` Define o suporte para identificar os usuários com base em um `ClaimsPrincipal` e especificando um manipulador de autenticação.

`IHttpUpgradeFeature` Define o suporte para [Upgrades de HTTP](#), que permitem ao cliente especificar quais protocolos adicionais ele desejará usar se o servidor quiser mudar os protocolos.

`IHttpBufferingFeature` Define métodos para desabilitar o buffer de solicitações e/ou respostas.

`IHttpConnectionFeature` Define propriedades para portas e endereços locais e remotos.

`IHttpRequestLifetimeFeature` Define o suporte para anular conexões ou detectar se uma solicitação foi encerrada prematuramente, como por uma desconexão do cliente.

`IHttpSendFileFeature` Define um método para enviar arquivos de forma assíncrona.

`IHttpWebSocketFeature` Define uma API para dar suporte a soquetes da Web.

`IHttpRequestIdentifierFeature` Adiciona uma propriedade que pode ser implementada para identificar as solicitações de forma exclusiva.

`ISessionFeature` Define abstrações `ISessionFactory` e `ISession` para dar suporte a sessões de usuário.

`ITlsConnectionFeature` Define uma API para recuperar os certificados de cliente.

`ITlsTokenBindingFeature` Define métodos para trabalhar com parâmetros de associação de token de TLS.

### NOTE

`ISessionFeature` não é um recurso de servidor, mas é implementado por `SessionMiddleware` (consulte [Gerenciando o estado do aplicativo](#)).

## Coleções de recursos

A propriedade `Features` de `HttpContext` fornece uma interface para obter e definir os recursos HTTP disponíveis para a solicitação atual. Como a coleção de recursos é mutável, mesmo no contexto de uma solicitação, o middleware pode ser usado para modificar a coleção e adicionar suporte para recursos adicionais.

## Middleware e recursos de solicitação

Embora os servidores sejam responsáveis por criar a coleção de recursos, o middleware pode adicionar recursos a essa coleção e consumi-los. Por exemplo, o `StaticFileMiddleware` acessa o recurso `IHttpSendFileFeature`. Se o recurso existir, ele será usado para enviar o arquivo estático solicitado de seu caminho físico. Caso contrário, um método alternativo mais lento será usado para enviar o arquivo. Quando disponível, o `IHttpSendFileFeature` permite que o sistema operacional abra o arquivo e faça uma cópia direta de modo kernel para a placa de rede.

Além disso, o middleware pode adicionar recursos à coleção de recursos estabelecida pelo servidor. Os recursos existentes podem até mesmo ser substituídos pelo middleware, permitindo que o middleware aumente a funcionalidade do servidor. Os recursos adicionados à coleção ficam disponíveis imediatamente para outro middleware ou para o próprio aplicativo subjacente posteriormente no pipeline de solicitação.

Combinando implementações personalizadas de servidor e melhorias específicas de middleware, o conjunto preciso de recursos exigido por um aplicativo pode ser construído. Isso permite que os recursos ausentes sejam adicionados, sem a necessidade de uma alteração no servidor, além de garantir que somente a quantidade mínima de recursos seja exposta, limitando a área da superfície de ataque e melhorando o desempenho.

## Resumo

As interfaces de recurso definem recursos HTTP específicos para os quais uma solicitação específica pode dar suporte. Os servidores definem coleções de recursos e o conjunto inicial de recursos compatíveis com o servidor, mas o middleware pode ser usado para aprimorar esses recursos.

## Recursos adicionais

- [Servidores](#)
- [Middleware](#)
- [OWIN \(Open Web Interface para .NET\)](#)

# Acessar o HttpContext no ASP.NET Core

04/02/2019 • 2 minutes to read • [Edit Online](#)

Os aplicativos ASP.NET Core acessam o `HttpContext` por meio da interface `IHttpContextAccessor` e da implementação padrão do `HttpContextAccessor`. Só é necessário usar o `IHttpContextAccessor` quando você precisar acessar o `HttpContext` em um serviço.

## Usar o HttpContext de Razor Pages

O `PageModel` das Razor Pages expõe a propriedade `HttpContext`:

```
public class AboutModel : PageModel
{
    public string Message { get; set; }

    public void OnGet()
    {
        Message = HttpContext.Request.PathBase;
    }
}
```

## Usar o HttpContext de uma exibição do Razor

As exibições do Razor expõem o `HttpContext` diretamente por meio de uma propriedade `RazorPage.Context` na exibição. O exemplo a seguir recupera o nome de usuário atual em um aplicativo de Intranet usando a Autenticação do Windows:

```
@{
    var username = Context.User.Identity.Name;
}
```

## Usar o HttpContext de um controlador

Os controladores expõem a propriedade `ControllerBase.HttpContext`:

```
public class HomeController : Controller
{
    public IActionResult About()
    {
        var pathBase = HttpContext.Request.PathBase;
        // Do something with the PathBase.

        return View();
    }
}
```

## Usar o HttpContext de middleware

Ao trabalhar com componentes personalizados de middleware, o `HttpContext` é passado para o método `Invoke` ou `InvokeAsync` e pode ser acessado quando o middleware for configurado:

```
public class MyCustomMiddleware
{
    public Task InvokeAsync(HttpContext context)
    {
        // Middleware initialization optionally using HttpContext
    }
}
```

## Usar o HttpContext de componentes personalizados

Para outras estruturas e componentes personalizados que exigem acesso ao `HttpContext`, a abordagem recomendada é registrar uma dependência usando o contêiner integrado de [injeção de dependência](#). O contêiner de injeção de dependência fornece o `IHttpContextAccessor` para todas as classes que o declaram como uma dependência em seus construtores.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc()
        .SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    services.AddHttpContextAccessor();
    services.AddTransient<IUserRepository, UserRepository>();
}
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddSingleton< IHttpContextAccessor, HttpContextAccessor>();
    services.AddTransient<IUserRepository, UserRepository>();
}
```

No exemplo a seguir:

- o `UserRepository` declara sua dependência no `IHttpContextAccessor`.
- A dependência é fornecida quando a injeção de dependência resolve a cadeia de dependências e cria uma instância do `UserRepository`.

```
public class UserRepository : IUserRepository
{
    private readonly IHttpContextAccessor _httpContextAccessor;

    public UserRepository(IHttpContextAccessor httpContextAccessor)
    {
        _httpContextAccessor = httpContextAccessor;
    }

    public void LogCurrentUser()
    {
        var username = _httpContextAccessor.HttpContext.User.Identity.Name;
        service.LogAccessRequest(username);
    }
}
```

# Detectar alterações com tokens de alteração no ASP.NET Core

30/10/2018 • 15 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

Um *token de alteração* é um bloco de construção de uso geral e de baixo nível, usado para controlar as alterações.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Interface `IChangeToken`

`IChangeToken` propaga notificações de que ocorreu uma alteração. `IChangeToken` reside no namespace `Microsoft.Extensions.Primitives`. Para aplicativos que não usam o metapacote `Microsoft.AspNetCore.All` (ASP.NET Core 2.1 ou posterior), veja o pacote NuGet `Microsoft.Extensions.Primitives` no arquivo de projeto.

`IChangeToken` tem duas propriedades:

- `ActiveChangedCallbacks` indica se o token gera retornos de chamada de forma proativa. Se `ActiveChangedCallbacks` é definido como `false`, um retorno de chamada nunca é chamado e o aplicativo precisa sondar `HasChanged` em busca de alterações. Também é possível que um token nunca seja cancelado se não ocorrerem alterações ou o ouvinte de alteração subjacente for removido ou desabilitado.
- `HasChanged` obtém um valor que indica se uma alteração ocorreu.

A interface tem um método, `RegisterChangeCallback(Action<Object>, Object)`, que registra um retorno de chamada que é invocado quando o token é alterado. `HasChanged` precisa ser definido antes de o retorno de chamada ser invocado.

## Classe `ChangeToken`

`ChangeToken` é uma classe estática usada para propagar notificações de que ocorreu uma alteração. `ChangeToken` reside no namespace `Microsoft.Extensions.Primitives`. Para aplicativos que não usam o metapacote `Microsoft.AspNetCore.All`, veja o pacote NuGet `Microsoft.Extensions.Primitives` no arquivo de projeto.

O método `ChangeToken` `OnChange(Func<IChangeToken>, Action)` registra um `Action` para chamar sempre que o token é alterado:

- `Func<IChangeToken>` produz o token.
- `Action` é chamado quando o token é alterado.

`ChangeToken` tem uma sobrecarga `OnChange<TState>(Func<IChangeToken>, Action<TState>, TState)` que usa um parâmetro `TState` adicional que é passado para o consumidor de token `Action`.

`OnChange` retorna um `IDisposable`. A chamada a `Dispose` interrompe a escuta do token de outras alterações e libera os recursos do token.

## Usos de exemplo de tokens de alteração no ASP.NET Core

Tokens de alteração são usados nas áreas proeminentes do monitoramento do ASP.NET Core de alterações em objetos:

- Para monitorar as alterações em arquivos, o método `Watch` de `IFileProvider` cria um `IChangeToken` para os

arquivos especificados ou para pasta a ser inspecionada.

- Tokens `IChangeToken` podem ser adicionados a entradas de cache para disparar remoções do cache após as alterações.
- Para alterações `Options`, a implementação padrão de `OptionsMonitor` de `IOptionsMonitor` tem uma sobrecarga que aceita uma ou mais instâncias `IOptionsChangeTokenSource`. Cada instância retorna um `IChangeToken` para registrar um retorno de chamada de notificação de alteração para o controle de alterações de opções.

## Monitorando alterações de configuração

Por padrão, os modelos do ASP.NET Core usam [arquivos de configuração JSON](#) (`appsettings.json`, `appsettings.Development.json` e `appsettings.Production.json`) para carregar as definições de configuração do aplicativo.

Esses arquivos são configurados com o método de extensão `AddJsonFile(IConfigurationBuilder, String, Boolean, Boolean)` no `ConfigurationBuilder` que aceita um parâmetro `reloadOnChange` (ASP.NET Core 1.1 e posterior).

`reloadOnChange` indica se a configuração deve ser recarregada após alterações de arquivo. Confira essa configuração no método prático `CreateDefaultBuilder` do `WebHost`:

```
config.AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
    .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true, reloadOnChange: true);
```

A configuração baseada em arquivo é representada por `FileConfigurationSource`. A `FileConfigurationSource` usa o `IFileProvider` para monitorar arquivos.

Por padrão, o `IFileMonitor` é fornecido por um `PhysicalFileProvider`, que usa `FileSystemWatcher` para monitorar as alterações no arquivo de configuração.

O aplicativo de exemplo demonstra duas implementações para monitorar as alterações de configuração. Se o arquivo `appsettings.json` é alterado ou a versão do Ambiente do arquivo é alterada, cada implementação executa um código personalizado. O aplicativo de exemplo grava uma mensagem no console.

O `FileSystemWatcher` de um arquivo de configuração pode disparar vários retornos de chamada de token para uma única alteração de arquivo de configuração. A implementação da amostra protege contra esse problema verificando os hashes de arquivo nos arquivos de configuração. A verificação de hashes de arquivo garante que, pelo menos, um dos arquivos de configuração foi alterado antes de executar o código personalizado. A amostra usa o hash de arquivo SHA1 (`Utilities/Utilities.cs`):

```

public static byte[] ComputeHash(string filePath)
{
    var runCount = 1;

    while(runCount < 4)
    {
        try
        {
            if (File.Exists(filePath))
            {
                using (var fs = File.OpenRead(filePath))
                {
                    return System.Security.Cryptography.SHA1.Create().ComputeHash(fs);
                }
            }
            else
            {
                throw new FileNotFoundException();
            }
        }
        catch (IOException ex)
        {
            if (runCount == 3 || ex.HResult != -2147024864)
            {
                throw;
            }
            else
            {
                Thread.Sleep(TimeSpan.FromSeconds(Math.Pow(2, runCount)));
                runCount++;
            }
        }
    }

    return new byte[20];
}

```

Uma nova tentativa é implementada com uma retirada exponencial. A nova tentativa está presente porque o bloqueio de arquivo pode ocorrer, o que impede temporariamente a computação de um novo hash em um dos arquivos.

### **Token de alteração de inicialização simples**

Registre um retorno de chamada `Action` do consumidor de token para notificações de alteração no token de recarregamento de configuração (*Startup.cs*):

```

ChangeToken.OnChange(
    () => config.GetReloadToken(),
    (state) => InvokeChanged(state),
    env);

```

`config.GetReloadToken()` fornece o token. O retorno de chamada é o método `InvokeChanged`:

```

private void InvokeChanged(IHostingEnvironment env)
{
    byte[] appsettingsHash = ComputeHash("appSettings.json");
    byte[] appsettingsEnvHash =
        ComputeHash($"appSettings.{env.EnvironmentName}.json");

    if (!_appsettingsHash.SequenceEqual(appsettingsHash) ||
        !_appsettingsEnvHash.SequenceEqual(appsettingsEnvHash))
    {
        _appsettingsHash = appsettingsHash;
        _appsettingsEnvHash = appsettingsEnvHash;

        WriteConsole("Configuration changed (Simple Startup Change Token)");
    }
}

```

O `state` do retorno de chamada é usado para passar o `IHostingEnvironment`. Isso é útil para determinar o arquivo de configuração JSON `appsettings` correto a ser monitorado, `appsettings.<Environment>.json`. Hashes de arquivo são usados para impedir que a instrução `WriteConsole` seja executada várias vezes, devido a vários retornos de chamada de token quando o arquivo de configuração é alterado somente uma vez.

Esse sistema é executado, desde que o aplicativo esteja em execução e não possa ser desabilitado pelo usuário.

## Monitorando alterações de configuração como um serviço

A amostra implementa:

- Monitoramento de token de inicialização básica.
- Monitoramento como serviço.
- Um mecanismo para habilitar e desabilitar o monitoramento.

A amostra estabelece uma interface `IConfigurationMonitor` (`Extensions/ConfigurationMonitor.cs`):

```

public interface IConfigurationMonitor
{
    bool MonitoringEnabled { get; set; }
    string CurrentState { get; set; }
}

```

O construtor da classe implementada, `ConfigurationMonitor`, registra um retorno de chamada para notificações de alteração:

```

public ConfigurationMonitor(IConfiguration config, IHostingEnvironment env)
{
    _env = env;

    ChangeToken.OnChange< IConfigurationMonitor>(
        () => config.GetReloadToken(),
        InvokeChanged,
        this);
}

public bool MonitoringEnabled { get; set; } = false;
public string CurrentState { get; set; } = "Not monitoring";

```

`config.GetReloadToken()` fornece o token. `InvokeChanged` é o método de retorno de chamada. O `state` nesta instância é uma referência à instância `IConfigurationMonitor` que é usada para acessar o estado de monitoramento. Duas propriedades são usadas:

- `MonitoringEnabled` indica se o retorno de chamada deve executar seu código personalizado.

- `CurrentState` descreve o estado atual de monitoramento para uso na interface do usuário.

O método `InvokeChanged` é semelhante à abordagem anterior, exceto que ele:

- Não executa o código, a menos que `MonitoringEnabled` seja `true`.
- Anota o `state` atual e sua saída `WriteConsole`.

```
private void InvokeChanged(IConfigurationMonitor state)
{
    if (MonitoringEnabled)
    {
        byte[] appsettingsHash = ComputeHash("appSettings.json");
        byte[] appsettingsEnvHash =
            ComputeHash($"appSettings.{_env.EnvironmentName}.json");

        if (!_appsettingsHash.SequenceEqual(appsettingsHash) ||
            !_appsettingsEnvHash.SequenceEqual(appsettingsEnvHash))
        {
            string message = $"State updated at {DateTime.Now}";

            _appsettingsHash = appsettingsHash;
            _appsettingsEnvHash = appsettingsEnvHash;

            WriteConsole($"Configuration changed (ConfigurationMonitor Class) {message}, state:
{state.CurrentState}");
        }
    }
}
```

Uma instância `ConfigurationMonitor` é registrada como um serviço em `ConfigureServices` de `Startup.cs`:

```
services.AddSingleton<IConfigurationMonitor, ConfigurationMonitor>();
```

A página Índice oferece ao usuário o controle sobre o monitoramento de configuração. A instância de `IConfigurationMonitor` é injetada no `IndexModel`:

```
public IndexModel(
    IConfiguration config,
    IConfigurationMonitor monitor,
    FileService fileService)
{
    _config = config;
    _monitor = monitor;
    _fileService = fileService;
}
```

Um botão habilita e desabilita o monitoramento:

```
<button class="btn btn-danger" asp-page-handler="StopMonitoring">Stop Monitoring</button>
```

```

public IActionResult OnPostStartMonitoring()
{
    _monitor.MonitoringEnabled = true;
    _monitor.CurrentState = "Monitoring!";

    return RedirectToAction();
}

public IActionResult OnPostStopMonitoring()
{
    _monitor.MonitoringEnabled = false;
    _monitor.CurrentState = "Not monitoring";

    return RedirectToAction();
}

```

Quando `OnPostStartMonitoring` é disparado, o monitoramento é habilitado e o estado atual é desmarcado.

Quando `OnPostStopMonitoring` é disparado, o monitoramento é desabilitado e o estado é definido para refletir que o monitoramento não está ocorrendo.

## Monitorando alterações de arquivos armazenados em cache

O conteúdo do arquivo pode ser armazenado em cache em memória usando [IMemoryCache](#). O cache na memória é descrito no tópico [Cache na memória](#). Sem realizar etapas adicionais, como a implementação descrita abaixo, dados *obsoletos* (desatualizados) são retornados de um cache se os dados de origem são alterados.

Não levando em conta o status de um arquivo de origem armazenado em cache durante a renovação de um período de [expiração deslizante](#) resulta em dados de cache obsoletos. Cada solicitação de dados renova o período de expiração deslizante, mas o arquivo nunca é recarregado no cache. Os recursos do aplicativo que usam o conteúdo armazenado em cache do arquivo estão sujeitos ao possível recebimento de conteúdo obsoleto.

O uso de tokens de alteração em um cenário de cache de arquivo impede o conteúdo de arquivo obsoleto no cache. O aplicativo de exemplo demonstra uma implementação da abordagem.

A amostra usa `GetFileContent` para:

- Retornar o conteúdo do arquivo.
- Implementar um algoritmo de repetição com retirada exponencial para abranger casos em que um bloqueio de arquivo está impedindo temporariamente a leitura de um arquivo.

*Utilities/Utilities.cs:*

```

public async static Task<string> GetFileContent(string filePath)
{
    var runCount = 1;

    while(runCount < 4)
    {
        try
        {
            if (File.Exists(filePath))
            {
                using (var fileStreamReader = File.OpenText(filePath))
                {
                    return await fileStreamReader.ReadToEndAsync();
                }
            }
            else
            {
                throw new FileNotFoundException();
            }
        }
        catch (IOException ex)
        {
            if (runCount == 3 || ex.HResult != -2147024864)
            {
                throw;
            }
            else
            {
                await Task.Delay(TimeSpan.FromSeconds(Math.Pow(2, runCount)));
                runCount++;
            }
        }
    }

    return null;
}

```

Um `FileService` é criado para manipular pesquisas de arquivos armazenados em cache. A chamada de método `GetFileContent` do serviço tenta obter o conteúdo do arquivo do cache em memória e retorná-lo para o chamador (*Services/FileService.cs*).

Se o conteúdo armazenado em cache não é encontrado com a chave de cache, as seguintes ações são executadas:

1. O conteúdo do arquivo é obtido com `GetFileContent`.
2. Um token de alteração é obtido do provedor de arquivo com `IFileProviders.Watch`. O retorno de chamada do token é disparado quando o arquivo é modificado.
3. O conteúdo do arquivo é armazenado em cache com um período de [expiração deslizante](#). O token de alteração é anexado com `MemoryCacheEntryExtensions.AddExpirationToken` para remover a entrada do cache se o arquivo é alterado enquanto ele é armazenado em cache.

```

public class FileService
{
    private readonly IMemoryCache _cache;
    private readonly IFileProvider _fileProvider;
    private List<string> _tokens = new List<string>();

    public FileService(IMemoryCache cache, IHostingEnvironment env)
    {
        _cache = cache;
        _fileProvider = env.ContentRootFileProvider;
    }

    public async Task<string> GetFileContents(string fileName)
    {
        // For the purposes of this example, files are stored
        // in the content root of the app. To obtain the physical
        // path to a file at the content root, use the
        // ContentRootFileProvider on IHostingEnvironment.
        var filePath = _fileProvider.GetFileInfo(fileName).PhysicalPath;
        string fileContent;

        // Try to obtain the file contents from the cache.
        if (_cache.TryGetValue(filePath, out fileContent))
        {
            return fileContent;
        }

        // The cache doesn't have the entry, so obtain the file
        // contents from the file itself.
        fileContent = await GetFileContent(filePath);

        if (fileContent != null)
        {
            // Obtain a change token from the file provider whose
            // callback is triggered when the file is modified.
            var changeToken = _fileProvider.Watch(fileName);

            // Configure the cache entry options for a five minute
            // sliding expiration and use the change token to
            // expire the file in the cache if the file is
            // modified.
            var cacheEntryOptions = new MemoryCacheEntryOptions()
                .SetSlidingExpiration(TimeSpan.FromMinutes(5))
                .AddExpirationToken(changeToken);

            // Put the file contents into the cache.
            _cache.Set(filePath, fileContent, cacheEntryOptions);
        }

        return fileContent;
    }

    return string.Empty;
}

```

O `FileService` é registrado no contêiner de serviço junto com o serviço de cache da memória (*Startup.cs*):

```

services.AddMemoryCache();
services.AddSingleton<FileService>();

```

O modelo de página carrega o conteúdo do arquivo usando o serviço (*Pages/Index.cshtml.cs*):

```

var fileContent = await _fileService.GetFileContents("poem.txt");

```

## Classe CompositeChangeToken

Para representar uma ou mais instâncias de `IChangeToken` em um único objeto, use a classe `CompositeChangeToken`.

```
var firstCancellationTokenSource = new CancellationTokenSource();
var secondCancellationTokenSource = new CancellationTokenSource();

var firstCancellationToken = firstCancellationTokenSource.Token;
var secondCancellationToken = secondCancellationTokenSource.Token;

var firstCancellationChangeToken = new CancellationChangeToken(firstCancellationToken);
var secondCancellationChangeToken = new CancellationChangeToken(secondCancellationToken);

var compositeChangeToken =
    new CompositeChangeToken(
        new List<IChangeToken>
    {
        firstCancellationChangeToken,
        secondCancellationChangeToken
    });
}
```

`HasChanged` nos relatórios de token compostos `true` se um token representado `HasChanged` é `true`.  
`ActiveChangeCallbacks` nos relatórios de token compostos `true` se um token representado `ActiveChangeCallbacks` é `true`. Se ocorrerem vários eventos de alteração simultâneos, o retorno de chamada de alteração composto será invocado exatamente uma vez.

## Recursos adicionais

- [Memória de cache no ASP.NET Core](#)
- [O cache no ASP.NET Core distribuído](#)
- [Cache de resposta no ASP.NET Core](#)
- [Middleware no ASP.NET Core de cache de resposta](#)
- [Auxiliar de Marca de Cache no ASP.NET Core MVC](#)
- [Auxiliar de Marca de Cache Distribuído no ASP.NET Core](#)

# OWIN (Open Web Interface para .NET) com o ASP.NET Core

10/01/2019 • 9 minutes to read • [Edit Online](#)

Por [Steve Smith](#) e [Rick Anderson](#)

O ASP.NET Core dá suporte para OWIN (Open Web Interface para .NET). O OWIN permite que os aplicativos Web sejam separados dos servidores Web. Ele define uma maneira padrão de usar o middleware em um pipeline para manipular solicitações e respostas associadas. O middleware e os aplicativos ASP.NET Core podem interoperar com aplicativos, servidores e middleware baseados no OWIN.

O OWIN fornece uma camada de desacoplamento que permite duas estruturas com modelos de objeto diferentes para ser usadas juntas. O pacote `Microsoft.AspNetCore.Owin` fornece duas implementações de adaptador:

- ASP.NET Core para OWIN
- OWIN para ASP.NET Core

Isso permite que o ASP.NET Core seja hospedado em um servidor/host compatível com OWIN ou que outros componentes compatíveis com OWIN sejam executados no ASP.NET Core.

## NOTE

O uso desses adaptadores implica um custo de desempenho. Os aplicativos que usam somente componentes do ASP.NET Core não devem usar o pacote `Microsoft.AspNetCore.Owin` ou os adaptadores.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Executando o middleware do OWIN no pipeline do ASP.NET Core

O suporte ao OWIN do ASP.NET Core é implantado como parte do pacote `Microsoft.AspNetCore.Owin`. Importe o suporte ao OWIN para seu projeto instalando esse pacote.

O middleware do OWIN está em conformidade com a [especificação do OWIN](#), que exige uma interface `Func<IDictionary<string, object>, Task>` e a definição de chaves específicas (como `owin.ResponseBody`). O seguinte middleware simples do OWIN exibe "Olá, Mundo":

```
public Task OwinHello(IDictionary<string, object> environment)
{
    string responseText = "Hello World via OWIN";
    byte[] responseBytes = Encoding.UTF8.GetBytes(responseText);

    // OWIN Environment Keys: http://owin.org/spec/spec/owin-1.0.0.html
    var responseStream = (Stream)environment["owin.ResponseBody"];
    var responseHeaders = (IDictionary<string, string[]>)environment["owin.ResponseHeaders"];

    responseHeaders["Content-Length"] = new string[] {
        responseBytes.Length.ToString(CultureInfo.InvariantCulture) };
    responseHeaders["Content-Type"] = new string[] { "text/plain" };

    return responseStream.WriteAsync(responseBytes, 0, responseBytes.Length);
}
```

A assinatura de exemplo retorna uma `Task` e aceita uma `IDictionary<string, object>`, conforme solicitado pelo OWIN.

O código a seguir mostra como adicionar o middleware `OwinHello` (mostrado acima) ao pipeline do ASP.NET Core com o método de extensão `UseOwin`.

```
public void Configure(IApplicationBuilder app)
{
    app.UseOwin(pipeline =>
    {
        pipeline(next => OwinHello);
    });
}
```

Configure outras ações a serem executadas no pipeline do OWIN.

#### NOTE

Os cabeçalhos de resposta devem ser modificados apenas antes da primeira gravação no fluxo de resposta.

#### NOTE

Não é recomendado fazer várias chamadas a `UseOwin` por motivos de desempenho. Os componentes do OWIN funcionarão melhor se forem agrupados.

```
app.UseOwin(pipeline =>
{
    pipeline(async (next) =>
    {
        // do something before
        await OwinHello(new OwinEnvironment(HttpContext));
        // do something after
    });
});
```

## Usando a hospedagem do ASP.NET Core em um servidor baseado no OWIN

Os servidores baseados no OWIN podem hospedar aplicativos ASP.NET Core. Um desses servidores é o [Nowin](#), um servidor Web do OWIN no .NET. Na amostra para este artigo, inclui um projeto que referencia o Nowin e usa-o para criar um `IIServer` com capacidade de auto-hospedar o ASP.NET Core.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Hosting;

namespace NowinSample
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var host = new WebHostBuilder()
                .UseNowin()
                .UseContentRoot(Directory.GetCurrentDirectory())
                .UseIISIntegration()
                .UseStartup<Startup>()
                .Build();

            host.Run();
        }
    }
}
```

O `IIServer` é uma interface que requer uma propriedade `Features` e um método `Start`.

`Start` é responsável por configurar e iniciar o servidor, que, nesse caso, é feito por meio de uma série de chamadas à API fluente que definem endereços analisados do `IIServerAddressesFeature`. Observe que a configuração fluente da variável `_builder` especifica que as solicitações serão manipuladas pelo `appFunc` definido anteriormente no método. Esse `Func` é chamado em cada solicitação para processar as solicitações de entrada.

Adicionaremos também uma extensão `IWebHostBuilder` para facilitar a adição e configuração do servidor `Nowin`.

```

using System;
using Microsoft.AspNetCore.Hosting.Server;
using Microsoft.Extensions.DependencyInjection;
using Nowin;
using NowinSample;

namespace Microsoft.AspNetCore.Hosting
{
    public static class NowinWebHostBuilderExtensions
    {
        public static IWebHostBuilder UseNowin(this IWebHostBuilder builder)
        {
            return builder.ConfigureServices(services =>
            {
                services.AddSingleton<IServer, NowinServer>();
            });
        }

        public static IWebHostBuilder UseNowin(this IWebHostBuilder builder, Action<ServerBuilder> configure)
        {
            builder.ConfigureServices(services =>
            {
                services.Configure(configure);
            });
            return builder.UseNowin();
        }
    }
}

```

Com isso em vigor, invoque a extensão no *Program.cs* para executar um aplicativo ASP.NET Core usando este servidor personalizado:

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Hosting;

namespace NowinSample
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var host = new WebHostBuilder()
                .UseNowin()
                .UseContentRoot(Directory.GetCurrentDirectory())
                .UseIISIntegration()
                .UseStartup<Startup>()
                .Build();

            host.Run();
        }
    }
}

```

Saiba mais sobre os [Servidores ASP.NET Core](#).

## Executar o ASP.NET Core em um servidor baseado no OWIN e usar seu suporte do WebSockets

Outro exemplo de como os recursos dos servidores baseados no OWIN podem ser aproveitados pelo ASP.NET

Core é o acesso a recursos como o WebSockets. O servidor Web do OWIN no .NET usado no exemplo anterior tem suporte para Soquetes da Web internos, que podem ser aproveitados por um aplicativo ASP.NET Core. O exemplo abaixo mostra um aplicativo Web simples que dá suporte a Soquetes da Web e repete tudo o que foi enviado ao servidor por meio do WebSockets.

```
public class Startup
{
    public void Configure(IApplicationBuilder app)
    {
        app.Use(async (context, next) =>
        {
            if (context.WebSockets.IsWebSocketRequest)
            {
                WebSocket webSocket = await context.WebSockets.AcceptWebSocketAsync();
                await EchoWebSocket(webSocket);
            }
            else
            {
                await next();
            }
        });
    }

    app.Run(context =>
    {
        return context.Response.WriteAsync("Hello World");
    });
}

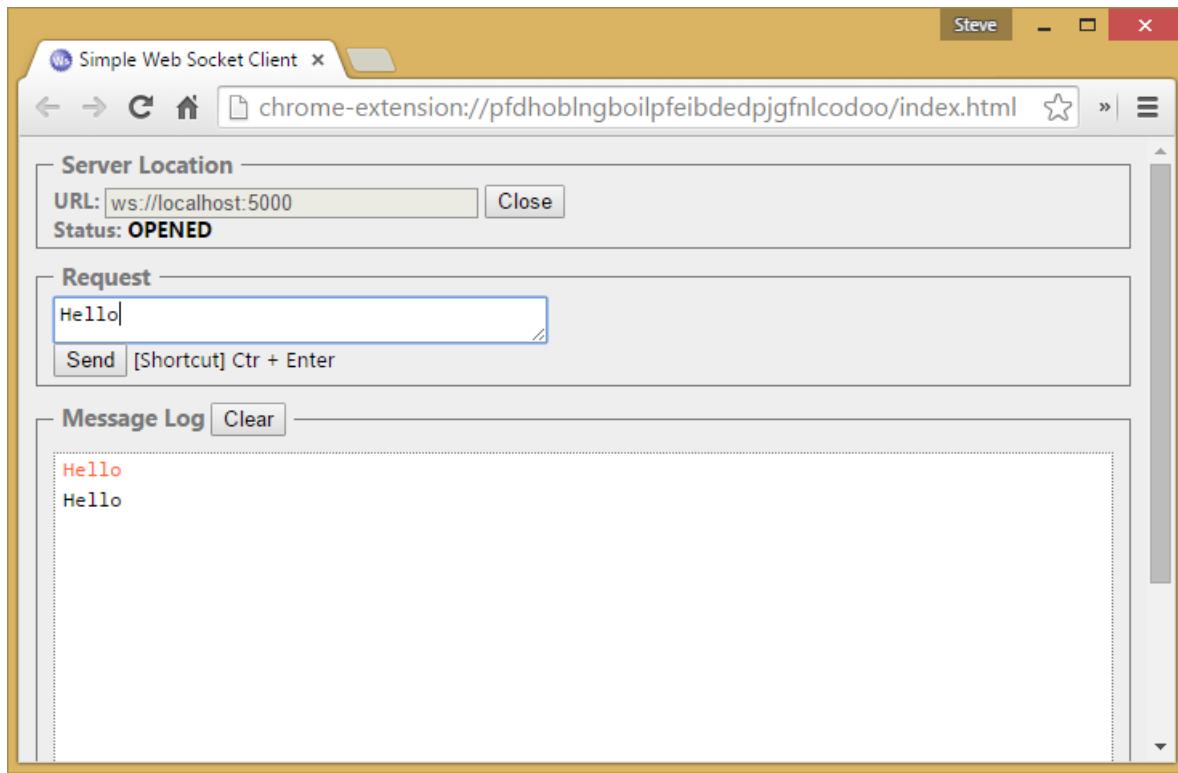
private async Task EchoWebSocket(WebSocket webSocket)
{
    byte[] buffer = new byte[1024];
    WebSocketReceiveResult received = await webSocket.ReceiveAsync(
        new ArraySegment<byte>(buffer), CancellationToken.None);

    while (!webSocket.CloseStatus.HasValue)
    {
        // Echo anything we receive
        await webSocket.SendAsync(new ArraySegment<byte>(buffer, 0, received.Count),
            received.MessageType, received.EndOfMessage, CancellationToken.None);

        received = await webSocket.ReceiveAsync(new ArraySegment<byte>(buffer),
            CancellationToken.None);
    }

    await webSocket.CloseAsync(webSocket.CloseStatus.Value,
        webSocket.CloseStatusDescription, CancellationToken.None);
}
}
```

Essa [amostra](#) é configurada usando o mesmo `NowinServer` que o anterior – a única diferença está em como o aplicativo é configurado em seu método `Configure`. Um teste usando [um cliente simples do WebSocket](#) demonstra o aplicativo:



## Ambiente do OWIN

Você pode construir um ambiente do OWIN usando o `HttpContext`.

```
var environment = new OwinEnvironment(HttpContext);
var features = new OwinFeatureCollection(environment);
```

## Chaves do OWIN

O OWIN depende de um objeto `IDictionary<string,object>` para transmitir informações em uma troca de Solicitação/Resposta HTTP. O ASP.NET Core implementa as chaves listadas abaixo. Consulte a [especificação primária](#), [extensões](#) e as [Diretrizes de chaves do OWIN](#) e [chaves comuns](#).

### Dados de solicitação (OWIN v1.0.0)

CHAVE	VALOR (TIPO)	DESCRIÇÃO
<code>owin.RequestScheme</code>	<code>String</code>	
<code>owin.RequestMethod</code>	<code>String</code>	
<code>owin.RequestPathBase</code>	<code>String</code>	
<code>owin.RequestPath</code>	<code>String</code>	
<code>owin.RequestQueryString</code>	<code>String</code>	
<code>owin.RequestProtocol</code>	<code>String</code>	
<code>owin.RequestHeaders</code>	<code>IDictionary&lt;string,string[]&gt;</code>	

CHAVE	VALOR (TIPO)	DESCRIÇÃO
owin.RequestBody	Stream	

#### Dados de solicitação (OWIN v1.1.0)

CHAVE	VALOR (TIPO)	DESCRIÇÃO
owin.RequestId	String	Opcional

#### Dados de resposta (OWIN v1.0.0)

CHAVE	VALOR (TIPO)	DESCRIÇÃO
owin.ResponseStatusCode	int	Opcional
owin.ResponseReasonPhrase	String	Opcional
owin.ResponseHeaders	IDictionary<string, string[]>	
owin.ResponseBody	Stream	

#### Outros dados (OWIN v1.0.0)

CHAVE	VALOR (TIPO)	DESCRIÇÃO
owin.CallCancelled	CancellationToken	
owin.Version	String	

#### Chaves comuns

CHAVE	VALOR (TIPO)	DESCRIÇÃO
ssl.ClientCertificate	X509Certificate	
ssl.LoadClientCertAsync	Func<Task>	
server.RemoteIpAddress	String	
server.RemotePort	String	
server.LocalIpAddress	String	
server.LocalPort	String	
server.IsLocal	bool	
server.OnSendingHeaders	Action<Action<object>, object>	

#### SendFiles v0.3.0

CHAVE	VALOR (TIPO)	DESCRIÇÃO
sendfile.SendAsync	Consulte <a href="#">Assinatura do delegado</a>	Por solicitação

## Opaque v0.3.0

CHAVE	VALOR (TIPO)	DESCRIÇÃO
opaque.Version	<code>String</code>	
opaque.Upgrade	<code>OpaqueUpgrade</code>	Consulte <a href="#">Assinatura do delegado</a>
opaque.Stream	<code>Stream</code>	
opaque.CallCancelled	<code>CancellationToken</code>	

## WebSocket v0.3.0

CHAVE	VALOR (TIPO)	DESCRIÇÃO
websocket.Version	<code>String</code>	
websocket.Accept	<code>WebSocketAccept</code>	Consulte <a href="#">Assinatura do delegado</a>
websocket.AcceptAlt		Sem especificação
websocket.SubProtocol	<code>String</code>	Consulte a etapa 5.5 <a href="#">RFC6455 Seção 4.2.2</a>
websocket.SendAsync	<code>WebSocketSendAsync</code>	Consulte <a href="#">Assinatura do delegado</a>
websocket.ReceiveAsync	<code>WebSocketReceiveAsync</code>	Consulte <a href="#">Assinatura do delegado</a>
websocket.CloseAsync	<code>WebSocketCloseAsync</code>	Consulte <a href="#">Assinatura do delegado</a>
websocket.CallCancelled	<code>CancellationToken</code>	
websocket.ClientCloseStatus	<code>int</code>	Opcional
websocket.ClientCloseDescription	<code>String</code>	Opcional

## Recursos adicionais

- [Middleware](#)
- [Servidores](#)

# Tarefas em segundo plano com serviços hospedados no ASP.NET Core

12/12/2018 • 9 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

No ASP.NET Core, as tarefas em segundo plano podem ser implementadas como *serviços hospedados*. Um serviço hospedado é uma classe com lógica de tarefa em segundo plano que implementa a interface [IHostedService](#). Este tópico fornece três exemplos de serviço hospedado:

- Tarefa em segundo plano que é executada com um temporizador.
- Serviço hospedado que ativa um serviço com escopo. O serviço com escopo pode usar a injeção de dependência.
- Tarefas em segundo plano na fila que são executadas sequencialmente.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

Este aplicativo de exemplo é fornecido em duas versões:

- Host da Web – O Host da Web é útil para hospedar aplicativos web. O código de exemplo mostrado neste tópico é da versão do host da web do exemplo. Para obter mais informações, consulte o tópico [Host da Web](#).
- Host Genérico – O Host Genérico é novo no ASP.NET Core 2.1. Para obter mais informações, confira o tópico [Host Genérico](#).

## Pacote

Referencie o [metapacote Microsoft.AspNetCore.App](#) ou adicione uma referência de pacote ao pacote [Microsoft.Extensions.Hosting](#).

## Interface [IHostedService](#)

Os serviços hospedados implementam a interface [IHostedService](#). A interface define dois métodos para objetos que são gerenciados pelo host:

- [StartAsync\(CancellationToken\)](#) – `StartAsync` contém a lógica para iniciar a tarefa em segundo plano. Ao usar o [host Web](#), `StartAsync` é chamado depois que o servidor é iniciado e [IApplicationLifetime.ApplicationStarted](#) é disparado. Ao usar o [host genérico](#), `StartAsync` é chamado antes de `ApplicationStarted` ser disparado.
- [StopAsync\(CancellationToken\)](#) – é disparado quando o host está executando um desligamento normal. `StopAsync` contém a lógica para encerrar a tarefa em segundo plano. Implemente o [IDisposable](#) e os [finalizadores \(destruidores\)](#) para descartar todos os recursos não gerenciados.

O token de cancelamento tem um tempo limite padrão de cinco segundos para indicar que o processo de desligamento não deve mais ser normal. Quando for solicitado um cancelamento no token:

- Todas as demais operações em segundo plano que o aplicativo estiver executando deverão ser anuladas.
- Todos os métodos chamados em `StopAsync` deverão retornar imediatamente.

No entanto, as tarefas não são abandonadas após a solicitação de cancelamento—o chamador aguarda a conclusão de todas as tarefas.

Se o aplicativo for desligado inesperadamente (por exemplo, em uma falha do processo do aplicativo),

`StopAsync` não poderá ser chamado. Portanto, os métodos chamados ou operações realizadas em

`StopAsync` talvez não ocorram.

Para estender o tempo limite de desligamento padrão de cinco segundos, defina:

- o [ShutdownTimeout](#) ao usar o Host Genérico. Para obter mais informações, consulte [Host Genérico .NET](#).
- o Desabilite a configuração de tempo limite de desligamento do host ao usar o Host da Web. Para obter mais informações, consulte [Host da Web do ASP.NET Core](#).

O serviço hospedado é ativado uma única vez na inicialização do aplicativo e desligado normalmente durante o desligamento do aplicativo. Se um erro for gerado durante a execução da tarefa em segundo plano, `Dispose` deverá ser chamado mesmo se `StopAsync` não for chamado.

## Tarefas em segundo plano temporizadas

Uma tarefa em segundo plano temporizada usa a classe [System.Threading.Timer](#). O temporizador dispara o método `DoWork` da tarefa. O temporizador é desabilitado em `StopAsync` e descartado quando o contêiner de serviço é descartado em `Dispose`:

```
internal class TimedHostedService : IHostedService, IDisposable
{
    private readonly ILogger _logger;
    private Timer _timer;

    public TimedHostedService(ILogger<TimedHostedService> logger)
    {
        _logger = logger;
    }

    public Task StartAsync(CancellationToken cancellationToken)
    {
        _logger.LogInformation("Timed Background Service is starting.");

        _timer = new Timer(DoWork, null, TimeSpan.Zero,
            TimeSpan.FromSeconds(5));

        return Task.CompletedTask;
    }

    private void DoWork(object state)
    {
        _logger.LogInformation("Timed Background Service is working.");
    }

    public Task StopAsync(CancellationToken cancellationToken)
    {
        _logger.LogInformation("Timed Background Service is stopping.");

        _timer?.Change(Timeout.Infinite, 0);

        return Task.CompletedTask;
    }

    public void Dispose()
    {
        _timer?.Dispose();
    }
}
```

O serviço está registrado em `Startup.ConfigureServices` com o método de extensão `AddHostedService`:

```
services.AddHostedService<TimedHostedService>();
```

## Consumindo um serviço com escopo em uma tarefa em segundo plano

Para usar os serviços com escopo dentro de um `IHostedService`, crie um escopo. Por padrão, nenhum escopo é criado para um serviço hospedado.

O serviço da tarefa em segundo plano com escopo contém a lógica da tarefa em segundo plano. No exemplo a seguir, um `ILogger` é injetado no serviço:

```
internal interface IScopedProcessingService
{
    void DoWork();
}

internal class ScopedProcessingService : IScopedProcessingService
{
    private readonly ILogger _logger;

    public ScopedProcessingService(ILogger<ScopedProcessingService> logger)
    {
        _logger = logger;
    }

    public void DoWork()
    {
        _logger.LogInformation("Scoped Processing Service is working.");
    }
}
```

O serviço hospedado cria um escopo para resolver o serviço da tarefa em segundo plano com escopo para chamar seu método `DoWork`:

```

internal class ConsumeScopedServiceHostedService : IHostedService
{
    private readonly ILogger _logger;

    public ConsumeScopedServiceHostedService(IServiceProvider services,
        ILogger<ConsumeScopedServiceHostedService> logger)
    {
        Services = services;
        _logger = logger;
    }

    public IServiceProvider Services { get; }

    public Task StartAsync(CancellationToken cancellationToken)
    {
        _logger.LogInformation(
            "Consume Scoped Service Hosted Service is starting.");

        DoWork();

        return Task.CompletedTask;
    }

    private void DoWork()
    {
        _logger.LogInformation(
            "Consume Scoped Service Hosted Service is working.");

        using (var scope = Services.CreateScope())
        {
            var scopedProcessingService =
                scope.ServiceProvider
                    .GetRequiredService<IScopedProcessingService>();

            scopedProcessingService.DoWork();
        }
    }

    public Task StopAsync(CancellationToken cancellationToken)
    {
        _logger.LogInformation(
            "Consume Scoped Service Hosted Service is stopping.");

        return Task.CompletedTask;
    }
}

```

Os serviços são registrados em `Startup.ConfigureServices`. A implementação `IHostedService` é registrada com o método de extensão `AddHostedService`:

```

services.AddHostedService<ConsumeScopedServiceHostedService>();
services.AddScoped<IScopedProcessingService, ScopedProcessingService>();

```

## Tarefas em segundo plano na fila

Uma fila de tarefas em segundo plano é baseada no [QueueBackgroundWorkItem](#) do .NET 4.x (provisoriamente agendado para ser integrado ao ASP.NET Core 3.0):

```

public interface IBackgroundTaskQueue
{
    void QueueBackgroundWorkItem(Func< CancellationToken, Task> workItem);

    Task< Func< CancellationToken, Task>> DequeueAsync(
        CancellationToken cancellationToken);
}

public class BackgroundTaskQueue : IBackgroundTaskQueue
{
    private ConcurrentQueue< Func< CancellationToken, Task>> _workItems =
        new ConcurrentQueue< Func< CancellationToken, Task>>();
    private SemaphoreSlim _signal = new SemaphoreSlim(0);

    public void QueueBackgroundWorkItem(
        Func< CancellationToken, Task> workItem)
    {
        if (workItem == null)
        {
            throw new ArgumentNullException(nameof(workItem));
        }

        _workItems.Enqueue(workItem);
        _signal.Release();
    }

    public async Task< Func< CancellationToken, Task>> DequeueAsync(
        CancellationToken cancellationToken)
    {
        await _signal.WaitAsync(cancellationToken);
        _workItems.TryDequeue(out var workItem);

        return workItem;
    }
}

```

No `QueueHostedService`, as tarefas em segundo plano na fila são removidas da fila e executadas como um `BackgroundService`, que é uma classe base para implementar um `IHostedService` de longa execução:

```

public class QueuedHostedService : BackgroundService
{
    private readonly ILogger _logger;

    public QueuedHostedService(IBackgroundTaskQueue taskQueue,
        ILoggerFactory loggerFactory)
    {
        TaskQueue = taskQueue;
        _logger = loggerFactory.CreateLogger<QueuedHostedService>();
    }

    public IBackgroundTaskQueue TaskQueue { get; }

    protected override Task ExecuteAsync(
        CancellationToken cancellationToken)
    {
        _logger.LogInformation("Queued Hosted Service is starting.");

        while (!cancellationToken.IsCancellationRequested)
        {
            var workItem = await TaskQueue.DequeueAsync(cancellationToken);

            try
            {
                await workItem(cancellationToken);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex,
                    $"Error occurred executing {nameof(workItem)}.");
            }
        }

        _logger.LogInformation("Queued Hosted Service is stopping.");
    }
}

```

Os serviços são registrados em `Startup.ConfigureServices`. A implementação `IHostedService` é registrada com o método de extensão `AddHostedService`:

```

services.AddHostedService<QueuedHostedService>();
services.AddSingleton<IBackgroundTaskQueue, BackgroundTaskQueue>();

```

Na classe de modelo de página de índice, a `IBackgroundTaskQueue` é injetada no construtor e atribuída à `Queue`:

```

public IndexModel(IBackgroundTaskQueue queue,
    IApplicationLifetime appLifetime,
    ILogger<IndexModel> logger)
{
    Queue = queue;
    _appLifetime = appLifetime;
    _logger = logger;
}

public IBackgroundTaskQueue Queue { get; }

```

Quando o botão **Adicionar Tarefa** é selecionado na página de índice, o método `OnPostAddTask` é executado. O `QueueBackgroundWorkItem` é chamado para enfileirar o item de trabalho:

```
public IActionResult OnPostAddTask()
{
    Queue.QueueBackgroundWorkItem(async token =>
    {
        var guid = Guid.NewGuid().ToString();

        for (int delayLoop = 0; delayLoop < 3; delayLoop++)
        {
            _logger.LogInformation(
                $"Queued Background Task {guid} is running. {delayLoop}/3");
            await Task.Delay(TimeSpan.FromSeconds(5), token);
        }

        _logger.LogInformation(
            $"Queued Background Task {guid} is complete. 3/3");
    });

    return RedirectToPage();
}
```

## Recursos adicionais

- [Implementar tarefas em segundo plano em microserviços com IHostedService e a classe BackgroundService](#)
- [System.Threading.Timer](#)

# Usar assemblies de inicialização de hospedagem no ASP.NET Core

10/01/2019 • 28 minutes to read • [Edit Online](#)

Por Luke Latham e Pavel Krymets

Uma implementação `IHostingStartup` (inicialização de hospedagem) adiciona melhorias a um aplicativo durante a inicialização de um assembly externo. Por exemplo, uma biblioteca externa pode usar uma implementação de inicialização de hospedagem para fornecer serviços ou provedores de configuração adicionais a um aplicativo. `IHostingStartup` está disponível no ASP.NET Core 2.0 ou posterior.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Atributo `HostingStartup`

Um atributo `HostingStartup` indica a presença de um assembly de inicialização de hospedagem para ativar em tempo de execução.

O assembly de entrada ou o assembly que contém a classe `Startup` é automaticamente examinado para o atributo `HostingStartup`. A lista de assemblies a ser pesquisada para os atributos `HostingStartup` é carregada no tempo de execução da configuração em `WebHostDefaults.HostingStartupAssembliesKey`. A lista de assemblies para excluir da descoberta é carregada de `WebHostDefaults.HostingStartupExcludeAssembliesKey`. Para obter mais informações, confira [Host da Web: Hospelando assemblies de inicialização](#) e [Host da Web: hospedando assemblies de exclusão de inicialização](#).

No exemplo a seguir, o namespace do assembly de inicialização de hospedagem é `startupEnhancement`. A classe que contém o código de inicialização de hospedagem é `StartupEnhancement.HostingStartup`:

```
[assembly: HostingStartup(typeof(StartupEnhancement.StartupEnhancement.HostingStartup))]
```

O atributo `HostingStartup` normalmente está localizado no arquivo de classe de implementação `IHostingStartup` do assembly de inicialização de hospedagem.

## Descobrir assemblies de inicialização de hospedagem carregados

Para descobrir os assemblies de inicialização de hospedagem carregados, habilite o registro em log e verifique os logs do aplicativo. Erros que ocorrem quando os assemblies carregados são registrados em log. Os assemblies de inicialização de hospedagem carregados são registrados em log no nível Depuração e todos os erros são registrados.

## Desabilitar o carregamento automático de assemblies de inicialização de hospedagem

Para desabilitar o carregamento automático de assemblies de inicialização de hospedagem, use uma das seguintes abordagens:

- Para impedir o carregamento de todos os assemblies de inicialização de hospedagem, defina o seguinte para `true` ou `1`:
  - Configuração do host [Impedir inicialização de hospedagem](#).

- A variável de ambiente `ASPNETCORE_PREVENTHOSTINGSTARTUP`.
- Para evitar o carregamento de assemblies específicos de inicialização de hospedagem, defina uma das opções a seguir como uma cadeia de caracteres delimitada por ponto e vírgula de assemblies de inicialização de hospedagem para excluir na inicialização:
  - Configuração do host [Assemblies de exclusão de inicialização de hospedagem](#).
  - A variável de ambiente `ASPNETCORE_HOSTINGSTARTUPEXCLUDEASSEMBLIES`.

Para desabilitar o carregamento automático de assemblies de inicialização de hospedagem, defina um dos seguintes como `true` ou `1`:

- Configuração do host [Impedir inicialização de hospedagem](#).
- A variável de ambiente `ASPNETCORE_PREVENTHOSTINGSTARTUP`.

Se a configuração do host e a variável de ambiente estiverem definidas, a configuração do host controlará o comportamento.

A desabilitação de assemblies de inicialização de hospedagem usando a configuração do host ou a variável de ambiente desabilita o assembly globalmente e poderá desabilitar várias características de um aplicativo.

## Projeto

Crie uma inicialização de hospedagem com qualquer um dos seguintes tipos de projeto:

- [Biblioteca de classes](#)
- [Aplicativo de console sem um ponto de entrada](#)

### Biblioteca de classes

Uma melhoria da inicialização de hospedagem pode ser fornecida em uma biblioteca de classes. A biblioteca contém um atributo `HostingStartup`.

O [código de exemplo](#) inclui um aplicativo Razor Pages, *HostingStartupApp* e uma biblioteca de classes, *HostingStartupLibrary*. A biblioteca de classes:

- Contém uma classe de inicialização de hospedagem, `ServiceKeyInjection`, que implementa `IHostingStartup`. `ServiceKeyInjection` adiciona um par de cadeias de caracteres de serviço à configuração do aplicativo usando o provedor de configuração na memória ([AddInMemoryCollection](#)).
- Inclui um atributo `HostingStartup` que identifica o namespace e a classe de inicialização de hospedagem.

O método `Configure` da classe `ServiceKeyInjection` usa um `IWebHostBuilder` para adicionar melhorias a um aplicativo. `IHostingStartup.Configure` no assembly de inicialização de hospedagem é chamado pelo tempo de execução antes de `Startup.Configure` no código do usuário, o que permite que o código de usuário substitua qualquer configuração fornecida pelo assembly de inicialização de hospedagem.

*HostingStartupLibrary/ServiceKeyInjection.cs*:

```
[assembly: HostingStartup(typeof(HostingStartupLibrary.ServiceKeyInjection))]

namespace HostingStartupLibrary
{
    public class ServiceKeyInjection : IHostingStartup
    {
        public void Configure(IWebHostBuilder builder)
        {
            builder.ConfigureAppConfiguration(config =>
            {
                var dict = new Dictionary<string, string>
                {
                    {"DevAccount_FromLibrary", "DEV_1111111-1111"}, 
                    {"ProdAccount_FromLibrary", "PROD_2222222-2222"} 
                };

                config.AddInMemoryCollection(dict);
            });
        }
    }
}
```

A página de índice do aplicativo lê e renderiza os valores de configuração para as duas chaves definidas pelo assembly de inicialização de hospedagem da biblioteca de classes:

*HostingStartupApp/Pages/Index.cshtml.cs*:

```
public class IndexModel : PageModel
{
    public IndexModel(IConfiguration config)
    {
        ServiceKey_Development_Library = config["DevAccount_FromLibrary"];
        ServiceKey_Production_Library = config["ProdAccount_FromLibrary"];
        ServiceKey_Development_Package = config["DevAccount_FromPackage"];
        ServiceKey_Production_Package = config["ProdAccount_FromPackage"];
    }

    public string ServiceKey_Development_Library { get; private set; }
    public string ServiceKey_Production_Library { get; private set; }
    public string ServiceKey_Development_Package { get; private set; }
    public string ServiceKey_Production_Package { get; private set; }

    public void OnGet()
    {
    }
}
```

O [código de exemplo](#) também inclui um projeto de pacote do NuGet que fornece uma inicialização de hospedagem separada, *HostingStartupPackage*. O pacote tem as mesmas características da biblioteca de classes descrita anteriormente. O pacote:

- Contém uma classe de inicialização de hospedagem, `ServiceKeyInjection`, que implementa `IHostingStartup`. `ServiceKeyInjection` adiciona um par de cadeias de caracteres de serviço para a configuração do aplicativo.
- Inclui um atributo `HostingStartup`.

*HostingStartupPackage/ServiceKeyInjection.cs*:

```
[assembly: HostingStartup(typeof(HostingStartupPackage.ServiceKeyInjection))]

namespace HostingStartupPackage
{
    public class ServiceKeyInjection : IHostingStartup
    {
        public void Configure(IWebHostBuilder builder)
        {
            builder.ConfigureAppConfiguration(config =>
            {
                var dict = new Dictionary<string, string>
                {
                    {"DevAccount_FromPackage", "DEV_3333333-3333"}, 
                    {"ProdAccount_FromPackage", "PROD_4444444-4444"} 
                };

                config.AddInMemoryCollection(dict);
            });
        }
    }
}
```

A página de índice do aplicativo lê e renderiza os valores de configuração para as duas chaves definidas pelo assembly de inicialização de hospedagem do pacote:

*HostingStartupApp/Pages/Index.cshtml.cs:*

```
public class IndexModel : PageModel
{
    public IndexModel(IConfiguration config)
    {
        ServiceKey_Development_Library = config["DevAccount_FromLibrary"];
        ServiceKey_Production_Library = config["ProdAccount_FromLibrary"];
        ServiceKey_Development_Package = config["DevAccount_FromPackage"];
        ServiceKey_Production_Package = config["ProdAccount_FromPackage"];
    }

    public string ServiceKey_Development_Library { get; private set; }
    public string ServiceKey_Production_Library { get; private set; }
    public string ServiceKey_Development_Package { get; private set; }
    public string ServiceKey_Production_Package { get; private set; }

    public void OnGet()
    {
    }
}
```

## Aplicativo de console sem um ponto de entrada

*Essa abordagem só está disponível para aplicativos .NET Core, não para .NET Framework.*

Uma melhoria de inicialização de hospedagem dinâmica que não requer uma referência de tempo de compilação para a ativação pode ser fornecida em um aplicativo de console sem um ponto de entrada que contenha um atributo `HostingStartup`. Publicar o aplicativo de console produz um assembly de inicialização de hospedagem que pode ser consumido do repositório de tempo de execução.

Um aplicativo de console sem um ponto de entrada é usado nesse processo porque:

- Um arquivo de dependências é necessário para consumir a inicialização de hospedagem no assembly de inicialização de hospedagem. Um arquivo de dependências é um ativo de aplicativo executável que é produzido ao publicar um aplicativo, não uma biblioteca.
- Uma biblioteca não pode ser adicionada diretamente ao [repositório de pacotes de tempo de execução](#), o que

exige um projeto executável que tem como alvo o tempo de execução compartilhado.

Na criação de uma inicialização de hospedagem dinâmica:

- Um assembly de inicialização de hospedagem é criado no aplicativo de console sem um ponto de entrada que:
  - Inclui uma classe que contém a implementação de `IHostingStartup`.
  - Inclui um atributo `HostingStartup` para identificar a classe de implementação de `IHostingStartup`.
- O aplicativo de console é publicado para obter as dependências da inicialização de hospedagem. Uma consequência de publicar o aplicativo de console é que as dependências não utilizadas são cortadas do arquivo de dependências.
- O arquivo de dependências é modificado para definir o local de tempo de execução do assembly de inicialização de hospedagem.
- O assembly de inicialização de hospedagem e seu arquivo de dependências são colocados no repositório do pacote de tempo de execução. Para descobrir o assembly de inicialização de hospedagem e seu arquivo de dependências, eles são listados em um par de variáveis de ambiente.

O assembly de aplicativo do console referencia o pacote `Microsoft.AspNetCore.Hosting.Abstractions`:

```
<Project Sdk="Microsoft.NET.Sdk">

    <PropertyGroup>
        <TargetFramework>netcoreapp2.1</TargetFramework>
    </PropertyGroup>

    <ItemGroup>
        <PackageReference Include="Microsoft.AspNetCore.Hosting.Abstractions"
            Version="2.1.1" />
    </ItemGroup>

</Project>
```

Um atributo `HostingStartup` identifica uma classe como uma implementação de `IHostingStartup` para o carregamento e a execução durante a criação do `IWebHost`. No seguinte exemplo, o namespace é `StartupEnhancement` e a classe é `StartupEnhancementHostingStartup`:

```
[assembly: HostingStartup(typeof(StartupEnhancement.StartupEnhancementHostingStartup))]
```

Uma classe implementa `IHostingStartup`. O método `Configure` da classe usa um `IWebHostBuilder` para adicionar melhorias a um aplicativo. `IHostingStartup.Configure` no assembly de inicialização de hospedagem é chamado pelo tempo de execução antes de `Startup.Configure` no código do usuário, o que permite que o código de usuário substitua qualquer configuração fornecida pelo assembly de inicialização de hospedagem.

```
namespace StartupEnhancement
{
    public class StartupEnhancementHostingStartup : IHostingStartup
    {
        public void Configure(IWebHostBuilder builder)
        {
            // Use the IWebHostBuilder to add app enhancements.
        }
    }
}
```

Ao criar um projeto `IHostingStartup`, o arquivo de dependências (`*.deps.json`) define o local `runtime` do assembly como a pasta `bin`:

```
"targets": {
  ".NETCoreApp,Version=v2.1": {
    "StartupEnhancement/1.0.0": {
      "dependencies": {
        "Microsoft.AspNetCore.Hosting.Abstractions": "2.1.1"
      },
      "runtime": {
        "StartupEnhancement.dll": {}
      }
    }
  }
}
```

Apenas uma parte do arquivo é mostrada. O nome do assembly no exemplo é `StartupEnhancement`.

## Especificar o assembly de inicialização de hospedagem

Para uma biblioteca de classes ou inicialização de hospedagem fornecida pelo aplicativo de console, especifique o nome do assembly de inicialização de hospedagem na variável de ambiente

`ASPNETCORE_HOSTINGSTARTUPASSEMBLIES`. A variável de ambiente é uma lista de assemblies delimitada por ponto e vírgula.

Apenas assemblies de inicialização de hospedagem são examinados quanto ao atributo `HostingStartup`. Para o aplicativo de exemplo, *HostingStartupApp*, para descobrir as inicializações de hospedagem descritas anteriormente, a variável de ambiente é definida como o seguinte valor:

```
HostingStartupLibrary;HostingStartupPackage;StartupDiagnostics
```

Um assembly de inicialização de hospedagem também pode ser definido usando a configuração do host [Assemblies de inicialização de hospedagem](#).

Quando há vários assemblies de inicialização de hospedagem, os métodos `Configure` são executados na ordem em que os assemblies são listados.

## Ativação

As opções para ativação da inicialização de hospedagem são:

- [Repositório de tempo de execução](#) – A ativação não requer uma referência de tempo de compilação para a ativação. O aplicativo de exemplo coloca os arquivos de dependências e o assembly de inicialização de hospedagem em uma pasta, *implantação*, para facilitar a implantação da inicialização de hospedagem em um ambiente multiccomputador. A pasta *implantação* também inclui um script do PowerShell que cria ou modifica variáveis de ambiente no sistema de implantação para habilitar a inicialização de hospedagem.
- Referência de tempo de compilação necessária para a ativação
  - [Pacote do NuGet](#)
  - [Pasta Lixeira do projeto](#)

### Repositório de tempo de execução

A implementação de inicialização de hospedagem é colocada no [repositório de tempo de execução](#). Uma referência de tempo de compilação para o assembly não é exigida pelo aplicativo aprimorado.

Depois que a inicialização de hospedagem é compilada, um repositório de tempo de execução é gerado, usando o arquivo de projeto do manifesto e o comando do [dotnet store](#).

```
dotnet store --manifest {MANIFEST FILE} --runtime {RUNTIME IDENTIFIER} --output {OUTPUT LOCATION} --skip-optimization
```

No aplicativo de exemplo (projeto *RuntimeStore*) é usado o seguinte comando:

```
dotnet store --manifest store.manifest.csproj --runtime win7-x64 --output ./deployment/store --skip-optimization
```

Para o tempo de execução descobrir o repositório de tempo de execução, o local do repositório de tempo de execução é adicionado à variável de ambiente `DOTNET_SHARED_STORE`.

### Modificar e colocar o arquivo de dependências da inicialização de hospedagem

Para ativar o aprimoramento sem uma referência de pacote ao aprimoramento, especifique as dependências adicionais do tempo de execução com `additionalDeps`. `additionalDeps` permite que você:

- Amplie o grafo de biblioteca do aplicativo, fornecendo um conjunto de arquivos `*.deps.json` adicionais a serem mesclados com o arquivo `*.deps.json` próprio do aplicativo na inicialização.
- Torne o assembly de inicialização de hospedagem detectável e carregável.

A abordagem recomendada para gerar o arquivo de dependências adicionais é:

1. Executar o `dotnet publish` no arquivo de manifesto do repositório de tempo de execução mencionado na seção anterior.
2. Remover a referência do manifesto das bibliotecas e a seção `runtime` resultantes do arquivo `*deps.json`.

No projeto de exemplo, a propriedade `store.manifest/1.0.0` é removida das seções `targets` e `libraries`:

```
{
  "runtimeTarget": {
    "name": ".NETCoreApp,Version=v2.1",
    "signature": "4ea77c7b75ad1895ae1ea65e6ba2399010514f99"
  },
  "compilationOptions": {},
  "targets": {
    ".NETCoreApp,Version=v2.1": {
      "store.manifest/1.0.0": {
        "dependencies": {
          "StartupDiagnostics": "1.0.0"
        },
        "runtime": {
          "store.manifest.dll": {}
        }
      },
      "StartupDiagnostics/1.0.0": {
        "runtime": {
          "lib/netcoreapp2.1/StartupDiagnostics.dll": {
            "assemblyVersion": "1.0.0.0",
            "fileVersion": "1.0.0.0"
          }
        }
      }
    },
    "libraries": {
      "store.manifest/1.0.0": {
        "type": "project",
        "serviceable": false,
        "sha512": ""
      },
      "StartupDiagnostics/1.0.0": {
        "type": "package",
        "serviceable": true,
        "sha512": "sha512-
oiQr60vBQW7+nBTmgKLSldj06WNLRTdhOZpAdEbCuapoZ+M2DJH2uQbRLvFT8EGAAv4TAKzNtcztpx5Y0gBXQQ==",
        "path": "startupdiagnostics/1.0.0",
        "hashPath": "startupdiagnostics.1.0.0.nupkg.sha512"
      }
    }
  }
}
```

Coloque o arquivo `*.deps.json` no seguinte local:

```
{ADDITIONAL DEPENDENCIES PATH}/shared/{SHARED FRAMEWORK NAME}/{SHARED FRAMEWORK VERSION}/{ENHANCEMENT ASSEMBLY NAME}.deps.json
```

- `{ADDITIONAL DEPENDENCIES PATH}` – Local adicionado à variável de ambiente `DOTNET_ADDITIONAL_DEPS`.
- `{SHARED FRAMEWORK NAME}` – Estrutura compartilhada necessária para esse arquivo de dependências adicionais.
- `{SHARED FRAMEWORK VERSION}` – Versão mínima de estrutura compartilhada.
- `{ENHANCEMENT ASSEMBLY NAME}` – Nome do assembly do aprimoramento.

No aplicativo de exemplo (projeto `RuntimeStore`), o arquivo de dependências adicionais é colocado no seguinte local:

```
additionalDeps/shared/Microsoft.AspNetCore.App/2.1.0/StartupDiagnostics.deps.json
```

Para o tempo de execução descobrir o local do repositório de tempo de execução, o local do arquivo de

dependências adicionais é adicionado à variável de ambiente `DOTNET_ADDITIONAL_DEPS`.

No aplicativo de exemplo (projeto *RuntimeStore*), crie o repositório de tempo de execução e gere o arquivo de dependências adicionais usando um script [PowerShell](#).

Para obter exemplos de como definir variáveis de ambiente para vários sistemas operacionais, confira [Usar vários ambientes](#).

## Implantação

Para facilitar a implantação de uma inicialização de hospedagem em um ambiente multicomputador, o aplicativo de exemplo cria uma pasta *implantação* na saída publicada que contém:

- O repositório de tempo de execução de inicialização de hospedagem.
- O arquivo de dependências de inicialização de hospedagem.
- Um script do PowerShell que cria ou modifica `ASPNETCORE_HOSTINGSTARTUPASSEMBLIES`, `DOTNET_SHARED_STORE` e `DOTNET_ADDITIONAL_DEPS` para dar suporte à ativação da inicialização de hospedagem. Execute o script de um prompt de comando do PowerShell administrativo no sistema de implantação.

## Pacote NuGet

Uma melhoria da inicialização de hospedagem pode ser fornecida em pacote do NuGet. O pacote tem um atributo `HostingStartup`. Os tipos de inicialização de hospedagem fornecidos pelo pacote são disponibilizados para o aplicativo usando qualquer uma das seguintes abordagens:

- O arquivo de projeto do aplicativo aprimorado faz uma referência de pacote para a inicialização de hospedagem no arquivo de projeto do aplicativo (uma referência de tempo de compilação). Com a referência de tempo de compilação em vigor, o assembly de inicialização de hospedagem e todas as suas dependências são incorporados ao arquivo de dependência do aplicativo (`*.deps.json`). Essa abordagem se aplica a um pacote de assembly de inicialização de hospedagem publicado para [nuget.org](#).
- O arquivo de dependências da inicialização de hospedagem fica disponível para o aplicativo avançado, conforme descrito na seção [Repositório de tempo de execução](#) (sem uma referência de tempo de compilação).

Para obter mais informações sobre pacotes do NuGet e o repositório de tempo de execução, consulte os tópicos a seguir:

- [Como criar um pacote do NuGet com ferramentas de plataforma cruzada](#)
- [Publicando pacotes](#)
- [Repositório de pacote de tempo de execução](#)

## Pasta Lixeira do projeto

Uma melhoria da inicialização de hospedagem pode ser fornecida por um assembly implantado na *lixeira* no aplicativo aprimorado. Os tipos de inicialização de hospedagem fornecidos pelo assembly são disponibilizados para o aplicativo usando uma das seguintes abordagens:

- O arquivo de projeto do aplicativo aprimorado faz uma referência de assembly para a inicialização de hospedagem (uma referência de tempo de compilação). Com a referência de tempo de compilação em vigor, o assembly de inicialização de hospedagem e todas as suas dependências são incorporados ao arquivo de dependência do aplicativo (`*.deps.json`). Essa abordagem é aplicável quando o cenário de implantação faz chamadas para mover o assembly compilado da biblioteca de inicialização de hospedagem (arquivo DLL) para o projeto consumidor ou para um local acessível pelo projeto consumidor e uma referência de tempo de compilação é feita para o assembly de inicialização de hospedagem.
- O arquivo de dependências da inicialização de hospedagem fica disponível para o aplicativo avançado, conforme descrito na seção [Repositório de tempo de execução](#) (sem uma referência de tempo de compilação).

# Código de exemplo

O [código de exemplo \(como baixar\)](#) demonstra cenários de implementação de inicialização de hospedagem:

- Dois assemblies de inicialização de hospedagem (bibliotecas de classes) definem um par chave-valor de configuração na memória cada:
  - Pacote do NuGet (*HostingStartupPackage*)
  - Biblioteca de classes (*HostingStartupLibrary*)
- Uma inicialização de hospedagem é ativada de um assembly implantado pelo repositório de tempo de execução (*StartupDiagnostics*). O assembly adiciona dois middlewares ao aplicativo na inicialização que fornecem informações de diagnóstico sobre:
  - Serviços registrados
  - Endereço (esquema, host, base do caminho, caminho, cadeia de caracteres de consulta)
  - Conexão (IP remoto, porta remota, IP local, porta local, certificado do cliente)
  - Cabeçalhos de solicitação
  - Variáveis de ambiente

Para executar a amostra:

## Ativação de um pacote do NuGet

1. Compile o pacote *HostingStartupPackage* com o comando [dotnet pack](#).
2. Adicione o nome do assembly do pacote do *HostingStartupPackage* para a variável de ambiente `ASPNETCORE_HOSTINGSTARTUPASSEMBLIES`.
3. Compile e execute o aplicativo. Uma referência de pacote está presente no aplicativo aprimorado (uma referência de tempo de compilação). Um `<PropertyGroup>` no arquivo de projeto do aplicativo especifica a saída do projeto de pacote (`./HostingStartupPackage/bin/Debug`) como uma origem de pacote. Isso permite que o aplicativo use o pacote sem carregar o pacote para [nuget.org](#). Para mais informações, consulte as notas no arquivo de projeto do *HostingStartupApp*.

```
<PropertyGroup>
  <RestoreSources>$(RestoreSources);https://api.nuget.org/v3/index.json;../HostingStartupPackage/bin/Debug</RestoreSources>
</PropertyGroup>
```

4. Observe que os valores de chave de configuração do serviço renderizados pela página de índice correspondem aos valores definidos pelo método `ServiceKeyInjection.Configure` do pacote.

Se você fizer alterações no projeto *HostingStartupPackage* e recompilá-lo, limpe os caches de pacote do NuGet locais para garantir que o *HostingStartupApp* receba o pacote atualizado e não um pacote obsoleto do cache local. Para limpar os caches locais do NuGet, execute o seguinte comando [dotnet nuget locals](#):

```
dotnet nuget locals all --clear
```

## Ativação de uma biblioteca de classes

1. Compile a biblioteca de classes *HostingStartupLibrary* com o comando [dotnet build](#).
2. Adicione nome do assembly da biblioteca de classes do *HostingStartupLibrary* à variável de ambiente `ASPNETCORE_HOSTINGSTARTUPASSEMBLIES`.
3. A pasta *lixeira* implanta o assembly da biblioteca de classes para o aplicativo ao copiar o arquivo

*HostingStartupLibrary.dll* da saída compilada da biblioteca de classes para a pasta *lixeira/Depurar* do aplicativo.

4. Compile e execute o aplicativo. Um `<ItemGroup>` do arquivo de projeto do aplicativo referencia o assembly da biblioteca de classes (`.\lixeira\Depurar\netcoreapp2.1\HostingStartupLibrary.dll`) (uma referência de tempo de compilação). Para mais informações, consulte as notas no arquivo de projeto do *HostingStartupApp*.

```
<ItemGroup>
<Reference Include=".\\bin\\Debug\\netcoreapp2.1\\HostingStartupLibrary.dll">
  <HintPath>.\\bin\\Debug\\netcoreapp2.1\\HostingStartupLibrary.dll</HintPath>
  <SpecificVersion>False</SpecificVersion>
</Reference>
</ItemGroup>
```

5. Observe que os valores de chave de configuração do serviço renderizados pela página de índice correspondem aos valores definidos pelo método `ServiceKeyInjection.Configure` da biblioteca de classes.

### Ativação de um assembly implantado pelo repositório de tempo de execução

1. O projeto *StartupDiagnostics* usa o [PowerShell](#) para modificar seu arquivo *StartupDiagnostics.deps.json*. O PowerShell é instalado por padrão em um sistema operacional Windows começando no Windows 7 SP1 e no Windows Server 2008 R2 SP1. Para obter o PowerShell em outras plataformas, confira [Instalando o Windows PowerShell](#).
2. Compilar o projeto *StartupDiagnostics*. Depois que o projeto é compilado, um destino de compilação no arquivo de projeto automaticamente:
  - Dispara o script do PowerShell para modificar o arquivo *StartupDiagnostics.deps.json*.
  - Move o arquivo *StartupDiagnostics.deps.json* para a pasta *additionalDeps* do perfil do usuário.
3. Execute o comando `dotnet store` em um prompt de comando no diretório de inicialização de hospedagem para armazenar o assembly e suas dependências no repositório de tempo de execução do perfil do usuário:

```
dotnet store --manifest StartupDiagnostics.csproj --runtime <RID>
```

Para Windows, o comando usa o `win7-x64` [RID \(identificador de tempo de execução\)](#). Ao fornecer a inicialização de hospedagem para um tempo de execução diferente, substitua o RID correto.

4. Defina as variáveis de ambiente:
  - Adicione o nome do assembly de *StartupDiagnostics* para a variável de ambiente `ASPNETCORE_HOSTINGSTARTUPASSEMBLIES`.
  - No Windows, defina a variável de ambiente `DOTNET_ADDITIONAL_DEPS` como `%UserProfile%\\.dotnet\\x64\\additionalDeps\\StartupDiagnostics\\`. No macOS/Linux, defina a variável de ambiente `DOTNET_ADDITIONAL_DEPS` como `/Users/<USER>/\\.dotnet/x64/additionalDeps/StartupDiagnostics/`, em que `<USER>` é o perfil do usuário que contém a inicialização de hospedagem.
5. Execute o aplicativo de exemplo.
6. Solicite o ponto de extremidade `/services` para ver os serviços registrados do aplicativo. Solicite o ponto de extremidade `/diag` para ver as informações de diagnóstico.

# Metapacote Microsoft.AspNetCore.App para ASP.NET Core 2.1

08/10/2018 • 7 minutes to read • [Edit Online](#)

Este recurso exige o ASP.NET Core 2.1 e posterior direcionado ao .NET Core 2.1 e posterior.

O [metapacote Microsoft.AspNetCore.App](#) para ASP.NET Core:

- Não tem dependências de terceiros, com exceção de [Json.NET](#), [Remotion.Linq](#) e [IX-Async](#). Essas dependências de terceiros são consideradas necessárias para garantir o funcionamento dos principais recursos das estruturas.
- Inclui todos os pacotes com suporte pela equipe do ASP.NET Core, exceto aqueles que contêm dependências de terceiros (que não sejam aqueles mencionados anteriormente).
- Inclui todos os pacotes com suporte pela equipe do Entity Framework Core, exceto aqueles que contêm dependências de terceiros (que não sejam aqueles mencionados anteriormente).

Todos os recursos do ASP.NET Core 2.1 e posterior e do Entity Framework Core 2.1 e posterior são incluídos no pacote `Microsoft.AspNetCore.App`. Os modelos de projeto padrão direcionados para ASP.NET Core 2.1 e posterior usam este pacote. Recomendamos que aplicativos voltados para o ASP.NET Core 2.1 e posterior e o Entity Framework Core 2.1 e posterior usem o pacote `Microsoft.AspNetCore.App`.

O número de versão do metapacote `Microsoft.AspNetCore.App` representa a versão do ASP.NET Core e a versão do Entity Framework Core.

O uso do metapacote `Microsoft.AspNetCore.App` fornece restrições de versões que protegem seu aplicativo:

- Se um pacote incluído tem uma dependência (não direta) transitiva em um pacote no `Microsoft.AspNetCore.App`, e os números de versão forem diferentes, o NuGet gera um erro.
- Outros pacotes adicionados ao seu aplicativo não podem alterar a versão dos pacotes incluídos no `Microsoft.AspNetCore.App`.
- A consistência de versão garante uma experiência confiável. `Microsoft.AspNetCore.App` foi projetado para evitar combinações de versão não testado de bits relacionados que estão sendo usados juntos no mesmo aplicativo.

Aplicativos que usam o metapacote `Microsoft.AspNetCore.App` aproveitam automaticamente a estrutura compartilhada do ASP.NET Core. Quando você usa o metapacote `Microsoft.AspNetCore.App`, **nenhum** ativo dos pacotes NuGet do ASP.NET Core referenciados é implantado com o aplicativo, porque a estrutura compartilhada do ASP.NET Core contém esses ativos. Os ativos na

estrutura compartilhada são pré-compilados para melhorar o tempo de inicialização do aplicativo. Para obter mais informações, veja "estrutura compartilhada" em [Empacotamento de distribuição do .NET Core](#).

O seguinte arquivo de projeto referencia o metapacote `Microsoft.AspNetCore.App` do ASP.NET Core e representa um modelo típico do ASP.NET Core 2.1:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp2.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.App" />
  </ItemGroup>

</Project>
```

A marcação anterior representa um modelo típico de ASP.NET Core 2.1 e posterior. Ela não especifica um número de versão para a referência de pacote `Microsoft.AspNetCore.App`. Quando a versão não for especificada, uma versão **implícita** será especificada pelo SDK, ou seja, `Microsoft.NET.Sdk.Web`.

Recomendamos que você conte com a versão implícita especificada pelo SDK, e não defina explicitamente o número de versão na referência de pacote. Se tiver dúvidas sobre essa abordagem, deixe um comentário no GitHub na [Discussão para a versão implícita do Microsoft.AspNetCore.App](#).

A versão implícita é definida como `major.minor.0` para aplicativos portátil. O mecanismo de roll forward estrutura compartilhada executará o aplicativo na versão compatível mais recente entre as estruturas compartilhadas instaladas. Para garantir que a mesma versão seja usada no desenvolvimento, no teste e na produção, certifique-se de que a mesma versão da estrutura compartilhada seja instalada em todos os ambientes. Para aplicativos independentes, o número de versão implícita é definido como `major.minor.patch` da estrutura compartilhada incluída no SDK instalado.

Especificar um número de versão na referência `Microsoft.AspNetCore.App` **não** garante que a versão da estrutura compartilhada será escolhida. Por exemplo, suponha que a versão "2.1.1" foi especificada, mas "2.1.3" está instalada. Nesse caso, o aplicativo usará "2.1.3". Embora não seja recomendado, você pode desabilitar o roll forward (patch e/ou secundária). Para obter mais informações sobre como efetuar roll forward do host dotnet e como configurar seu comportamento, veja [Efetuar roll forward do host dotnet](#).

`<Project Sdk` deve ser definido como `Microsoft.NET.Sdk.Web` para usar o `Microsoft.AspNetCore.App` da versão implícita. Quando `<Project Sdk="Microsoft.NET.Sdk">` (sem o `.Web` à direita) é usado:

- O aviso a seguir é gerado:

*Aviso NU1604: a dependência de projeto Microsoft.AspNetCore.App não tem um limite inferior inclusivo. Inclua um limite inferior na versão de dependência para garantir resultados consistentes de restauração.*

- Esse é um problema conhecido com o SDK do .NET Core 2.1 e será

corrigido no SDK do .NET Core 2.2.

## Atualizar o ASP.NET Core

O `Microsoft.AspNetCore.App` metapacote não é um pacote tradicional atualizado do NuGet. Semelhante ao `Microsoft.NETCore.App`, `Microsoft.AspNetCore.App` representa um tempo de execução compartilhado, que tem semântica de controle de versão especial tratada fora do NuGet. Para obter mais informações, veja [Pacotes, metapacotes e estruturas](#).

Para atualizar o ASP.NET Core:

- Em computadores de desenvolvimento e servidores de compilação: baixe e instale o [SDK do .NET Core](#).
- Nos servidores de implantação: baixe e instale o [tempo de execução do .NET Core](#).

Os aplicativos efetuarão roll forward para a versão mais recente instalada na reinicialização do aplicativo. Não é necessário atualizar o número de versão `Microsoft.AspNetCore.App` no arquivo de projeto. Para obter mais informações, consulte [Roll forward de aplicativos dependentes de estrutura](#).

Se seu aplicativo tiver usado `Microsoft.AspNetCore.All`, veja [Migração do Microsoft.AspNetCore.All para Microsoft.AspNetCore.App](#).

# Metapacote Microsoft.AspNetCore.All para ASP.NET Core 2.0

01/11/2018 • 7 minutes to read • [Edit Online](#)

## NOTE

Recomendamos que os aplicativos direcionados ao ASP.NET Core 2.1 e posterior usem o [metapacote Microsoft.AspNetCore.App](#) em vez desse pacote. Veja [Migração do Microsoft.AspNetCore.All para Microsoft.AspNetCore.App](#) neste artigo.

Este recurso exige o ASP.NET Core 2.x direcionado ao .NET Core 2.x.

O [Metapacote do Microsoft.AspNetCore.All](#) para ASP.NET Core inclui:

- Todos os pacotes com suporte da equipe do ASP.NET Core.
- Todos os pacotes com suporte pelo Entity Framework Core.
- Dependências internas e de terceiros usadas por ASP.NET Core e pelo Entity Framework Core.

Todos os recursos do ASP.NET Core 2.x e do Entity Framework Core 2.x são incluídos no pacote [Microsoft.AspNetCore.All](#). Os modelos de projeto padrão direcionados para ASP.NET Core 2.0 usam este pacote.

O número de versão do metapacote [Microsoft.AspNetCore.All](#) representa a versão do ASP.NET Core e a versão do Entity Framework Core.

Aplicativos que usam o metapacote [Microsoft.AspNetCore.All](#) aproveitam automaticamente o novo [Repositório de Tempo de Execução do .NET Core](#). O Repositório de Tempo de Execução contém todos os ativos de tempo de execução necessários para executar aplicativos ASP.NET Core 2.x. Quando você usa o metapacote [Microsoft.AspNetCore.All](#), **nenhum** ativo dos pacotes NuGet do ASP.NET Core referenciados é implantado com o aplicativo —, porque o Repositório de Tempo de Execução do .NET Core contém esse ativos. Os ativos no Repositório de Tempo de Execução são pré-compilados para melhorar o tempo de inicialização do aplicativo.

Use o processo de filtragem de pacote para remover os pacotes não utilizados. Pacotes filtrados são excluídos na saída do aplicativo publicado.

O seguinte arquivo `.csproj` referencia os metapacotes [Microsoft.AspNetCore.All](#) para o ASP.NET Core:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

    <PropertyGroup>
        <TargetFramework>netcoreapp2.0</TargetFramework>
    </PropertyGroup>

    <ItemGroup>
        <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.9" />
    </ItemGroup>

</Project>
```

## Controle de versão implícita

No ASP.NET Core 2.1 ou posterior, você pode especificar a referência de pacote `Microsoft.AspNetCore.All` sem uma versão. Quando a versão não for especificada, uma versão implícita será especificada pelo SDK (`Microsoft.NET.Sdk.Web`). Recomendamos que você conte com a versão implícita especificada pelo SDK, e não defina explicitamente o número de versão na referência de pacote. Caso tenha dúvidas sobre essa abordagem, deixe um comentário no GitHub na [Discussion for the Microsoft.AspNetCore.App implicit version](#) (Discussão sobre a versão implícita do `Microsoft.AspNetCore.App`).

A versão implícita é definida como `major.minor.0` para aplicativos portátil. O mecanismo de roll forward da estrutura compartilhada executará o aplicativo na versão compatível mais recente entre as estruturas compartilhadas instaladas. Para garantir que a mesma versão seja usada no desenvolvimento, no teste e na produção, certifique-se de que a mesma versão da estrutura compartilhada seja instalada em todos os ambientes. Para aplicativos autossuficientes, o número de versão implícita é definido como o `major.minor.patch` da estrutura compartilhada agrupada no SDK instalado.

A especificação de um número de versão na referência de pacote `Microsoft.AspNetCore.All` **não** assegura que a versão da estrutura compartilhada será escolhida. Por exemplo, suponha que a versão "2.1.1" foi especificada, mas "2.1.3" está instalada. Nesse caso, o aplicativo usará "2.1.3". Embora não seja recomendado, você pode desabilitar o roll forward (patch e/ou secundária). Para obter mais informações sobre como efetuar roll forward do host dotnet e como configurar seu comportamento, veja [Efetuar roll forward do host dotnet](#).

O SDK do projeto precisa ser definido como `Microsoft.NET.Sdk.Web` no arquivo de projeto para usar a versão implícita do `Microsoft.AspNetCore.All`. Quando o SDK `Microsoft.NET.Sdk` for especificado (`<Project Sdk="Microsoft.NET.Sdk">` na parte superior do arquivo de projeto), o seguinte aviso será gerado:

*Aviso NU1604: a dependência do projeto Microsoft.AspNetCore.All não contém um limite inferior inclusivo. Inclua um limite inferior na versão de dependência para garantir resultados consistentes de restauração.*

Esse é um problema conhecido com o SDK do .NET Core 2.1 e será corrigido no SDK do .NET Core 2.2.

## Migração do `Microsoft.AspNetCore.All` para `Microsoft.AspNetCore.App`

Os pacotes a seguir estão incluídos no `Microsoft.AspNetCore.All`, mas não no pacote `Microsoft.AspNetCore.App`.

- `Microsoft.AspNetCore.ApplicationInsights.HostingStartup`
- `Microsoft.AspNetCore.AzureAppServices.HostingStartup`
- `Microsoft.AspNetCore.AzureAppServicesIntegration`
- `Microsoft.AspNetCore.DataProtection.AzureKeyVault`
- `Microsoft.AspNetCore.DataProtection.AzureStorage`
- `Microsoft.AspNetCore.Server.Kestrel.Transport.Libuv`
- `Microsoft.AspNetCore.SignalR.Redis`
- `Microsoft.Data.Sqlite`
- `Microsoft.Data.Sqlite.Core`
- `Microsoft.EntityFrameworkCore.Sqlite`
- `Microsoft.EntityFrameworkCore.Sqlite.Core`
- `Microsoft.Extensions.Caching.Redis`
- `Microsoft.Extensions.Configuration.AzureKeyVault`
- `Microsoft.Extensions.Logging.AzureAppServices`

- `Microsoft.VisualStudio.Web.BrowserLink`

Para mover de `Microsoft.AspNetCore.All` para `Microsoft.AspNetCore.App`, se seu aplicativo usa APIs dos pacotes acima ou de pacotes trazidos por eles, adicione referências a eles em seu projeto.

Todas as dependências dos pacotes anteriores que, de outra forma, não são dependências de `Microsoft.AspNetCore.App` não são incluídas implicitamente. Por exemplo:

- `StackExchange.Redis` como uma dependência de `Microsoft.Extensions.Caching.Redis`
- `Microsoft.ApplicationInsights` como uma dependência de `Microsoft.AspNetCore.ApplicationInsights.HostingStartup`

## Atualizar o ASP.NET Core 2.1

É recomendável migrar para o metapacote `Microsoft.AspNetCore.App` para a versão 2.1 e posteriores. Para continuar usando o metapacote `Microsoft.AspNetCore.All` e certificar-se de que a versão de patch mais recente foi implantada:

- Em computadores de desenvolvimento e em servidores de build: instale o [SDK do .NET Core](#) mais recente.
- Nos servidores de implantação: instale o [tempo de execução do .NET Core](#) mais recente. Seu aplicativo efetuará roll forward para a versão instalada mais recente em uma reinicialização do aplicativo.

# Registro em log de alto desempenho com o LoggerMessage no ASP.NET Core

30/10/2018 • 12 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

Os recursos do [LoggerMessage](#) criam delegados armazenáveis em cache que exigem menos alocações de objeto e sobrecarga de computação reduzida em comparação aos [métodos de extensão do agente](#), como `LogInformation`, `LogDebug` e `.LogError`. Para cenários de registro em log de alto desempenho, use o padrão `LoggerMessage`.

`LoggerMessage` fornece as seguintes vantagens de desempenho em relação aos métodos de extensão do Agente:

- Métodos de extensão do agente exigem tipos de valor de conversão boxing, como `int`, em `object`. O padrão `LoggerMessage` evita a conversão boxing usando campos `Action` estáticos e métodos de extensão com parâmetros fortemente tipados.
- Os métodos de extensão do agente precisam analisar o modelo de mensagem (cadeia de caracteres de formato nomeada) sempre que uma mensagem de log é gravada. `LoggerMessage` exige apenas a análise de um modelo uma vez quando a mensagem é definida.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

O aplicativo de exemplo demonstra recursos do `LoggerMessage` com um sistema básico de acompanhamento de aspas. O aplicativo adiciona e exclui aspas usando um banco de dados em memória. Conforme ocorrem essas operações, são geradas mensagens de log usando o padrão `LoggerMessage`.

## LoggerMessage.Define

`Define(LogLevel, EventId, String)` cria um delegado `Action` para registrar uma mensagem em log. Sobreargas de `Define` permitem passar até seis parâmetros de tipo para uma cadeia de caracteres de formato nomeada (modelo).

A cadeia de caracteres fornecida para o método `Define` é um modelo e não uma cadeia de caracteres interpolada. Os espaços reservados são preenchidos na ordem em que os tipos são especificados. Os nomes do espaço reservado no modelo devem ser descritivos e consistentes em todos os modelos. Eles servem como nomes de propriedade em dados de log estruturado. Recomendamos o uso da [formatação Pascal Case](#) para nomes de espaço reservado. Por exemplo, `{Count}`, `{FirstName}`.

Cada mensagem de log é uma `Action` mantida em um campo estático criado por `LoggerMessage.Define`. Por exemplo, o aplicativo de exemplo cria um campo para descrever uma mensagem de log para uma solicitação GET para a página de Índice (*Internal/LoggerExtensions.cs*):

```
private static readonly Action<ILogger, Exception> _indexPageRequested;
```

Para a `Action`, especifique:

- O nível de log.
- Um identificador de evento exclusivo (`EventId`) com o nome do método de extensão estático.
- O modelo de mensagem (cadeia de caracteres de formato nomeada).

Uma solicitação para a página de Índice do aplicativo de exemplo define:

- O nível de log como `Information`.
- A ID do evento como `1` com o nome do método `IndexPageRequested`.
- Modelo de mensagem (cadeia de caracteres de formato nomeada) como uma cadeia de caracteres.

```
_indexPageRequested = LoggerMessage.Define(
    LogLevel.Information,
    new EventId(1, nameof(IndexPageRequested)),
    "GET request for Index page");
```

Repositórios de log estruturado podem usar o nome do evento quando recebem a ID do evento para enriquecer o log. Por exemplo, [Serilog](#) usa o nome do evento.

A `Action` é invocada por meio de um método de extensão fortemente tipado. O método `IndexPageRequested` registra uma mensagem para uma solicitação GET da página de Índice no aplicativo de exemplo:

```
public static void IndexPageRequested(this ILogger logger)
{
    _indexPageRequested(logger, null);
}
```

`IndexPageRequested` é chamado no agente no método `OnGetAsync` em `Pages/Index.cshtml.cs`:

```
public async Task OnGetAsync()
{
    _logger.IndexPageRequested();

    Quotes = await _db.Quotes.AsNoTracking().ToListAsync();
}
```

Inspecione a saída do console do aplicativo:

```
info: LoggerMessageSample.Pages.IndexModel[1]
      => RequestId:0HL90M6E7PHK4:00000001 RequestPath:/ => /Index
      GET request for Index page
```

Para passar parâmetros para uma mensagem de log, defina até seis tipos ao criar o campo estático. O aplicativo de exemplo registra uma cadeia de caracteres em log ao adicionar aspas definindo um tipo `string` para o campo `Action`:

```
private static readonly Action<ILogger, string, Exception> _quoteAdded;
```

O modelo de mensagem de log do delegado recebe seus valores de espaço reservado dos tipos fornecidos. O aplicativo de exemplo define um delegado para adicionar aspas quando o parâmetro de aspas é uma `string`:

```
_quoteAdded = LoggerMessage.Define<string>(
    LogLevel.Information,
    new EventId(2, nameof(QuoteAdded)),
    "Quote added (Quote = '{Quote}')");
```

O método de extensão estático para adicionar aspas, `QuoteAdded`, recebe o valor do argumento de aspas e passa-o para o delegado `Action`:

```
public static void QuoteAdded(this ILogger logger, string quote)
{
    _quoteAdded(logger, quote, null);
}
```

No modelo da página de Índice (*Pages/Index.cshtml.cs*), `QuoteAdded` é chamado para registrar a mensagem em log:

```
public async Task<IActionResult> OnPostAddQuoteAsync()
{
    _db.Quotes.Add(Quote);
    await _db.SaveChangesAsync();

    _logger.QuoteAdded(Quote.Text);

    return RedirectToAction();
}
```

Inspecione a saída do console do aplicativo:

```
info: LoggerMessageSample.Pages.IndexModel[2]
      => RequestId:0HL90M6E7PHK5:0000000A RequestPath:/ => /Index
        Quote added (Quote = 'You can avoid reality, but you cannot avoid the consequences of avoiding reality.
        - Ayn Rand')
```

O aplicativo de exemplo implementa um padrão `try` – `catch` para a exclusão de aspas. Uma mensagem informativa é registrada em log para uma operação de exclusão bem-sucedida. Uma mensagem de erro é registrada em log para uma operação de exclusão quando uma exceção é gerada. A mensagem de log para a operação de exclusão sem êxito inclui o rastreamento de pilha da exceção (*Internal/LoggerExtensions.cs*):

```
private static readonly Action<ILogger, string, int, Exception> _quoteDeleted;
private static readonly Action<ILogger, int, Exception> _quoteDeleteFailed;
```

```
_quoteDeleted = LoggerMessage.Define<string, int>(
    LogLevel.Information,
    new EventId(4, nameof(QuoteDeleted)),
    "Quote deleted (Quote = '{Quote}' Id = {Id})");

_quoteDeleteFailed = LoggerMessage.Define<int>(
    LogLevel.Error,
    new EventId(5, nameof(QuoteDeleteFailed)),
    "Quote delete failed (Id = {Id})");
```

Observe como a exceção é passada para o delegado em `QuoteDeleteFailed`:

```
public static void QuoteDeleted(this ILogger logger, string quote, int id)
{
    _quoteDeleted(logger, quote, id, null);
}

public static void QuoteDeleteFailed(this ILogger logger, int id, Exception ex)
{
    _quoteDeleteFailed(logger, id, ex);
}
```

No modelo da página de Índice, uma exclusão de aspas bem-sucedida chama o método `QuoteDeleted` no agente.

Quando as aspas não são encontradas para exclusão, uma `ArgumentNullException` é gerada. A exceção é interceptada pela instrução `try - catch` e registrada em log com uma chamada ao método `QuoteDeleteFailed` no agente no bloco `catch` (`Pages/Index.cshtml.cs`):

```
public async Task<IActionResult> OnPostDeleteQuoteAsync(int id)
{
    var quote = await _db.Quotes.FindAsync(id);

    // DO NOT use this approach in production code!
    // You should check quote to see if it's null before removing
    // it and saving changes to the database. A try-catch is used
    // here for demonstration purposes of LoggerMessage features.
    try
    {
        _db.Quotes.Remove(quote);
        await _db.SaveChangesAsync();

        _logger.QuoteDeleted(quote.Text, id);
    }
    catch (ArgumentNullException ex)
    {
        _logger.QuoteDeleteFailed(id, ex);
    }

    return RedirectToPage();
}
```

Quando as aspas forem excluídas com êxito, inspecione a saída do console do aplicativo:

```
info: LoggerMessageSample.Pages.IndexModel[4]
=> RequestId:0HL90M6E7PHK5:00000016 RequestPath:/ => /Index
    Quote deleted (Quote = 'You can avoid reality, but you cannot avoid the consequences of avoiding
    reality. - Ayn Rand' Id = 1)
```

Quando a exclusão de aspas falha, inspecione a saída do console do aplicativo. Observe que a exceção é incluída na mensagem de log:

```
fail: LoggerMessageSample.Pages.IndexModel[5]
=> RequestId:0HL90M6E7PHK5:00000010 RequestPath:/ => /Index
    Quote delete failed (Id = 999)
System.ArgumentNullException: Value cannot be null.
Parameter name: entity
    at Microsoft.EntityFrameworkCore.Utilities.Check.NotNull[T](T value, String parameterName)
    at Microsoft.EntityFrameworkCore.DbContext.Remove[TEntity](TEntity entity)
    at Microsoft.EntityFrameworkCore.Internal.InternalDbSet`1.Remove(TEntity entity)
    at LoggerMessageSample.Pages.IndexModel.<OnPostDeleteQuoteAsync>d__14.MoveNext() in
        <PATH>\sample\Pages\Index.cshtml.cs:line 87
```

## LoggerMessage.DefineScope

`DefineScope(String)` cria um delegado `Func` para definir um [escopo de log](#). Sobrecargas de `DefineScope` permitem passar até três parâmetros de tipo para uma cadeia de caracteres de formato nomeada (modelo).

Como é o caso com o método `Define`, a cadeia de caracteres fornecida ao método `DefineScope` é um modelo e não uma cadeia de caracteres interpolada. Os espaços reservados são preenchidos na ordem em que os tipos são especificados. Os nomes do espaço reservado no modelo devem ser descritivos e consistentes em todos os modelos. Eles servem como nomes de propriedade em dados de log estruturado. Recomendamos o uso da [formatação Pascal Case](#) para nomes de espaço reservado. Por exemplo, `{Count}`, `{FirstName}`.

Defina um [escopo de log](#) a ser aplicado a uma série de mensagens de log usando o método `DefineScope(String)`.

O aplicativo de exemplo tem um botão **Limpar Tudo** para excluir todas as aspas no banco de dados. As aspas são excluídas com a remoção das aspas individualmente, uma por vez. Sempre que aspas são excluídas, o método `QuoteDeleted` é chamado no agente. Um escopo de log é adicionado a essas mensagens de log.

Habilite `IncludeScopes` na seção de agente do console de `appSettings.json`:

```
{  
    "Logging": {  
        "Console": {  
            "IncludeScopes": true  
        },  
        "LogLevel": {  
            "Default": "Warning"  
        }  
    },  
    "AllowedHosts": "*"  
}
```

Para criar um escopo de log, adicione um campo para conter um delegado `Func` para o escopo. O aplicativo de exemplo cria um campo chamado `_allQuotesDeletedScope` (`Internal/LoggerExtensions.cs`):

```
private static Func<ILogger, int, IDisposable> _allQuotesDeletedScope;
```

Use `DefineScope` para criar o delegado. Até três tipos podem ser especificados para uso como argumentos de modelo quando o delegado é invocado. O aplicativo de exemplo usa um modelo de mensagem que inclui o número de aspas excluídas (um tipo `int`):

```
_allQuotesDeletedScope = LoggerMessage.DefineScope<int>("All quotes deleted (Count = {Count})");
```

Forneça um método de extensão estático para a mensagem de log. Inclua os parâmetros de tipo para propriedades nomeadas exibidos no modelo de mensagem. O aplicativo de exemplo usa uma `count` de aspas a ser excluída e retorna `_allQuotesDeletedScope`:

```
public static IDisposable AllQuotesDeletedScope(this ILogger logger, int count)  
{  
    return _allQuotesDeletedScope(logger, count);  
}
```

O escopo encapsula as chamadas de extensão de log em um bloco `using`:

```
public async Task<IActionResult> OnPostDeleteAllQuotesAsync()
{
    var quoteCount = await _db.Quotes.CountAsync();

    using (_logger.AllQuotesDeletedScope(quoteCount))
    {
        foreach (Quote quote in _db.Quotes)
        {
            _db.Quotes.Remove(quote);

            _logger.QuoteDeleted(quote.Text, quote.Id);
        }
        await _db.SaveChangesAsync();
    }

    return RedirectToPage();
}
```

Inspecione as mensagens de log na saída do console do aplicativo. O seguinte resultado mostra três aspas excluídas com a mensagem de escopo de log incluída:

```
info: LoggerMessageSample.Pages.IndexModel[4]
=> RequestId:0HL90M6E7PHK5:0000002E RequestPath:/ => /Index => All quotes deleted (Count = 3)
  Quote deleted (Quote = 'Quote 1' Id = 2)
info: LoggerMessageSample.Pages.IndexModel[4]
=> RequestId:0HL90M6E7PHK5:0000002E RequestPath:/ => /Index => All quotes deleted (Count = 3)
  Quote deleted (Quote = 'Quote 2' Id = 3)
info: LoggerMessageSample.Pages.IndexModel[4]
=> RequestId:0HL90M6E7PHK5:0000002E RequestPath:/ => /Index => All quotes deleted (Count = 3)
  Quote deleted (Quote = 'Quote 3' Id = 4)
```

## Recursos adicionais

- [Registro em log](#)

# Desenvolver aplicativos ASP.NET Core usando um observador de arquivo

16/01/2019 • 6 minutes to read • [Edit Online](#)

Por [Rick Anderson](#) e [Victor Hurdugaci](#)

`dotnet watch` é uma ferramenta que executa um comando do [CLI do .NET Core](#) quando os arquivos de origem são alterados. Por exemplo, uma alteração de arquivo pode disparar uma compilação, execução de teste ou uma implantação.

Este tutorial usa um aplicativo de API Web existente com dois pontos de extremidade: um que retorna uma soma e outro que retorna um produto. O método de produto tem um bug, que é corrigido neste tutorial.

Baixe o [aplicativo de exemplo](#). Ele é composto por dois projetos: *WebApp* (uma API Web do ASP.NET Core) e *WebAppTests* (testes de unidade para a API Web).

Em um shell de comando, navegue até a pasta *WebApp*. Execute o seguinte comando:

```
dotnet run
```

O resultado do console mostra mensagens semelhantes à seguinte (indicando que o aplicativo está em execução e aguarda solicitações):

```
$ dotnet run
Hosting environment: Development
Content root path: C:/Docs/aspnetcore/tutorials/dotnet-watch/sample/WebApp
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```

Em um navegador da Web, navegue até `http://localhost:<port number>/api/math/sum?a=4&b=5`. Você deve ver o resultado de [9](#).

Navegue para o API do produto (`http://localhost:<port number>/api/math/product?a=4&b=5`). Ele retorna [9](#), não [20](#), conforme o esperado. Esse problema é corrigido mais adiante no tutorial.

## Adicionar `dotnet watch` a um projeto

A ferramenta de observador de arquivo `dotnet watch` está incluída com a versão 2.1.300 do SDK do .NET Core. As etapas a seguir são necessárias ao usar uma versão anterior do SDK do .NET Core.

1. Adicionar uma referência ao pacote de `Microsoft.DotNet.Watcher.Tools` para o arquivo `.csproj`:

```
<ItemGroup>
  <DotNetCliToolReference Include="Microsoft.DotNet.Watcher.Tools" Version="2.0.0" />
</ItemGroup>
```

2. Instale o pacote `Microsoft.DotNet.Watcher.Tools` executando o seguinte comando:

```
dotnet restore
```

## Executar os comandos da CLI do .NET Core usando `dotnet watch`

Qualquer [comando da CLI do .NET Core](#) pode ser executado com `dotnet watch`. Por exemplo:

COMANDO	COMANDO COM INSPEÇÃO
<code>dotnet run</code>	<code>dotnet watch run</code>
<code>dotnet run -f netcoreapp2.0</code>	<code>dotnet watch run -f netcoreapp2.0</code>
<code>dotnet run -f netcoreapp2.0 -- --arg1</code>	<code>dotnet watch run -f netcoreapp2.0 -- --arg1</code>
<code>dotnet test</code>	<code>dotnet watch test</code>

Executar `dotnet watch run` na pasta *WebApp*. O resultado do console indica que `watch` foi iniciado.

## Fazer alterações com `dotnet watch`

Verifique se `dotnet watch` está em execução.

Corrija o bug no método `Product` do *MathController.cs* para que ele retorne o produto e não a soma:

```
public static int Product(int a, int b)
{
    return a * b;
}
```

Salve o arquivo. O resultado do console indica que o `dotnet watch` detectou uma alteração de arquivo e reiniciou o aplicativo.

Verifique se <http://localhost:<port number>/api/math/product?a=4&b=5> retorna o resultado correto.

## Executar testes usando o `dotnet watch`

1. Altere o método `Product` de *MathController.cs* de volta para retornar a soma. Salve o arquivo.
2. Em um shell de comando, navegue até a pasta *WebAppTests*.
3. Execute `dotnet restore`.
4. Execute `dotnet watch test`. Seu resultado indica que um teste falhou e que o observador está aguardando as alterações de arquivo:

```
Total tests: 2. Passed: 1. Failed: 1. Skipped: 0.
Test Run Failed.
```

5. Corrija o código do método `Product` para que ele retorne o produto. Salve o arquivo.

`dotnet watch` detecta a alteração de arquivo e executa os testes novamente. O resultado do console indica a aprovação nos testes.

## Personalizar lista de arquivos a observar

Por padrão, o `dotnet-watch` controla todos os arquivos que correspondem aos seguintes padrões glob:

- `**/*.cs`
- `*.csproj`
- `**/*.resx`

Mais itens podem ser adicionados à lista de inspeção editando o arquivo `.csproj`. Os itens podem ser especificados individualmente ou usando padrões glob.

```
<ItemGroup>
    <!-- extends watching group to include *.js files -->
    <Watch Include="**\*.js" Exclude="node_modules\**\*;**\*.js.map;obj\**\*;bin\**\*" />
</ItemGroup>
```

## Recusa de arquivos a serem observados

`dotnet-watch` pode ser configurado para ignorar as configurações padrão. Para ignorar arquivos específicos, adicione o atributo `Watch="false"` à definição de um item no arquivo `.csproj`:

```
<ItemGroup>
    <!-- exclude Generated.cs from dotnet-watch -->
    <Compile Include="Generated.cs" Watch="false" />

    <!-- exclude Strings.resx from dotnet-watch -->
    <EmbeddedResource Include="Strings.resx" Watch="false" />

    <!-- exclude changes in this referenced project -->
    <ProjectReference Include="..\ClassLibrary1\ClassLibrary1.csproj" Watch="false" />
</ItemGroup>
```

## Projetos de inspeção personalizados

`dotnet-watch` não é restrito a projetos C#. Projetos de inspeção personalizados podem ser criados para lidar com cenários diferentes. Considere o layout de projeto a seguir:

- **test/**
  - `UnitTests/UnitTests.csproj`
  - `IntegrationTests/IntegrationTests.csproj`

Se a meta é observar ambos os projetos, crie um arquivo de projeto personalizado configurado para observar os dois projetos:

```
<Project>
    <ItemGroup>
        <TestProjects Include="**\*.csproj" />
        <Watch Include="**\*.cs" />
    </ItemGroup>

    <Target Name="Test">
        <MSBuild Targets="VSTest" Projects="@({TestProjects})" />
    </Target>

    <Import Project="$(MSBuildExtensionsPath)\Microsoft.Common.targets" />
</Project>
```

Para iniciar a observação de arquivo em ambos os projetos, mude para a pasta de `teste`. Execute o seguinte comando:

```
dotnet watch msbuild /t:Test
```

O VSTest é executado quando há qualquer mudança de arquivo no projeto de teste.

## dotnet-watch no GitHub

dotnet-watch faz parte do [repositório aspnet/AspNetCore](#) do GitHub.

# Ativação de middleware baseada em alocador no ASP.NET Core

30/10/2018 • 3 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

[IMiddlewareFactory](#)/[IMiddleware](#) é um ponto de extensibilidade para a ativação de [middleware](#).

Os métodos de extensão `UseMiddleware` verificam se o tipo registrado de um middleware implementa `IMiddleware`. Se isso acontecer, a instância `IMiddlewareFactory` registrada no contêiner será usada para resolver a implementação `IMiddleware` em vez de usar a lógica de ativação de middleware baseada em convenção. O middleware é registrado como um serviço com escopo ou transitório no contêiner de serviço do aplicativo.

Benefícios:

- Ativação por solicitação (injeção de serviços com escopo)
- Tipagem forte de middleware

`IMiddleware` é ativado por solicitação, de modo que os serviços com escopo possam ser injetados no construtor do middleware.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

O aplicativo de exemplo demonstra o middleware ativado por:

- Convenção. Para obter mais informações sobre a ativação de middleware convencional, consulte o tópico [Middleware](#).
- Uma implementação de `IMiddleware`. A [classe MiddlewareFactory](#) padrão ativa o middleware.

As implementações de middleware funcionam de forma idêntica e registram o valor fornecido por um parâmetro de cadeia de caracteres de consulta (`key`). Os middlewares usam um contexto de banco de dados injetado (um serviço com escopo) para registrar o valor de cadeia de caracteres de consulta em um banco de dados em memória.

## IMiddleware

`IMiddleware` define o middleware para o pipeline de solicitação do aplicativo. O método `InvokeAsync(HttpContext, RequestDelegate)` manipula as solicitações e retorna uma `Task` que representa a execução do middleware.

Middleware ativado por convenção:

```

public class ConventionalMiddleware
{
    private readonly RequestDelegate _next;

    public ConventionalMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public async Task InvokeAsync(HttpContext context, AppDbContext db)
    {
        var keyValue = context.Request.Query["key"];

        if (!string.IsNullOrWhiteSpace(keyValue))
        {
            db.Add(new Request()
            {
                DT = DateTime.UtcNow,
                MiddlewareActivation = "ConventionalMiddleware",
                Value = keyValue
            });
        }

        await db.SaveChangesAsync();
    }

    await _next(context);
}

```

Middleware ativado por `MiddlewareFactory` :

```

public class FactoryActivatedMiddleware : IMiddleware
{
    private readonly AppDbContext _db;

    public FactoryActivatedMiddleware(AppDbContext db)
    {
        _db = db;
    }

    public async Task InvokeAsync(HttpContext context, RequestDelegate next)
    {
        var keyValue = context.Request.Query["key"];

        if (!string.IsNullOrWhiteSpace(keyValue))
        {
            _db.Add(new Request()
            {
                DT = DateTime.UtcNow,
                MiddlewareActivation = "FactoryActivatedMiddleware",
                Value = keyValue
            });
        }

        await _db.SaveChangesAsync();
    }

    await next(context);
}

```

As extensões são criadas para os middlewares:

```

public static class MiddlewareExtensions
{
    public static IApplicationBuilder UseConventionalMiddleware(
        this IApplicationBuilder builder)
    {
        return builder.UseMiddleware<ConventionalMiddleware>();
    }

    public static IApplicationBuilder UseFactoryActivatedMiddleware(
        this IApplicationBuilder builder)
    {
        return builder.UseMiddleware<FactoryActivatedMiddleware>();
    }
}

```

Não é possível passar objetos para o middleware ativado por alocador com `UseMiddleware`:

```

public static IApplicationBuilder UseFactoryActivatedMiddleware(
    this IApplicationBuilder builder, bool option)
{
    // Passing 'option' as an argument throws a NotSupportedException at runtime.
    return builder.UseMiddleware<FactoryActivatedMiddleware>(option);
}

```

O middleware ativado por alocador é adicionado ao contêiner interno em *Startup.cs*:

```

public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddDbContext<AppDbContext>(options =>
        options.UseInMemoryDatabase("InMemoryDb"));

    services.AddTransient<FactoryActivatedMiddleware>();

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
}

```

Ambos os middlewares estão registrados no pipeline de processamento de solicitação em `Configure`:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }

    app.UseConventionalMiddleware();
    app.UseFactoryActivatedMiddleware();

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseCookiePolicy();
    app.UseMvc();
}
```

## IMiddlewareFactory

[IMiddlewareFactory](#) fornece métodos para a criação do middleware. A implementação de alocador do middleware é registrada no contêiner como um serviço com escopo.

A implementação `IMiddlewareFactory` padrão, [MiddlewareFactory](#), foi encontrada no pacote [Microsoft.AspNetCore.Http](#).

## Recursos adicionais

- [Middleware do ASP.NET Core](#)
- [Ativação de middleware com um contêiner de terceiros no ASP.NET Core](#)

# Ativação de middleware com um contêiner de terceiros no ASP.NET Core

30/10/2018 • 4 minutes to read • [Edit Online](#)

Por [Luke Latham](#)

Este artigo demonstra como usar o [IMiddlewareFactory](#) e o [IMiddleware](#) como um ponto de extensibilidade para a ativação do [middleware](#) com um contêiner de terceiros. Para obter informações introdutórias sobre o [IMiddlewareFactory](#) e o [IMiddleware](#), confira o tópico [Ativação de middleware baseada em alocador no ASP.NET Core](#).

## [Exibir ou baixar código de exemplo \(como baixar\)](#)

O aplicativo de exemplo demonstra a ativação do middleware por uma implementação de [IMiddlewareFactory](#), [SimpleInjectorMiddlewareFactory](#). O exemplo usa o contêiner de DI (injeção de dependência) [Simple Injector](#).

A implementação do middleware do exemplo registra o valor fornecido por um parâmetro de cadeia de consulta ([key](#)). O middleware usa um contexto de banco de dados injetado (um serviço com escopo) para registrar o valor da cadeia de consulta em um banco de dados em memória.

### **NOTE**

O aplicativo de exemplo usa o [Simple Injector](#) simplesmente para fins de demonstração. O uso do Simple Injector não é um endosso. As abordagens de ativação do middleware descritas na documentação do Simple Injector e nos problemas do GitHub são recomendadas pelos mantenedores do Simple Injector. Para obter mais informações, confira a [Documentação do Simple Injector](#) e o [repositório do GitHub do Simple Injector](#).

## [IMiddlewareFactory](#)

[IMiddlewareFactory](#) fornece métodos para a criação do middleware.

No aplicativo de amostra, um alocador de middleware é implementado para criar uma instância de [SimpleInjectorActivatedMiddleware](#). O alocador de middleware usa o contêiner do Simple Injector para resolver o middleware:

```

public class SimpleInjectorMiddlewareFactory : IMiddlewareFactory
{
    private readonly Container _container;

    public SimpleInjectorMiddlewareFactory(Container container)
    {
        _container = container;
    }

    public IMiddleware Create(Type middlewareType)
    {
        return _container.GetInstance(middlewareType) as IMiddleware;
    }

    public void Release(IMiddleware middleware)
    {
        // The container is responsible for releasing resources.
    }
}

```

## IMiddleware

[IMiddleware](#) define o middleware para o pipeline de solicitação do aplicativo.

Middleware ativado por uma implementação de [IMiddlewareFactory](#) (*Middleware/SimpleInjectorActivatedMiddleware.cs*):

```

public class SimpleInjectorActivatedMiddleware : IMiddleware
{
    private readonly AppDbContext _db;

    public SimpleInjectorActivatedMiddleware(AppDbContext db)
    {
        _db = db;
    }

    public async Task InvokeAsync(HttpContext context, RequestDelegate next)
    {
        var keyValue = context.Request.Query["key"];

        if (!string.IsNullOrWhiteSpace(keyValue))
        {
            _db.Add(new Request()
            {
                DT = DateTime.UtcNow,
                MiddlewareActivation = "SimpleInjectorActivatedMiddleware",
                Value = keyValue
            });
        }

        await _db.SaveChangesAsync();
    }

    await next(context);
}

```

Uma extensão é criada para o middleware (*Middleware/MiddlewareExtensions.cs*):

```

public static class MiddlewareExtensions
{
    public static IApplicationBuilder UseSimpleInjectorActivatedMiddleware(
        this IApplicationBuilder builder)
    {
        return builder.UseMiddleware<SimpleInjectorActivatedMiddleware>();
    }
}

```

O `Startup.ConfigureServices` precisa executar várias tarefas:

- Configure o contêiner do Simple Injector.
- Registre o alocador e o middleware.
- Disponibilize o contexto do banco de dados do aplicativo por meio do contêiner do Simple Injector para uma página Razor.

```

public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    // Replace the default middleware factory with the
    // SimpleInjectorMiddlewareFactory.
    services.AddTransient<IMiddlewareFactory>(_ =>
    {
        return new SimpleInjectorMiddlewareFactory(_container);
    });

    // Wrap ASP.NET Core requests in a Simple Injector execution
    // context.
    services.UseSimpleInjectorAspNetRequestScoping(_container);

    // Provide the database context from the Simple
    // Injector container whenever it's requested from
    // the default service container.
    services.AddScoped<AppDbContext>(provider =>
        _container.GetInstance<AppDbContext>());

    _container.Options.DefaultScopedLifestyle = new AsyncScopedLifestyle();

    _container.Register<AppDbContext>(() =>
    {
        var optionsBuilder = new DbContextOptionsBuilder<DbContext>();
        optionsBuilder.UseInMemoryDatabase("InMemoryDb");
        return new AppDbContext(optionsBuilder.Options);
    }, Lifestyle.Scoped);

    _container.Register<SimpleInjectorActivatedMiddleware>();

    _container.Verify();
}

```

O middleware é registrado no pipeline de processamento da solicitação em `Startup.Configure`:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }

    app.UseSimpleInjectorActivatedMiddleware();

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseCookiePolicy();
    app.UseMvc();
}
```

## Recursos adicionais

- [Middleware](#)
- [Ativação de middleware de fábrica](#)
- [Repositório do GitHub do Simple Injector](#)
- [Documentação do Simple Injector](#)

# Migrar do ASP.NET Core 2.2 a 3.0 versão prévia 2

07/02/2019 • 4 minutes to read • [Edit Online](#)

Por [Scott Addie](#) e [Rick Anderson](#)

Este artigo explica como atualizar um projeto existente do ASP.NET Core 2.2 para ASP.NET Core 3.0 versão prévia 2.

## Pré-requisitos

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- [Visualização do Visual Studio 2019](#) com o **ASP.NET e desenvolvimento web** carga de trabalho
- [.NET core SDK 3.0 versão prévia](#)

## Atualizar o arquivo de projeto

- Defina as [Framework Moniker \(TFM\) de destino](#) para `netcoreapp3.0` :

```
<TargetFramework>netcoreapp3.0</TargetFramework>
```

- Remova qualquer `<PackageReference>` para o [Microsoft.AspNetCore.All](#) ou [Microsoft metapacote](#).
- Atualizar o `Version` em demais `<PackageReference>` elementos para `Microsoft.AspNetCore.*` pacotes para a visualização atual (por exemplo, 3.0.0-preview-18579-0053).

Se não houver nenhuma versão 3.0 de um pacote, o pacote pode ter sido preferido no 3.0. Muitos deles são parte do `Microsoft.AspNetCore.App` e não devem ser referenciados individualmente. Para obter uma lista preliminar de pacotes não mais produzidas no 3.0, consulte [aspnet/AspNetCore #3756](#).

- Alguns assemblies foram removidos do `Microsoft.AspNetCore.App` entre 2.x e o 3.0. Talvez você precise adicionar `<PackageReference>` itens se você estiver usando as APIs dos pacotes listados no [aspnet/AspNetCore #3755](#)

Por exemplo, `Microsoft.EntityFrameworkCore` e `System.Data.SqlClient` fazem parte não dos `Microsoft.AspNetCore.App`. A lista de assemblies de envio no `Microsoft.AspNetCore.App` ainda não foram finalizados e será alterado antes da RTM 3.0.

- Adicionar [Json.NET suporte](#)

## Suporte de Json.NET

Como parte do trabalho para [melhorar a estrutura compartilhada do ASP.NET Core](#), [Json.NET](#) foi removida da estrutura compartilhada do ASP.NET Core.

Para usar o Json.NET em um projeto do ASP.NET Core 3.0:

- Adicione uma referência de pacote ao `Microsoft.AspNetCore.Mvc.NewtonsoftJson`
- Atualização `ConfigureServices` chamar `AddNewtonsoftJson()` .

```
services.AddMvc()
    .AddNewtonsoftJson();
```

Newtonsoft podem ser definidas com `AddNewtonsoftJson`:

```
services.AddMvc()
    .AddNewtonsoftJson(options =>
        options.SerializerSettings.ContractResolver =
            new CamelCasePropertyNamesContractResolver());
```

## Substitui o HostBuilder WebHostBuilder

Usam os modelos do ASP.NET Core 3.0 [Host genérico](#). Versões anteriores usadas [Host Web](#). O código a seguir mostra o modelo do ASP.NET Core 3.0 gerado `Program` classe:

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateHostBuilder(args).Build().Run();
    }

    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.UseStartup<Startup>();
            });
}
```

O código a seguir mostra o modelo gerado do ASP.NET Core 2.2 `Program` classe:

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>();
}
```

`IWebHostBuilder` permanece no 3.0, e é o tipo do `webBuilder` visto no exemplo de código anterior.

`WebHostBuilder` será preferido em uma versão futura e substituído por `HostBuilder`.

## Movimentação de WebHostBuilder para HostBuilder

A alteração mais significativa do `WebHostBuilder` à `HostBuilder` está em [injeção de dependência \(DI\)](#). Ao usar `HostBuilder`, você apenas pode injetar [IConfiguration](#) e [IHostingEnvironment](#) no construtor da inicialização. O `HostBuilder` DI restrições:

- Habilite o contêiner de injeção de dependência a ser criado apenas uma vez.
- Evita os problemas de tempo de vida do objeto resultante como resolução de várias instâncias de singletons.

## Atualizar o código do SignalR

Se você chamar `AddJsonProtocol`, substitua-a por `AddNewtonsoftJsonProtocol`.

- Os exemplos a seguir mostram o código do servidor antes e depois da alteração:

```
services.AddSignalR(...)  
    .AddJsonProtocol(...) // 2.2
```

```
services.AddSignalR(...)  
    .AddNewtonsoftJsonProtocol(...) // 3.0
```

- Os exemplos a seguir mostram o código do cliente .NET antes e depois da alteração:

```
connection = new HubConnectionBuilder()  
    .WithUrl(...)  
    .AddJsonProtocol(...) // 2.2  
    .Build()
```

```
connection = new HubConnectionBuilder()  
    .WithUrl(...)  
    .AddNewtonsoftJsonProtocol(...) // 3.0  
    .Build()
```

# Migrar do ASP.NET Core 2.1 para 2.2

12/02/2019 • 8 minutes to read • [Edit Online](#)

Por [Scott Addie](#)

Este artigo explica como atualizar um projeto existente do ASP.NET Core 2.1 para ASP.NET Core 2.2.

## Pré-requisitos

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Visual Studio para Mac](#)
- [Visual Studio 2017 versão 15.9 ou posterior](#) com a carga de trabalho **ASP.NET e desenvolvimento para a Web**
- [SDK 2.2 ou posterior do .NET Core](#)

## Atualizar TFM (Moniker da Estrutura de Destino)

Projetos direcionados ao .NET Core devem usar o [TFM](#) de uma versão maior ou igual a 2.2 do .NET Core.

Atualizar o `<TargetFramework>` texto interno do nó com `netcoreapp2.2`:

```
<TargetFramework>netcoreapp2.2</TargetFramework>
```

Projetos direcionados ao .NET Framework podem continuar a usar o TFM de uma versão maior ou igual ao .NET Framework 4.6.1:

```
<TargetFramework>net461</TargetFramework>
```

## Adote o modelo de hospedagem em processo IIS

Para adotar a [modelo de hospedagem em processo para o IIS](#), adicione o `<AspNetCoreHostingModel>` propriedade com um valor de `InProcess` para um `<PropertyGroup>` no arquivo de projeto:

```
<AspNetCoreHostingModel>InProcess</AspNetCoreHostingModel>
```

Não há suporte para o modelo de hospedagem em processo para aplicativos ASP.NET Core direcionados ao .NET Framework.

Para obter mais informações, consulte [Módulo do ASP.NET Core](#).

## Referências do pacote de atualização

Se estiver direcionando para .NET Core, remova o `Version` atributo para a referência do metapacote. Inclusão de um `Version` atributo resulta em aviso a seguir:

```
A PackageReference to 'Microsoft.AspNetCore.App' specified a Version of `2.2.0`. Specifying the version of this package is not recommended. For more information, see https://aka.ms/sdkimplicitrefs
```

Para obter mais informações, consulte [Metapacote Microsoft.AspNetCore.App para ASP.NET Core 2.1 e posterior](#).

A referência do metapacote deve ser semelhante à seguinte `<PackageReference />` nó:

```
<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore.App" />
</ItemGroup>
```

Se o destino do .NET Framework, atualizar cada referência de pacote `Version` atributo 2.2.0 ou posterior. Aqui estão as referências do pacote em um projeto ASP.NET Core 2.2 típico direcionado ao .NET Framework:

```
<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore" Version="2.2.0" />
  <PackageReference Include="Microsoft.AspNetCore.CookiePolicy" Version="2.2.0" />
  <PackageReference Include="Microsoft.AspNetCore.HttpsPolicy" Version="2.2.0" />
  <PackageReference Include="Microsoft.AspNetCore.Mvc" Version="2.2.0" />
  <PackageReference Include="Microsoft.AspNetCore.StaticFiles" Version="2.2.0" />
</ItemGroup>
```

Se fazendo referência a [Microsoft.AspNetCore.Razor.Design](#) do pacote, atualize seu `Version` atributo 2.2.0 ou posterior. Falha ao fazer isso resulta no seguinte erro:

```
Detected package downgrade: Microsoft.AspNetCore.Razor.Design from 2.2.0 to 2.1.2. Reference the package directly from the project to select a different version.
```

## Atualizar a versão do SDK do .NET Core em global.json

Se sua solução depende de um [global.JSON](#) arquivo para direcionar uma versão específica do SDK do .NET Core, atualize seu `version` propriedade para a versão 2.2 instalada em seu computador:

```
{
  "sdk": {
    "version": "2.2.100"
  }
}
```

## Atualizar configurações de inicialização

Se usando o Visual Studio Code, atualize o arquivo de configurações de inicialização do projeto (`.vscode/launch.json`). O `program` caminho deve referenciar o novo TFM:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": ".NET Core Launch (web)",
      "type": "coreclr",
      "request": "launch",
      "preLaunchTask": "build",
      // If you have changed target frameworks, make sure to update the program path.
      "program": "${workspaceFolder}/bin/Debug/netcoreapp2.2/test-app.dll",
      "args": [],
      "cwd": "${workspaceFolder}",
      "stopAtEntry": false,
      "internalConsoleOptions": "openOnSessionStart",
      "launchBrowser": {
        "enabled": true,
        "args": "${auto-detect-url}",
        "windows": {
          "command": "cmd.exe",
          "args": "/C start ${auto-detect-url}"
        },
        "osx": {
          "command": "open"
        },
        "linux": {
          "command": "xdg-open"
        }
      },
      "env": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      },
      "sourceFileMap": {
        "/Views": "${workspaceFolder}/Views"
      }
    },
    {
      "name": ".NET Core Attach",
      "type": "coreclr",
      "request": "attach",
      "processId": "${command:pickProcess}"
    }
  ]
}
```

## Atualizar a configuração do Kestrel

Se o aplicativo chama `UseKestrel` chamando `CreateDefaultBuilder` na `método CreateWebHostBuilder` da `Program` de classe, chamada `ConfigureKestrel` para configurar o servidor Kestrel em vez de `UseKestrel` para evitar está em conflito com o [IIS de modelo de hospedagem em processo](#):

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
  WebHost.CreateDefaultBuilder(args)
    .UseStartup<Startup>()
    .ConfigureKestrel((context, options) =>
    {
      // Set properties and call methods on options
    });
  
```

Se o aplicativo não chama `CreateDefaultBuilder` e cria o host manualmente na `Program` classe, chame `UseKestrel` **antes** chamando `ConfigureKestrel`:

```

public static void Main(string[] args)
{
    var host = new WebHostBuilder()
        .UseContentRoot(Directory.GetCurrentDirectory())
        .UseKestrel()
        .UseIISIntegration()
        .UseStartup<Startup>()
        .ConfigureKestrel((context, options) =>
    {
        // Set properties and call methods on options
    })
    .Build();

    host.Run();
}

```

Para obter mais informações, consulte [Implementação do servidor Web Kestrel no ASP.NET Core](#).

## Versão de compatibilidade de atualização

Atualizar a versão de compatibilidade nas `Startup.ConfigureServices` para `Version_2_2`:

```

services.AddMvc()
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_2);

```

## Atualizar a política de CORS

No ASP.NET Core 2.2, o middleware do CORS responde com uma origem de curinga (`*`) se uma política de permitir qualquer origem e permite que as credenciais. As credenciais não são suportadas quando uma origem de curinga (`*`) for especificado, e os navegadores não permitirão a solicitação CORS. Para obter mais informações, incluindo as opções para corrigir o problema no cliente, consulte a [docs da web MDN](#).

Para corrigir esse problema no servidor, execute uma das seguintes ações:

- Modificar a política CORS para não permitir credenciais. Ou seja, remova a chamada para `AllowCredentials` ao configurar a política.
- Se as credenciais são necessárias para a solicitação CORS seja bem-sucedida, modificar a política para especificar hosts permitidos. Por exemplo, use

```
builder.WithOrigins("https://api.example1.com", "https://example2.com")
```

em vez de usar `AllowAnyOrigin`.

## Atualizar as imagens do Docker

A tabela a seguir mostra a imagem do Docker alterações de marca:

2.1	2.2
<code>microsoft/dotnet:2.1-aspnetcore-runtime</code>	<code>microsoft/dotnet:2.2-aspnetcore-runtime</code>
<code>microsoft/dotnet:2.1-sdk</code>	<code>microsoft/dotnet:2.2-sdk</code>

Alterar o `FROM` linhas no seu *Dockerfile* para usar as novas marcas de imagem na coluna de 2,2 da tabela anterior.

## Criar manualmente no Visual Studio ao usar a hospedagem em processo do IIS

Visual Studio **criação automática na solicitação do navegador** experiência não funciona com o [IIS de modelo de hospedagem em processo](#). Você deve recompilar manualmente o projeto ao usar a hospedagem em processo. Aprimoramentos dessa experiência são planejados para uma versão futura do Visual Studio.

## Atualizar o código de registro em log

Código de configuração de log recomendados não foi alterada do 2.1 2.2, mas alguns padrões de codificação de 1. x ainda trabalhou no 2.1 não funciona no 2.2.

Se seu aplicativo faz a inicialização do provedor de log, filtragem e configuração de carregamento a `Startup` classe, mova esse código para `Program.Main`:

- Inicialização do provedor:

exemplo 1.x:

```
public void Configure(IApplicationBuilder app, ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole();
}
```

exemplo de 2.2:

```
public static void Main(string[] args)
{
    var webHost = new WebHostBuilder()
        // ...
        .ConfigureLogging((hostingContext, logging) =>
    {
        logging.AddConsole();
    })
    // ...
}
```

- Filtrando:

exemplo 1.x:

```
public void Configure(IApplicationBuilder app, ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole(LogLevel.Information);
    // or
    loggerFactory.AddConsole((category, level) =>
        category == "A" || level == LogLevel.Critical);
}
```

exemplo de 2.2:

```

public static void Main(string[] args)
{
    var webHost = new WebHostBuilder()
        // ...
        .ConfigureLogging((hostingContext, logging) =>
    {
        logging.AddConsole()
            .AddFilter<ConsoleLoggerProvider>
                (category: null, level: LogLevel.Information)
            // or
            .AddFilter<ConsoleLoggerProvider>
                ((category, level) => category == "A" ||
                    level == LogLevel.Critical)
    );
})
// ...
}

```

- Carregamento da configuração:

exemplo 1.x:

```

public void Configure(IApplicationBuilder app, ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole(Configuration);
}

```

exemplo de 2.2:

```

public static void Main(string[] args)
{
    var webHost = new WebHostBuilder()
        // ...
        .ConfigureLogging((hostingContext, logging) =>
    {
        logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));
        logging.AddConsole();
    })
    // ...
}

```

Para obter mais informações, consulte [Registro em log no ASP.NET Core](#).

## Recursos adicionais

- [Versão de compatibilidade do ASP.NET Core MVC](#)
- [Metapacote Microsoft.AspNetCore.App para ASP.NET Core 2.1 e posterior](#)
- [Referências de pacote implícitas](#)

# Migrar do ASP.NET Core 2.0 para 2.1

05/12/2018 • 18 minutes to read • [Edit Online](#)

Por [Rick Anderson](#)

Ver [o que há de novo no ASP.NET Core 2.1](#) para uma visão geral dos novos recursos no ASP.NET Core 2.1.

Neste artigo:

- Aborda os conceitos básicos da migração de um aplicativo ASP.NET Core 2.0 para 2.1.
- Fornece uma visão geral das alterações para os modelos de aplicativos web ASP.NET Core.

É uma maneira rápida de obter uma visão geral das alterações no 2.1:

- Crie um aplicativo web do ASP.NET Core 2.0 chamado WebApp1.
- Confirme o WebApp1 em um sistema de controle de origem.
- Excluir WebApp1 e criar um aplicativo web do ASP.NET Core 2.1 chamado WebApp1 no mesmo local.
- Revise as alterações na versão 2.1.

Este artigo fornece uma visão geral sobre a migração para o ASP.NET Core 2.1. Ele não contém uma lista completa de todas as alterações necessárias para migrar para a versão 2.1. Alguns projetos podem exigir mais etapas dependendo das opções selecionadas quando o projeto foi criado e as modificações feitas no projeto.

## Atualizar o arquivo de projeto para usar versões 2.1

Atualize o arquivo de projeto:

- Alterar a estrutura de destino para o .NET Core 2.1 Atualizando o arquivo de projeto para `<TargetFramework>netcoreapp2.1</TargetFramework>`.
- Substitua a referência de pacote para `Microsoft.AspNetCore.All` com uma referência de pacote para `Microsoft.AspNetCore.App`. Talvez você precise adicionar dependências que foram removidas da `Microsoft.AspNetCore.All`. Para obter mais informações, consulte [Metapacote Microsoft.AspNetCore.All para ASP.NET Core 2.0 e Metapacote Microsoft.AspNetCore.App para ASP.NET Core 2.1 e posterior](#).
- Remova o atributo "Version" na referência de pacote para `Microsoft.AspNetCore.App`. Projetos que usam `<Project Sdk="Microsoft.NET.Sdk.Web">` não precisa definir a versão. A versão será indicada pela estrutura de destino e selecionada para corresponder melhor o funcionamento do ASP.NET Core 2.1. (Consulte abaixo para obter mais informações.)
- Para aplicativos destinados ao .NET Framework, atualize cada referência de pacote a 2.1.
- Remover referências a `<DotNetCliToolReference>` elementos para os pacotes a seguir. Essas ferramentas são agrupadas por padrão na CLI do .NET Core e não precisam ser instalado separadamente.
  - `Microsoft.DotNet.Watcher.Tools` (`dotnet watch`)
  - `Entityframeworkcore` (`dotnet ef`)
  - `Microsoft.Extensions.Caching.SqlConfig.Tools` (`dotnet sql-cache`)
  - `Secretmanager` (`dotnet user-secrets`)
- Opcional: você pode remover o `<DotNetCliToolReference>` elemento para `Microsoft.VisualStudio.Web.CodeGeneration.Tools`. Você pode substituir essa ferramenta com uma versão instalada globalmente executando `dotnet tool install -g dotnet-aspnet-codegenerator`.
- Para o 2.1, uma [biblioteca de classes Razor](#) é a solução recomendada para distribuir arquivos do Razor. Se seu aplicativo usa exibições inseridas ou, caso contrário, se baseia na compilação de tempo de execução de arquivos

do Razor, adicione `<CopyRefAssembliesToPublishDirectory>true</CopyRefAssembliesToPublishDirectory>` para um `<PropertyGroup>` no arquivo de projeto.

A marcação a seguir mostra o arquivo de 2.0 projeto gerado pelo modelo:

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>
    <UserSecretsId>aspnet-{Project Name}-{GUID}</UserSecretsId>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.9" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="2.0.3" PrivateAssets="All" />
    <PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Design" Version="2.0.4"
PrivateAssets="All" />
  </ItemGroup>
  <ItemGroup>
    <DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet" Version="2.0.3" />
    <DotNetCliToolReference Include="Microsoft.Extensions.SecretManager.Tools" Version="2.0.2" />
    <DotNetCliToolReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Tools" Version="2.0.4" />
  </ItemGroup>
</Project>
```

A marcação a seguir mostra o arquivo de 2.1 projeto gerado pelo modelo:

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp2.1</TargetFramework>
    <UserSecretsId>aspnet-{Project Name}-{GUID}</UserSecretsId>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.App" />
    <PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Design" Version="2.1.1"
PrivateAssets="All" />
  </ItemGroup>
</Project>
```

## Regras para projetos direcionados para a estrutura compartilhada

ASP.NET Core 2.1 inclui as seguintes estruturas compartilhadas:

- [Microsoft.AspNetCore.App](#)
- [Microsoft.AspNetCore.All](#)

A versão especificada pela referência de pacote é o *mínimo necessário* versão. Por exemplo, um projeto referenciando o 2.1.1 versões desses pacotes não serão executados em um computador com apenas o 2.1.0 tempo de execução instalado.

Problemas conhecidos para projetos direcionados para a estrutura compartilhada:

- O .NET Core 2.1.300 SDK (incluído pela primeira vez no Visual Studio 15.6) defina a versão implícita do `Microsoft.AspNetCore.App` à 2.1.0 que causaram conflitos com o Entity Framework Core 2.1.1. A solução recomendada é atualizar o SDK do .NET Core para 2.1.301 ou posterior. Para obter mais informações, consulte [pacotes que compartilham dependências com a Microsoft não podem fazer referência a versões de patch](#).
- Todos os projetos que devem usar `Microsoft.AspNetCore.All` ou `Microsoft.AspNetCore.App` deve adicionar

uma referência de pacote para o pacote no arquivo de projeto, mesmo se eles contêm uma referência de projeto para outro projeto usando `Microsoft.AspNetCore.All` ou `Microsoft.AspNetCore.App`.

Exemplo:

- `MyApp` tem uma referência de pacote para `Microsoft.AspNetCore.App`.
- `MyApp.Tests` tem uma referência de projeto para `MyApp.csproj`.

Adicionar uma referência de pacote para `Microsoft.AspNetCore.App` para `MyApp.Tests`. Para obter mais informações, consulte [testes de integração é difícil de configurar e podem ser divididas em manutenção da estrutura compartilhada](#).

## Atualizar para as imagens do Docker 2.1

No ASP.NET Core 2.1, as imagens do Docker são migrados para o [repositório do GitHub dotnet/dotnet-docker](#). A tabela a seguir mostra o Docker alterações de imagem e marca:

2.0	2.1
Microsoft / Microsoft/aspnetcore:2.0	microsoft/dotnet:2.1-aspnetcore-runtime
Microsoft/aspnetcore-build: 2.0	Microsoft / dotnet:2.1-sdk

Alterar o `FROM` linhas no seu *Dockerfile* para usar as marcas e os novos nomes de imagem na coluna de 2.1 da tabela anterior. Para obter mais informações, consulte [Migrando do aspnetcore repositórios de docker para dotnet](#).

Alterações para aproveitar as novas expressões baseadas em código são recomendados no ASP.NET Core 2.1

### Alterações para Main

As imagens a seguir mostram as alterações feitas para o modelo gerado *Program.cs* arquivo.

```
namespace WebApp1
{
    public class Program
    {
        public static void Main(string[] args)
        {
            BuildWebHost(args).Run();
        }

        public static IWebHost BuildWebHost(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>()
                .Build();
    }
}
```

A imagem anterior mostra a versão 2.0 com as exclusões em vermelho.

A imagem a seguir mostra o código 2.1. O código em verde substituído na versão 2.0:

```
namespace WebApp1
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateWebHostBuilder(args).Build().Run();
        }

        public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
           WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>();
    }
}
```

O código a seguir mostra a versão 2.1 das *Program.cs*:

```
namespace WebApp1
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateWebHostBuilder(args).Build().Run();
        }

        public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
           WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>();
    }
}
```

O novo `Main` substitui a chamada para `BuildWebHost` com `CreateWebHostBuilder`. `IWebHostBuilder` foi adicionado para dar suporte a um novo [infraestrutura de teste de integração](#).

## Alterações para a inicialização

O código a seguir mostra as alterações ao código de modelo 2.1 gerado. Todas as alterações são adicionadas recentemente o código, exceto que `UseBrowserLink` foi removido:

```

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;

namespace WebApp1
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        public void ConfigureServices(IServiceCollection services)
        {
            services.Configure<CookiePolicyOptions>(options =>
            {
                // This lambda determines whether user consent for non-essential cookies is needed for a given
request.
                options.CheckConsentNeeded = context => true;
                options.MinimumSameSitePolicy = SameSiteMode.None;
            });

            services.AddMvc()
                .SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
        }

        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Error");
                app.UseHsts();
            }

            app.UseHttpsRedirection();
            app.UseStaticFiles();
            app.UseCookiePolicy();
            // If the app uses Session or TempData based on Session:
            // app.UseSession();

            app.UseMvc();
        }
    }
}

```

As alterações de código anteriores são detalhadas em:

- [Suporte de GDPR no ASP.NET Core](#) para `CookiePolicyOptions` e `UseCookiePolicy`.
- [Protocolo de segurança de transporte estrito HTTP \(HSTS\)](#) para `UseHsts`.
- [Exigir HTTPS para](#) `UseHttpsRedirection`.
- [SetCompatibilityVersion](#) para `SetCompatibilityVersion(CompatibilityVersion.Version_2_1)`.

# Alterações no código de autenticação

ASP.NET Core 2.1 oferece [ASP.NET Core Identity](#) como um [biblioteca de classes Razor \(RCL\)](#).

O padrão 2.1 interface de identidade do usuário atualmente não fornece novos recursos significativos em relação à versão 2.0. Substituindo a identidade com o pacote RCL é opcional. As vantagens para substituir o modelo gerado identidade do código com a versão da RCL incluem:

- Muitos arquivos são movidos para fora de sua árvore de origem.
- Quaisquer correções de bug ou novos recursos de identidade estão incluídos na [metapacote Microsoft](#). Você obtém a identidade atualizada automaticamente quando `Microsoft.AspNetCore.App` é atualizado.

Se você tiver feito não triviais alterações ao modelo de código de identidade gerado:

- As vantagens anteriores provavelmente fazem isso **não** justificar a conversão para a versão da RCL.
- Você pode manter seu código de identidade do ASP.NET Core 2.0, ele é totalmente suportado.

Identidade 2.1 expõe pontos de extremidade com o `Identity` área. Por exemplo, a tabela a seguir mostra exemplos de pontos de extremidade de identidade que alterar de 2.0 para 2.1:

2.0 URL	2.1 URL DE
/ Account/Login	Identity/logon/conta
/ Conta/Logout	Identity/logoff/conta
/ Conta/gerenciar	/ Identity/conta/gerenciar

Identidade de interface do usuário com a necessidade de biblioteca de identidade 2.1 para levar em conta as URLs de identidade de aplicativos que tem o código usando a identidade e substitua 2.0 têm `/Identity` anexado para os URIs de segmento. É uma maneira de lidar com os novos pontos de extremidade de identidade é configurar redirecionamentos, por exemplo da `/Account/Login` para `/Identity/Account/Login`.

## Atualizar identidade para a versão 2.1

As seguintes opções estão disponíveis para atualizar a identidade para o 2.1.

- Use o código de identidade da interface do usuário 2.0 sem alterações. Usando o código de identidade da interface do usuário 2.0 tem suporte total. Isso é uma boa abordagem quando alterações significativas foram feitas para o código de identidade gerado.
- Excluir seu código 2.0 de identidade existente e [identidade de Scaffold](#) em seu projeto. Seu projeto usará o [ASP.NET Core Identity biblioteca de classes Razor](#). Você pode gerar o código e a interface do usuário para qualquer parte do código de identidade da interface do usuário que você modificou. Aplica as alterações de código para o código gerado automaticamente recém-criado de interface do usuário.
- Excluir seu código 2.0 de identidade existente e [identidade de Scaffold](#) em seu projeto com a opção de **substituir todos os arquivos**.

## Substitua a identidade 2.0 interface do usuário com a biblioteca de classes Razor identidade 2.1

Esta seção descreve as etapas para substituir o código de identidade do ASP.NET Core 2.0 modelo gerado com o [ASP.NET Core Identity biblioteca de classes Razor](#). As etapas a seguir são para um projeto de páginas do Razor, mas a abordagem para um projeto MVC é semelhante.

- Verifique se o [arquivo de projeto é atualizado para usar 2.1 versões](#)
- Exclua as seguintes pastas e todos os arquivos contidos neles:
  - *Controladores*
  - *Páginas/Account /*

- Extensões
- Compile o projeto.
- Criar o scaffolding de identidade em seu projeto:
  - Selecione os projetos saindo \_layout.cshtml arquivo.
  - Selecione o + ícone no lado direito dos **classe de contexto de dados**. Aceite o nome padrão.
  - Selecione **adicionar** para criar uma nova classe de contexto de dados. Criar um novo contexto de dados é necessária para criar o scaffolding. Você pode remover o novo contexto de dados na próxima seção.

### Atualizar após o scaffolding de identidade

- Excluir o scaffolder de identidade gerado `IdentityDbContext` derivado da classe na `áreas/identidade/Data/` pasta.
- Excluir `Areas/Identity/IdentityHostingStartup.cs`
- Atualizar o `_Loginpartial` arquivo:
  - Mover páginas `_Loginpartial` à páginas/Shared/`_Loginpartial`
  - Adicionar `asp-area="Identity"` os links de âncora e o formulário.
  - Atualização de `<form />` elemento
 

```
<form asp-area="Identity" asp-page="/Account/Logout" asp-route-returnUrl="@Url.Page("/Index", new { area = "" })" method="post" id="logoutForm" class="navbar-right">
```

O código a seguir mostra o atualizada `_Loginpartial` arquivo:

```
@using Microsoft.AspNetCore.Identity

@inject SignInManager<ApplicationUser> SignInManager
@inject UserManager<ApplicationUser> UserManager

@if (SignInManager.IsSignedIn(User))
{
    <form asp-area="Identity" asp-page="/Account/Logout" asp-route-returnUrl="@Url.Page("/Index", new { area = "" })" method="post" id="logoutForm" class="navbar-right">
        <ul class="nav navbar-nav navbar-right">
            <li>
                <a asp-area="Identity" asp-page="/Account/Manage/Index" title="Manage">Hello
                    @UserManager.GetUserName(User) !</a>
            </li>
            <li>
                <button type="submit" class="btn btn-link navbar-btn navbar-link">Log out</button>
            </li>
        </ul>
    </form>
}
else
{
    <ul class="nav navbar-nav navbar-right">
        <li><a asp-area="Identity" asp-page="/Account/Register">Register</a></li>
        <li><a asp-area="Identity" asp-page="/Account/Login">Log in</a></li>
    </ul>
}
```

Atualização `ConfigureServices` com o código a seguir:

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddDefaultIdentity<ApplicationUser>()
        .AddEntityFrameworkStores<ApplicationContext>()
        .AddDefaultTokenProviders();

    services.AddMvc();

    // Register no-op EmailSender used by account confirmation and password reset
    // during development
    services.AddSingleton<IEmailSender, EmailSender>();
}

```

## Altera a páginas do Razor projetos de arquivos do Razor

### O arquivo de layout

- Mover *páginas/\_layout.cshtml* à *páginas/Shared/\_layout.cshtml*
- O *layout.cshtml* arquivo tem as seguintes alterações:
  - <partial name="\_CookieConsentPartial" /> é adicionado. Para obter mais informações, veja [Suporte RGPD no ASP.NET Core](#).
  - alterações de jQuery do 2.2.0 para 3.3.1

### \_ValidationScriptsPartial.cshtml

- *Páginas/\_ValidationScriptsPartial.cshtml* move para *páginas/Shared/\_ValidationScriptsPartial.cshtml*
- *jQuery.Validate/1.14.0* muda para *jquery.validate/1.17.0*

### Novos arquivos

Os seguintes arquivos são adicionados:

- *Privacy.cshtml*
- *Privacy.cshtml.cs*

Ver [GDPR suporte no ASP.NET Core](#) para obter informações sobre os arquivos anteriores.

## Alterações em arquivos de projetos do MVC Razor

### O arquivo de layout

O *layout.cshtml* arquivo tem as seguintes alterações:

- <partial name="\_CookieConsentPartial" /> é adicionado.
- alterações de jQuery do 2.2.0 para 3.3.1

### \_ValidationScriptsPartial.cshtml

*jQuery.Validate/1.14.0* muda para *jquery.validate/1.17.0*

### Novos arquivos e métodos de ação

O exemplo a seguir foram adicionados:

- *Views/Home/Privacy.cshtml*
- O *Privacy* método de ação é adicionado ao controlador Home.

Ver [GDPR suporte no ASP.NET Core](#) para obter informações sobre os arquivos anteriores.

## Alterações no arquivo launchsettings.json

Como os aplicativos ASP.NET Core agora usam HTTPS por padrão, o *Properties/launchSettings.json* arquivo foi alterado.

O JSON a seguir mostra o 2.0 anteriores modelo gerado *launchsettings.json* arquivo:

```
{  
    "iisSettings": {  
        "windowsAuthentication": false,  
        "anonymousAuthentication": true,  
        "iisExpress": {  
            "applicationUrl": "http://localhost:1799/",  
            "sslPort": 0  
        }  
    },  
    "profiles": {  
        "IIS Express": {  
            "commandName": "IISExpress",  
            "launchBrowser": true,  
            "environmentVariables": {  
                "ASPNETCORE_ENVIRONMENT": "Development"  
            }  
        },  
        "WebApp1": {  
            "commandName": "Project",  
            "launchBrowser": true,  
            "environmentVariables": {  
                "ASPNETCORE_ENVIRONMENT": "Development"  
            },  
            "applicationUrl": "http://localhost:1798/"  
        }  
    }  
}
```

O JSON a seguir mostra o novo 2.1 modelo gerado *launchsettings.json* arquivo:

```
{  
    "iisSettings": {  
        "windowsAuthentication": false,  
        "anonymousAuthentication": true,  
        "iisExpress": {  
            "applicationUrl": "http://localhost:39191",  
            "sslPort": 44390  
        }  
    },  
    "profiles": {  
        "IIS Express": {  
            "commandName": "IISExpress",  
            "launchBrowser": true,  
            "environmentVariables": {  
                "ASPNETCORE_ENVIRONMENT": "Development"  
            }  
        },  
        "WebApp1": {  
            "commandName": "Project",  
            "launchBrowser": true,  
            "applicationUrl": "https://localhost:5001;http://localhost:5000",  
            "environmentVariables": {  
                "ASPNETCORE_ENVIRONMENT": "Development"  
            }  
        }  
    }  
}
```

Para obter mais informações, consulte [Impor HTTPS no ASP.NET Core](#).

## Alterações adicionais

- Se hospedar o aplicativo no Windows com o IIS, instalar a versão mais recente [pacote de hospedagem do .NET Core](#).
- [SetCompatibilityVersion](#)
- [Configuração de transporte](#)

# Migrar do ASP.NET Core 1.x para 2.0

26/10/2018 • 13 minutes to read • [Edit Online](#)

Por [Scott Addie](#)

Neste artigo, vamos orientá-lo pela atualização de um projeto existente ASP.NET Core 1.x para o ASP.NET Core 2.0. A migração do aplicativo para o ASP.NET Core 2.0 permite que você aproveite [muitos novos recursos e melhorias de desempenho](#).

Os aplicativos ASP.NET Core 1.x existentes baseiam-se em modelos de projeto específicos à versão. Conforme a estrutura do ASP.NET Core evolui, os modelos do projeto e o código inicial contido neles também. Além de atualizar a estrutura ASP.NET Core, você precisa atualizar o código para o aplicativo.

## Pré-requisitos

Veja a [Introdução ao ASP.NET Core](#).

## Atualizar TFM (Moniker da Estrutura de Destino)

Projetos direcionados ao .NET Core devem usar o [TFM](#) de uma versão maior ou igual ao .NET Core 2.0. Pesquise o nó `<TargetFramework>` no arquivo `.csproj` e substitua seu texto interno por `netcoreapp2.0`:

```
<TargetFramework>netcoreapp2.0</TargetFramework>
```

Projetos direcionados ao .NET Framework devem usar o TFM de uma versão maior ou igual ao .NET Framework 4.6.1. Pesquise o nó `<TargetFramework>` no arquivo `.csproj` e substitua seu texto interno por `net461`:

```
<TargetFramework>net461</TargetFramework>
```

### NOTE

O .NET Core 2.0 oferece uma área de superfície muito maior do que o .NET Core 1.x. Se você estiver direcionando o .NET Framework exclusivamente devido a APIs ausentes no .NET Core 1.x, o direcionamento do .NET Core 2.0 provavelmente funcionará.

## Atualizar a versão do SDK do .NET Core em `global.json`

Se a solução depender de um arquivo [global.json](#) para direcionar uma versão específica do SDK do .NET Core, atualize sua propriedade `version` para que ela use a versão 2.0 instalada no computador:

```
{
  "sdk": {
    "version": "2.0.0"
  }
}
```

## Referências do pacote de atualização

O arquivo `.csproj` em um projeto 1.x lista cada pacote NuGet usado pelo projeto.

Em um projeto ASP.NET Core 2.0 direcionado ao .NET Core 2.0, uma única referência de [metapacote](#) no arquivo `.csproj` substitui a coleção de pacotes:

```
<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.9" />
</ItemGroup>
```

Todos os recursos do ASP.NET Core 2.0 e do Entity Framework Core 2.0 são incluídos no metapacote.

Os projetos do ASP.NET Core 2.0 direcionados ao .NET Framework devem continuar a referenciar pacotes NuGet individuais. Atualize o atributo `Version` de cada nó `<PackageReference />` para 2.0.0.

Por exemplo, esta é a lista de nós `<PackageReference />` usados em um projeto ASP.NET Core 2.0 típico direcionado ao .NET Framework:

```
<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore" Version="2.0.0" />
  <PackageReference Include="Microsoft.AspNetCore.Authentication.Cookies" Version="2.0.0" />
  <PackageReference Include="Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore" Version="2.0.0" />
  <PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="2.0.0" />
  <PackageReference Include="Microsoft.AspNetCore.Mvc" Version="2.0.0" />
  <PackageReference Include="Microsoft.AspNetCore.Mvc.Razor.ViewCompilation" Version="2.0.0" />
  PrivateAssets="All" />
  <PackageReference Include="Microsoft.AspNetCore.StaticFiles" Version="2.0.0" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="2.0.0" PrivateAssets="All" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="2.0.0" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="2.0.0" PrivateAssets="All" />
  <PackageReference Include="Microsoft.VisualStudio.Web.BrowserLink" Version="2.0.0" />
  <PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Design" Version="2.0.0" />
  PrivateAssets="All" />
</ItemGroup>
```

## Atualizar ferramentas da CLI do .NET Core

No arquivo `.csproj`, atualize o atributo `Version` de cada nó `<DotNetCliToolReference />` para 2.0.0.

Por exemplo, esta é a lista de ferramentas da CLI usadas em um projeto ASP.NET Core 2.0 típico direcionado ao .NET Core 2.0:

```
<ItemGroup>
  <DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet" Version="2.0.0" />
  <DotNetCliToolReference Include="Microsoft.Extensions.SecretManager.Tools" Version="2.0.0" />
  <DotNetCliToolReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Tools" Version="2.0.0" />
</ItemGroup>
```

## Renomear a propriedade de Fallback de Destino do Pacote

O arquivo `.csproj` de um projeto 1.x usou um nó `PackageTargetFallback` e uma variável:

```
<PackageTargetFallback>$(<PackageTargetFallback>) ; portable-net45+win8+wp8+wpa81;</PackageTargetFallback>
```

Renomeie o nó e a variável como `AssetTargetFallback`:

```
<AssetTargetFallback>$(AssetTargetFallback);portable-net45+win8+wp8+wpa81;</AssetTargetFallback>
```

## Atualizar o método Main em Program.cs

Em projetos 1.x, o método `Main` de *Program.cs* era parecido com este:

```
using System.IO;
using Microsoft.AspNetCore.Hosting;

namespace AspNetCoreDotNetCore1App
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var host = new WebHostBuilder()
                .UseKestrel()
                .UseContentRoot(Directory.GetCurrentDirectory())
                .UseIISIntegration()
                .UseStartup<Startup>()
                .UseApplicationInsights()
                .Build();

            host.Run();
        }
    }
}
```

Em projetos 2.0, o método `Main` de *Program.cs* foi simplificado:

```
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;

namespace AspNetCoreDotNetCore2App
{
    public class Program
    {
        public static void Main(string[] args)
        {
            BuildWebHost(args).Run();
        }

        public static IWebHost BuildWebHost(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>()
                .Build();
    }
}
```

A adoção desse novo padrão do 2.0 é altamente recomendada e é necessária para que os recursos de produto como as [Migrações do Entity Framework \(EF\) Core](#) funcionem. Por exemplo, a execução de `Update-Database` na janela do Console do Gerenciador de Pacotes ou de `dotnet ef database update` na linha de comando (em projetos convertidos no ASP.NET Core 2.0) gera o seguinte erro:

```
Unable to create an object of type '<Context>'. Add an implementation of
'IDesignTimeDbContextFactory<Context>' to the project, or see https://go.microsoft.com/fwlink/?linkid=851728
for additional patterns supported at design time.
```

## Adicionar provedores de configuração

Nos projetos 1.x, a adição de provedores de configuração em um aplicativo foi realizada por meio do construtor `Startup`. As etapas incluíram a criação de uma instância de `ConfigurationBuilder`, o carregamento de provedores aplicáveis (variáveis de ambiente, configurações do aplicativo, etc.) e a inicialização de um membro de `IConfigurationRoot`.

```
public Startup(IHostingEnvironment env)
{
    var builder = new ConfigurationBuilder()
        .SetBasePath(env.ContentRootPath)
        .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)
        .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true);

    if (env.IsDevelopment())
    {
        builder.AddUserSecrets<Startup>();
    }

    builder.AddEnvironmentVariables();
    Configuration = builder.Build();
}

public IConfigurationRoot Configuration { get; }
```

O exemplo anterior carrega o membro `Configuration` com definições de configuração do `appsettings.json`, bem como qualquer arquivo `appsettings.<EnvironmentName>.json` correspondente à propriedade `IHostingEnvironment.EnvironmentName`. Estes arquivos estão localizados no mesmo caminho que `Startup.cs`.

Nos projetos 2.0, o código de configuração clichê inherente aos projetos 1.x é executado de modo oculto. Por exemplo, as variáveis de ambiente e as configurações do aplicativo são carregadas na inicialização. O código `Startup.cs` equivalente é reduzido à inicialização `IConfiguration` com a instância injetada:

```
public Startup(IConfiguration configuration)
{
    Configuration = configuration;
}

public IConfiguration Configuration { get; }
```

Para remover os provedores padrão adicionados por `WebHostBuilder.CreateDefaultBuilder`, invoque o método `Clear` na propriedade `IConfigurationBuilder.Sources` dentro de `ConfigureAppConfiguration`. Para adicionar os provedores de volta, utilize o método `ConfigureAppConfiguration` em `Program.cs`:

```
public static void Main(string[] args)
{
    BuildWebHost(args).Run();
}

public static IWebHost BuildWebHost(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .ConfigureAppConfiguration((hostContext, config) =>
    {
        // delete all default configuration providers
        config.Sources.Clear();
        config.AddJsonFile("myconfig.json", optional: true);
    })
    .Build();
```

A configuração usada pelo método `CreateDefaultBuilder` no snippet de código anterior pode ser vista [aqui](#).

Para obter mais informações, consulte [Configuração no ASP.NET Core](#).

## Mover código de inicialização do banco de dados

Em projetos do 1.x usando o EF Core 1.x, um comando como `dotnet ef migrations add` faz o seguinte:

1. Cria uma instância `Startup`
2. Invoca o método `ConfigureServices` para registrar todos os serviços com injeção de dependência (incluindo tipos `DbContext`)
3. Executa suas tarefas de requisito

Em projetos do 2.0 usando o EF Core 2.0, o `Program.BuildWebHost` é chamado para obter os serviços do aplicativo. Ao contrário do 1.x, isso tem o efeito colateral adicional de chamar o `Startup.Configure`. Caso seu aplicativo do 1.x tenha chamado o código de inicialização do banco de dados no seu método `Configure`, problemas inesperados poderão ocorrer. Por exemplo, se o banco de dados não existir ainda, o código de propagação será executado antes da execução do comando EF Core Migrations. Esse problema fará com que um comando `dotnet ef migrations list` falhe se o banco de dados ainda não existir.

Considere o seguinte código de inicialização de propagação do 1.x no método `Configure` de `Startup.cs`:

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});

SeedData.Initialize(app.ApplicationServices);
```

Em projetos do 2.0, mova a chamada `SeedData.Initialize` para o método `Main` do `Program.cs`:

```
var host = BuildWebHost(args);

using (var scope = host.Services.CreateScope())
{
    var services = scope.ServiceProvider;

    try
    {
        // Requires using RazorPagesMovie.Models;
        SeedData.Initialize(services);
    }
    catch (Exception ex)
    {
        var logger = services.GetRequiredService<ILogger<Program>>();
        logger.LogError(ex, "An error occurred seeding the DB.");
    }
}

host.Run();
```

A partir do 2.0, é uma prática inadequada realizar qualquer ação no `BuildWebHost`, exceto compilação e configuração do host da Web. Tudo relacionado à execução do aplicativo deve ser tratado fora do `BuildWebHost` —, normalmente no método `Main` do `Program.cs`.

## Examinar a configuração de compilação do Modo de Exibição do Razor

Um tempo de inicialização mais rápido do aplicativo e pacotes publicados menores são de extrema importância para você. Por esses motivos, a opção [Compilação de exibição do Razor](#) é habilitada por padrão no ASP.NET Core 2.0.

A configuração da propriedade `MvcRazorCompileOnPublish` como verdadeiro não é mais necessária. A menos que você esteja desabilitando a compilação de exibição, a propriedade poderá ser removida do arquivo `.csproj`.

Ao direcionar o .NET Framework, você ainda precisa referenciar explicitamente o pacote NuGet [Microsoft.AspNetCore.Mvc.Razor.ViewCompilation](#) no arquivo `.csproj`:

```
<PackageReference Include="Microsoft.AspNetCore.Mvc.Razor.ViewCompilation" Version="2.0.0" PrivateAssets="All" />
```

## Depender dos recursos “Light-Up” do Application Insights

A instalação fácil da instrumentação de desempenho do aplicativo é importante. Agora, você pode contar com os novos recursos “light-up” do [Application Insights](#) disponíveis nas ferramentas do Visual Studio 2017.

Os projetos do ASP.NET Core 1.1 criados no Visual Studio 2017 adicionaram o Application Insights por padrão. Se não estiver usando o SDK do Application Insights diretamente, fora de `Program.cs` e `Startup.cs`, siga estas etapas:

1. Se você estiver direcionando ao .NET Core, remova o nó `<PackageReference />` seguinte do arquivo `.csproj`:

```
<PackageReference Include="Microsoft.ApplicationInsights.AspNetCore" Version="2.0.0" />
```

2. Se você estiver direcionando ao .NET Core, remova a invocação do método de extensão `UseApplicationInsights` de `Program.cs`:

```
public static void Main(string[] args)
{
    var host = new WebHostBuilder()
        .UseKestrel()
        .UseContentRoot(Directory.GetCurrentDirectory())
        .UseIISIntegration()
        .UseStartup<Startup>()
        .UseApplicationInsights()
        .Build();

    host.Run();
}
```

3. Remova a chamada à API do lado do cliente do Application Insights de `_Layout.cshtml`. Ela inclui as duas seguintes linhas de código:

```
@inject Microsoft.ApplicationInsights.AspNetCore.JavaScriptSnippet JavaScriptSnippet
@Html.Raw(JavaScriptSnippet.FullScript)
```

Se você estiver usando o SDK do Application Insights diretamente, continue fazendo isso. O [pacote](#) do 2.0 inclui a última versão do Application Insights. Portanto, um erro de downgrade do pacote será exibido se você estiver referenciando uma versão mais antiga.

## Adotar melhorias de autenticação/identidade

O ASP.NET Core 2.0 tem um novo modelo de autenticação e uma série de alterações significativas no ASP.NET

Core Identity. Se você criar o projeto com a opção Contas de Usuário Individuais habilitada ou se adicionar manualmente a autenticação ou a identidade, consulte [Migrar a autenticação e a identidade para o ASP.NET Core 2.0](#).

## Recursos adicionais

- [Últimas alterações no ASP.NET Core 2.0](#)

# Migrar autenticação e identidade para o ASP.NET Core 2.0

27/12/2018 • 13 minutes to read • [Edit Online](#)

Por [Scott Addie](#) e [Kung Hao](#)

ASP.NET Core 2.0 tem um novo modelo para autenticação e [identidade](#) que simplifica a configuração por meio de serviços. Aplicativos ASP.NET Core 1.x que usam autenticação ou a identidade podem ser atualizados para usar o novo modelo, conforme descrito abaixo.

## Middleware de autenticação e serviços

Em projetos 1.x, a autenticação é configurada por meio do middleware. Um método de middleware é invocado para cada esquema de autenticação que você deseja dar suporte.

O exemplo a seguir 1.x configura a autenticação do Facebook com identidade no *Startup.cs*:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationContext>();

}

public void Configure(IApplicationBuilder app, ILoggerFactory loggerfactory)
{
    app.UseIdentity();
    app.UseFacebookAuthentication(new FacebookOptions {
        AppId = Configuration["auth:facebook:appid"],
        AppSecret = Configuration["auth:facebook:appsecret"]
    });
}
```

Em projetos do 2.0, a autenticação é configurada por meio de serviços. Cada esquema de autenticação é registrada na `ConfigureServices` método de *Startup.cs*. O `UseIdentity` o método é substituído por `UseAuthentication`.

O exemplo a seguir 2.0 configura a autenticação do Facebook com identidade no *Startup.cs*:

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationContext>();

    // If you want to tweak Identity cookies, they're no longer part of IdentityOptions.
    services.ConfigureApplicationCookie(options => options.LoginPath = "/Account/LogIn");
    services.AddAuthentication()
        .AddFacebook(options =>
    {
        options.AppId = Configuration["auth:facebook:appid"];
        options.AppSecret = Configuration["auth:facebook:appsecret"];
    });
}

public void Configure(IApplicationBuilder app, IBuilderFactory loggerfactory) {
    app.UseAuthentication();
}

```

O `UseAuthentication` método adiciona um componente de middleware de autenticação único que é responsável pela autenticação automática e o tratamento de solicitações de autenticação remota. Ele substitui todos os componentes de middleware individuais com um componente de middleware simples e comum.

Veja abaixo as instruções de migração de 2.0 para cada esquema de autenticação principal.

### Autenticação baseada em cookie

Selecione uma das duas opções abaixo e faça as alterações necessárias na `Startup.cs`:

#### 1. Usar cookies com identidade

- Substitua `UseIdentity` com `UseAuthentication` no `Configure` método:

```
app.UseAuthentication();
```

- Invocar o `AddIdentity` método no `ConfigureServices` método para adicionar os serviços de autenticação de cookie.
- Opcionalmente, invocar o `ConfigureApplicationCookie` ou `ConfigureExternalCookie` método no `ConfigureServices` método ajustar as configurações de cookie de identidade.

```

services.AddIdentity<ApplicationUser, IdentityRole>()
    .AddEntityFrameworkStores<ApplicationContext>()
    .AddDefaultTokenProviders();

services.ConfigureApplicationCookie(options => options.LoginPath = "/Account/LogIn");

```

#### 2. Usar cookies sem o Identity

- Substitua os `UseCookieAuthentication` chamada de método na `Configure` método com `UseAuthentication`:

```
app.UseAuthentication();
```

- Invocar o `AddAuthentication` e `AddCookie` métodos no `ConfigureServices` método:

```

// If you don't want the cookie to be automatically authenticated and assigned to
HttpContext.User,
// remove the CookieAuthenticationDefaults.AuthenticationScheme parameter passed to
AddAuthentication.
services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie(options =>
{
    options.LoginPath = "/Account/LogIn";
    options.LogoutPath = "/Account/LogOff";
});

```

## Autenticação do portador do JWT

Faça as seguintes alterações na *Startup.cs*:

- Substitua os `UseJwtBearerAuthentication` chamada de método na `Configure` método com `UseAuthentication`:

```
app.UseAuthentication();
```

- Invocar o `AddJwtBearer` método no `ConfigureServices` método:

```

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
{
    options.Audience = "http://localhost:5001/";
    options.Authority = "http://localhost:5000/";
});

```

Este trecho de código não usa a identidade, portanto, o esquema padrão deve ser definido, passando `JwtBearerDefaults.AuthenticationScheme` para o `AddAuthentication` método.

## Autenticação OIDC (OpenID Connect)

Faça as seguintes alterações na *Startup.cs*:

- Substitua os `UseOpenIdConnectAuthentication` chamada de método na `Configure` método com `UseAuthentication`:

```
app.UseAuthentication();
```

- Invocar o `AddOpenIdConnect` método no `ConfigureServices` método:

```

services.AddAuthentication(options =>
{
    options.DefaultScheme = CookieAuthenticationDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = OpenIdConnectDefaults.AuthenticationScheme;
})
.AddCookie()
.AddOpenIdConnect(options =>
{
    options.Authority = Configuration["auth:oidc:authority"];
    options.ClientId = Configuration["auth:oidc:clientid"];
});

```

## autenticação do Facebook

Faça as seguintes alterações na *Startup.cs*:

- Substitua os `UseFacebookAuthentication` chamada de método na `Configure` método com `UseAuthentication` :

```
app.UseAuthentication();
```

- Invocar o `AddFacebook` método no `ConfigureServices` método:

```
services.AddAuthentication()
    .AddFacebook(options =>
{
    options.AppId = Configuration["auth:facebook:appid"];
    options.AppSecret = Configuration["auth:facebook:appsecret"];
});
```

## Autenticação do Google

Faça as seguintes alterações na `Startup.cs`:

- Substitua os `UseGoogleAuthentication` chamada de método na `Configure` método com `UseAuthentication` :

```
app.UseAuthentication();
```

- Invocar o `AddGoogle` método no `ConfigureServices` método:

```
services.AddAuthentication()
    .AddGoogle(options =>
{
    options.ClientId = Configuration["auth:google:clientid"];
    options.ClientSecret = Configuration["auth:google:clientsecret"];
});
```

## Autenticação da Microsoft Account

Faça as seguintes alterações na `Startup.cs`:

- Substitua os `UseMicrosoftAccountAuthentication` chamada de método na `Configure` método com `UseAuthentication` :

```
app.UseAuthentication();
```

- Invocar o `AddMicrosoftAccount` método no `ConfigureServices` método:

```
services.AddAuthentication()
    .AddMicrosoftAccount(options =>
{
    options.ClientId = Configuration["auth:microsoft:clientid"];
    options.ClientSecret = Configuration["auth:microsoft:clientsecret"];
});
```

## Autenticação do Twitter

Faça as seguintes alterações na `Startup.cs`:

- Substitua os `UseTwitterAuthentication` chamada de método na `Configure` método com `UseAuthentication` :

```
app.UseAuthentication();
```

- Invocar o `AddTwitter` método no `ConfigureServices` método:

```
services.AddAuthentication()
    .AddTwitter(options =>
{
    options.ConsumerKey = Configuration["auth:twitter:consumerkey"];
    options.ConsumerSecret = Configuration["auth:twitter:consumersecret"];
});
```

## Esquemas de autenticação padrão de configuração

No 1.x, o `AutomaticAuthenticate` e `AutomaticChallenge` propriedades da `AuthenticationOptions` classe base foram deve ser definida em um esquema de autenticação. Não havia nenhuma boa maneira para impor isso.

No 2.0, essas duas propriedades foram removidas como propriedades individual `AuthenticationOptions` instância. Eles podem ser configurados na `AddAuthentication` chamada de método dentro a `ConfigureServices` método de `Startup.cs`:

```
services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme);
```

No trecho de código anterior, o esquema padrão é definido como

```
CookieAuthenticationDefaults.AuthenticationScheme ("Cookies").
```

Como alternativa, use uma versão sobrecarregada do `AddAuthentication` método para definir mais de uma propriedade. No exemplo a seguir o método sobrecarregado, o esquema padrão é definido como `CookieAuthenticationDefaults.AuthenticationScheme`. O esquema de autenticação como alternativa, pode ser especificado dentro de seu indivíduo `[Authorize]` atributos ou as políticas de autorização.

```
services.AddAuthentication(options =>
{
    options.DefaultScheme = CookieAuthenticationDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = OpenIdConnectDefaults.AuthenticationScheme;
});
```

Defina um esquema padrão do 2.0, se uma das seguintes condições for verdadeira:

- Você deseja que o usuário a ser conectado automaticamente
- Você usa o `[Authorize]` políticas de autorização ou atributo sem especificar esquemas

Uma exceção a essa regra é o `AddIdentity` método. Esse método adiciona cookies para você e define o padrão autenticar e esquemas de desafio para o cookie do aplicativo `IdentityConstants.ApplicationScheme`. Além disso, ele define o esquema de entrada padrão para o cookie externo `IdentityConstants.ExternalScheme`.

## Usar extensões de autenticação `HttpContext`

O `IAuthenticationManager` interface é o ponto de entrada principal para o sistema de autenticação de 1.x. Ele foi substituído por um novo conjunto de `HttpContext` métodos de extensão no `Microsoft.AspNetCore.Authentication` namespace.

Por exemplo, projetos do 1.x referência um `Authentication` propriedade:

```
// Clear the existing external cookie to ensure a clean login process
await HttpContext.Authentication.SignOutAsync(_externalCookieScheme);
```

Em 2.0 projetos, importe as `Microsoft.AspNetCore.Authentication` namespace e exclua o `Authentication` referências de propriedade:

```
// Clear the existing external cookie to ensure a clean login process
await HttpContext.SignOutAsync(IdentityConstants.ExternalScheme);
```

## Autenticação do Windows (`http.sys / IISIntegration`)

Há duas variações de autenticação do Windows:

1. O host permite que somente usuários autenticados
2. O host permite que ambos anônimos e usuários autenticados

A primeira descrita acima não é afetada pelas alterações de 2.0.

A segunda descrita acima é afetada pelas alterações de 2.0. Por exemplo, você pode ser permitir que usuários anônimos em seu aplicativo no IIS ou `HTTP.sys` mas autorizar usuários no nível do controlador de camada.

Nesse cenário, defina o esquema padrão `IISDefaults.AuthenticationScheme` no `Startup.ConfigureServices` método:

```
services.AddAuthentication(IISDefaults.AuthenticationScheme);
```

Falha ao definir o esquema padrão adequadamente impede que a solicitação de autorização de desafio do trabalho.

## Instâncias de `IdentityCookieOptions`

Um efeito colateral das 2.0 alterações é o comutador que usa o chamado opções em vez de instâncias de opções de cookie. A capacidade de personalizar os nomes de esquema do cookie de identidade é removida.

Por exemplo, 1.x projetos usam [injeção de construtor](#) para passar um `IdentityCookieOptions` parâmetro em `AccountController.cs`. O esquema de autenticação de cookie externa é acessado da instância fornecida:

```
public AccountController(
    UserManager< ApplicationUser > userManager,
    SignInManager< ApplicationUser > signInManager,
    IOptions< IdentityCookieOptions > identityCookieOptions,
    IEmailSender emailSender,
    ISmsSender smsSender,
    ILoggerFactory loggerFactory)
{
    _userManager = userManager;
    _signInManager = signInManager;
    _externalCookieScheme = identityCookieOptions.Value.ExternalCookieAuthenticationScheme;
    _emailSender = emailSender;
    _smsSender = smsSender;
    _logger = loggerFactory.CreateLogger< AccountController >();
}
```

A injeção de construtor mencionados anteriormente se torna desnecessária em 2.0 projetos e o `_externalCookieScheme` campo pode ser excluído:

```

public AccountController(
    UserManager< ApplicationUser > userManager,
    SignInManager< ApplicationUser > signInManager,
    IEmailSender emailSender,
    ISmsSender smsSender,
    ILoggerFactory loggerFactory)
{
    _userManager = userManager;
    _signInManager = signInManager;
    _emailSender = emailSender;
    _smsSender = smsSender;
    _logger = loggerFactory.CreateLogger< AccountController >();
}

```

O `IdentityConstants.ExternalScheme` constante pode ser usada diretamente:

```

// Clear the existing external cookie to ensure a clean login process
await HttpContext.SignOutAsync(IdentityConstants.ExternalScheme);

```

## Adicionar propriedades de navegação IdentityUser POCO

As propriedades de navegação do Entity Framework (EF) Core da base de `IdentityUser` POCO (objeto Plain Old CLR) foram removidos. Se seu projeto 1.x usou essas propriedades, manualmente adicione ao projeto 2.0:

```

/// <summary>
/// Navigation property for the roles this user belongs to.
/// </summary>
public virtual ICollection< IdentityUserRole< int > > Roles { get; } = new List< IdentityUserRole< int > >();

/// <summary>
/// Navigation property for the claims this user possesses.
/// </summary>
public virtual ICollection< IdentityUserClaim< int > > Claims { get; } = new List< IdentityUserClaim< int > >();

/// <summary>
/// Navigation property for this users login accounts.
/// </summary>
public virtual ICollection< IdentityUserLogin< int > > Logins { get; } = new List< IdentityUserLogin< int > >();

```

Para impedir que as chaves estrangeiras duplicadas ao executar migrações do EF Core, adicione o seguinte ao seu `IdentityDbContext` classe `OnModelCreating` método (depois que o `base.OnModelCreating();` chamar):

```

protected override void OnModelCreating(ModelBuilder builder)
{
    base.OnModelCreating(builder);
    // Customize the ASP.NET Core Identity model and override the defaults if needed.
    // For example, you can rename the ASP.NET Core Identity table names and more.
    // Add your customizations after calling base.OnModelCreating(builder);

    builder.Entity<ApplicationUser>()
        .HasMany(e => e.Claims)
        .WithOne()
        .HasForeignKey(e => e.UserId)
        .IsRequired()
        .OnDelete(DeleteBehavior.Cascade);

    builder.Entity< ApplicationUser >()
        .HasMany(e => e.Logins)
        .WithOne()
        .HasForeignKey(e => e.UserId)
        .IsRequired()
        .OnDelete(DeleteBehavior.Cascade);

    builder.Entity< ApplicationUser >()
        .HasMany(e => e.Roles)
        .WithOne()
        .HasForeignKey(e => e.UserId)
        .IsRequired()
        .OnDelete(DeleteBehavior.Cascade);
}

```

## Substituir GetExternalAuthenticationSchemes

O método síncrono `GetExternalAuthenticationSchemes` foi removido para uma versão assíncrona. projetos do 1.x tem o seguinte código `ManageController.cs`:

```

var otherLogins = _signInManager.GetExternalAuthenticationSchemes().Where(auth => userLogins.All(ul =>
auth.AuthenticationScheme != ul.LoginProvider)).ToList();

```

Este método é exibido no `cshtml` muito:

```

var loginProviders = SignInManager.GetExternalAuthenticationSchemes().ToList();
<div>
    <p>
        @foreach (var provider in loginProviders)
        {
            <button type="submit" class="btn btn-default" name="provider"
value="@provider.AuthenticationScheme" title="Log in using your @provider.DisplayName
account">@provider.AuthenticationScheme</button>
        }
    </p>
</div>
</form>
}

```

Em projetos do 2.0, use o `GetExternalAuthenticationSchemesAsync` método:

```

var schemes = await _signInManager.GetExternalAuthenticationSchemesAsync();
var otherLogins = schemes.Where(auth => userLogins.All(ul => auth.Name != ul.LoginProvider)).ToList();

```

Na `cshtml`, o `AuthenticationScheme` propriedade acessada no `foreach` loop é alterado para `Name`:

```

var loginProviders = (await SignInManager.GetExternalAuthenticationSchemesAsync()).ToList();
    <div>
        <p>
            @foreach (var provider in loginProviders)
            {
                <button type="submit" class="btn btn-default" name="provider" value="@provider.Name"
title="Log in using your @provider.DisplayName account">@provider.DisplayName</button>
            }
        </p>
    </div>
</form>
}

```

## Alteração da propriedade ManageLoginsViewModel

Um `ManageLoginsViewModel` objeto é usado em de `ManageLogins` ação de `ManageController.cs`. Em projetos 1.x, o objeto `OtherLogins` propriedade de tipo de retorno é `IList<AuthenticationDescription>`. Esse tipo de retorno requer uma importação de `Microsoft.AspNetCore.Http.Authentication`:

```

using System.Collections.Generic;
using Microsoft.AspNetCore.Http.Authentication;
using Microsoft.AspNetCore.Identity;

namespace AspNetCoreDotNetCore1App.Models.ManageViewModels
{
    public class ManageLoginsViewModel
    {
        public IList<UserLoginInfo> CurrentLogins { get; set; }

        public IList<AuthenticationDescription> OtherLogins { get; set; }
    }
}

```

Em projetos do 2.0, o tipo de retorno é alterado para `IList<AuthenticationScheme>`. Esse novo tipo de retorno requer substituindo o `Microsoft.AspNetCore.Http.Authentication` importação de um `Microsoft.AspNetCore.Authentication` importar.

```

using System.Collections.Generic;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Identity;

namespace AspNetCoreDotNetCore2App.Models.ManageViewModels
{
    public class ManageLoginsViewModel
    {
        public IList<UserLoginInfo> CurrentLogins { get; set; }

        public IList<AuthenticationScheme> OtherLogins { get; set; }
    }
}

```

## Recursos adicionais

Para obter mais detalhes e discussões, consulte o [discussão do Auth 2.0](#) problema no GitHub.

# Migrar do ASP.NET para o ASP.NET Core

13/12/2018 • 13 minutes to read • [Edit Online](#)

Por [Isaac Levin](#)

Este artigo serve como um guia de referência para migração de aplicativos ASP.NET para o ASP.NET Core.

## Pré-requisitos

[SDK 2.2 ou posterior do .NET Core](#)

## Frameworks de destino

Projetos do ASP.NET Core oferecem aos desenvolvedores a flexibilidade de direcionamento para o .NET Core, .NET Framework ou ambos. Consulte [Escolhendo entre o .NET Core e .NET Framework para aplicativos de servidor](#) para determinar qual estrutura de destino é mais apropriada.

Ao usar o .NET Framework como destino, projetos precisam fazer referência a pacotes NuGet individuais.

Usar o .NET Core como destino permite que você elimine várias referências de pacote explícitas, graças ao [metapacote](#) do ASP.NET Core. Instale o metapacote `Microsoft.AspNetCore.App` em seu projeto:

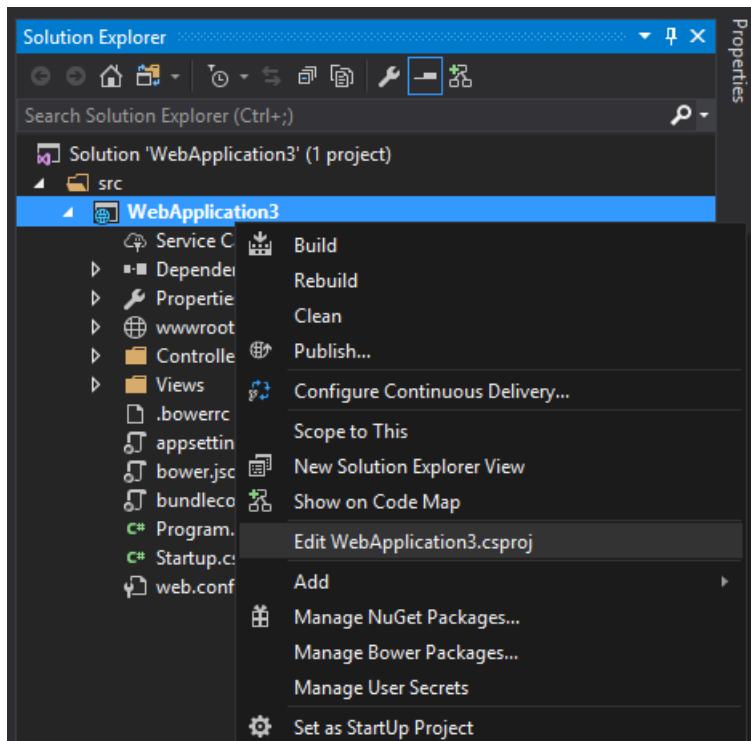
```
<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore.App" />
</ItemGroup>
```

Quando o metapacote é usado, nenhum pacote referenciado no metapacote é implantado com o aplicativo. O repositório de tempo de execução do .NET Core inclui esses ativos e eles são pré-compilados para melhorar o desempenho. Para saber mais, confira [Metapacote Microsoft.AspNetCore.App para ASP.NET Core](#).

## Diferenças de estrutura do projeto

O formato de arquivo `.csproj` foi simplificado no ASP.NET Core. Algumas alterações importantes incluem:

- A inclusão explícita de arquivos não é necessária para que eles possam ser considerados como parte do projeto. Isso reduz o risco de conflitos de mesclagem XML ao trabalhar em grandes equipes.
- Não há referências baseadas em GUID a outros projetos, o que melhora a legibilidade do arquivo.
- O arquivo pode ser editado sem descarregá-lo no Visual Studio:



## Substituição do arquivo Global.asax

O ASP.NET Core introduziu um novo mecanismo para inicializar um aplicativo. O ponto de entrada para aplicativos ASP.NET é o arquivo *Global.asax*. Tarefas como configuração de roteamento e registros de filtro e de área são tratadas no arquivo *Global.asax*.

```
public class MvcApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        AreaRegistration.RegisterAllAreas();
        FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
        RouteConfig.RegisterRoutes(RouteTable.Routes);
        BundleConfig.RegisterBundles(BundleTable.Bundles);
    }
}
```

Essa abordagem associa o aplicativo e o servidor no qual ele é implantado de uma forma que interfere na implementação. Em um esforço para desassociar, a [OWIN](#) foi introduzida para fornecer uma maneira mais limpa de usar várias estruturas juntas. A OWIN fornece um pipeline para adicionar somente os módulos necessários. O ambiente de hospedagem leva uma função [Startup](#) para configurar serviços e pipeline de solicitação do aplicativo. [Startup](#) registra um conjunto de middleware com o aplicativo. Para cada solicitação, o aplicativo chama cada um dos componentes de middleware com o ponteiro de cabeçalho de uma lista vinculada para um conjunto existente de manipuladores. Cada componente de middleware pode adicionar um ou mais manipuladores para a pipeline de tratamento de solicitação. Isso é feito retornando uma referência para o manipulador que é o novo cabeçalho da lista. Cada manipulador é responsável por se lembrar do próximo manipulador na lista e por invocá-lo. Com o ASP.NET Core, o ponto de entrada para um aplicativo é [startup](#) e você não tem mais uma dependência de *Global.asax*. Ao usar a OWIN com o .NET Framework, use algo parecido com o seguinte como um pipeline:

```

using Owin;
using System.Web.Http;

namespace WebApi
{
    // Note: By default all requests go through this OWIN pipeline. Alternatively you can turn this off by
    // adding an appSetting owin:AutomaticAppStartup with value "false".
    // With this turned off you can still have OWIN apps listening on specific routes by adding routes in
    global.asax file using MapOwinPath or MapOwinRoute extensions on RouteTable.Routes
    public class Startup
    {
        // Invoked once at startup to configure your application.
        public void Configuration(IAppBuilder builder)
        {
            HttpConfiguration config = new HttpConfiguration();
            config.Routes.MapHttpRoute("Default", "{controller}/{customerID}", new { controller = "Customer",
customerID = RouteParameter.Optional });

            config.Formatters.XmlFormatter.UseXmlSerializer = true;
            config.Formatters.Remove(config.Formatters.JsonFormatter);
            // config.Formatters.JsonFormatter.UseDataContractJsonSerializer = true;

            builder.UseWebApi(config);
        }
    }
}

```

Isso configura as rotas padrão e usa XmlSerialization em Json por padrão. Adicione outro Middleware para este pipeline conforme necessário (carregamento de serviços, definições de configuração, arquivos estáticos, etc.).

O ASP.NET Core usa uma abordagem semelhante, mas não depende de OWIN para manipular a entrada. Em vez disso, isso é feito por meio do método `Main` de `Program.cs` (semelhante a aplicativos de console) e `Startup` é carregado por lá.

```

using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;

namespace WebApplication2
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateWebHostBuilder(args).Build().Run();
        }

        public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
           WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>();
    }
}

```

`Startup` deve incluir um método `Configure`. Em `Configure`, adicione o middleware necessário ao pipeline. No exemplo a seguir (com base no modelo de site da Web padrão), os métodos de extensão configuram o pipeline com suporte para:

- Páginas de erro
- Segurança de Transporte Estrita de HTTP
- Redirecionamento de HTTP para HTTPS
- ASP.NET Core MVC

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseMvc();
}
```

O host e o aplicativo foram separados, o que fornece a flexibilidade de mover para uma plataforma diferente no futuro.

#### NOTE

Para obter uma referência mais aprofundada sobre o Startup do ASP.NET Core e middleware, veja [Startup no ASP.NET Core](#)

## Armazenar configurações

O ASP.NET dá suporte ao armazenamento de configurações. Essas configurações são usadas, por exemplo, para dar suporte ao ambiente no qual os aplicativos foram implantados. Uma prática comum era armazenar todos os pares chave-valor personalizados na seção `<appSettings>` do arquivo `Web.config`:

```
<appSettings>
  <add key="UserName" value="User" />
  <add key="Password" value="Password" />
</appSettings>
```

Aplicativos leem essas configurações usando a coleção `ConfigurationManager.AppSettings` no namespace `System.Configuration`:

```
string userName = System.Web.Configuration.ConfigurationManager.AppSettings["UserName"];
string password = System.Web.Configuration.ConfigurationManager.AppSettings["Password"];
```

O ASP.NET Core pode armazenar dados de configuração para o aplicativo em qualquer arquivo e carregá-los como parte da inicialização de middleware. O arquivo padrão usado em modelos de projeto é `appsettings.json`:

```
{
    "Logging": {
        "IncludeScopes": false,
        "LogLevel": {
            "Default": "Debug",
            "System": "Information",
            "Microsoft": "Information"
        }
    },
    // Here is where you can supply custom configuration settings, Since it is JSON, everything is represented
    // as key: value pairs
    // Name of section is your choice
    "AppConfiguration": {
        "UserName": "UserName",
        "Password": "Password"
    }
}
```

Carregar esse arquivo em uma instância de `IConfiguration` dentro de seu aplicativo é feito em `Startup.cs`:

```
public Startup(IConfiguration configuration)
{
    Configuration = configuration;
}

public IConfiguration Configuration { get; }
```

O aplicativo lê de `Configuration` para obter as configurações:

```
string userName = Configuration.GetSection("AppConfiguration")["UserName"];
string password = Configuration.GetSection("AppConfiguration")["Password"];
```

Existem extensões para essa abordagem para tornar o processo mais robusto, tais como o uso de DI ([injeção de dependência](#)) para carregar um serviço com esses valores. A abordagem de DI fornece um conjunto fortemente tipado de objetos de configuração.

```
// Assume AppConfiguration is a class representing a strongly-typed version of AppConfiguration section
services.Configure<AppConfiguration>(Configuration.GetSection("AppConfiguration"));
```

#### NOTE

Para uma referência mais detalhada sobre configuração do ASP.NET Core, veja [Configuração no ASP.NET Core](#).

## Injeção de dependência nativa

Uma meta importante ao criar aplicativos escalonáveis e grandes é o acoplamento flexível de componentes e serviços. A [injeção de dependência](#) é uma técnica popular para conseguir isso e é também um componente nativo do ASP.NET Core.

Em aplicativos ASP.NET, os desenvolvedores contam com uma biblioteca de terceiros para implementar injeção de dependência. Um biblioteca desse tipo é a [Unity](#), fornecida pelas Diretrizes da Microsoft.

Um exemplo de configuração da injeção de dependência com Unity é a implementação de `IDependencyResolver`, que encapsula uma `UnityContainer`:

```

using Microsoft.Practices.Unity;
using System;
using System.Collections.Generic;
using System.Web.Http.Dependencies;

public class UnityResolver : IDependencyResolver
{
    protected IUnityContainer container;

    public UnityResolver(IUnityContainer container)
    {
        if (container == null)
        {
            throw new ArgumentNullException("container");
        }
        this.container = container;
    }

    public object GetService(Type serviceType)
    {
        try
        {
            return container.Resolve(serviceType);
        }
        catch (ResolutionFailedException)
        {
            return null;
        }
    }

    public IEnumerable<object> GetServices(Type serviceType)
    {
        try
        {
            return container.ResolveAll(serviceType);
        }
        catch (ResolutionFailedException)
        {
            return new List<object>();
        }
    }

    public IDependencyScope BeginScope()
    {
        var child = container.CreateChildContainer();
        return new UnityResolver(child);
    }

    public void Dispose()
    {
        Dispose(true);
    }

    protected virtual void Dispose(bool disposing)
    {
        container.Dispose();
    }
}

```

Crie uma instância de sua `UnityContainer`, registre seu serviço e defina o resolvedor de dependência de `HttpConfiguration` para a nova instância de `UnityResolver` para o contêiner:

```
public static void Register(HttpConfiguration config)
{
    var container = new UnityContainer();
    container.RegisterType<IProductRepository, ProductRepository>(new HierarchicalLifetimeManager());
    config.DependencyResolver = new UnityResolver(container);

    // Other Web API configuration not shown.
}
```

Injete `IProductRepository` quando necessário:

```
public class ProductsController : ApiController
{
    private IProductRepository _repository;

    public ProductsController(IProductRepository repository)
    {
        _repository = repository;
    }

    // Other controller methods not shown.
}
```

Já que a injeção de dependência é parte do ASP.NET Core, você pode adicionar o serviço no método `ConfigureServices` de `Startup.cs`:

```
public void ConfigureServices(IServiceCollection services)
{
    // Add application services.
    services.AddTransient<IProductRepository, ProductRepository>();
}
```

O repositório pode ser injetado em qualquer lugar, como ocorria com a Unity.

#### NOTE

Para obter mais informações sobre injeção de dependência, confira [Injeção de dependência](#).

## Fornecer arquivos estáticos

Uma parte importante do desenvolvimento da Web é a capacidade de servir ativos estáticos, do lado do cliente. Os exemplos mais comuns de arquivos estáticos são HTML, CSS, Javascript e imagens. Esses arquivos precisam ser salvos no local de publicação do aplicativo (ou CDN) e referenciados para que eles possam ser carregados por uma solicitação. Esse processo foi alterado no ASP.NET Core.

No ASP.NET, arquivos estáticos são armazenados em vários diretórios e referenciados nas exibições.

No ASP.NET Core, arquivos estáticos são armazenados na "raiz da Web" (`<raiz do conteúdo>/wwwroot`), a menos que configurado de outra forma. Os arquivos são carregados no pipeline de solicitação invocando o método de extensão `UseStaticFiles` de `Startup.Configure`:

```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles();
}
```

**NOTE**

Se você usar o .NET Framework como destino, instale o pacote NuGet `Microsoft.AspNetCore.StaticFiles`.

Por exemplo, um ativo de imagem na pasta `wwwroot/imagens` está acessível para o navegador em um local como

`http://<app>/images/<imageFileName>`.

**NOTE**

Para obter uma referência mais aprofundada sobre como servir arquivos estáticos no ASP.NET Core, veja [Arquivos estáticos](#).

## Recursos adicionais

- [Fazendo a portabilidade de Bibliotecas para o .NET Core](#)

# Migrar do ASP.NET MVC para ASP.NET Core MVC

10/11/2018 • 15 minutes to read • [Edit Online](#)

Por [Rick Anderson](#), [Daniel Roth](#), [Steve Smith](#), e [Scott Addie](#)

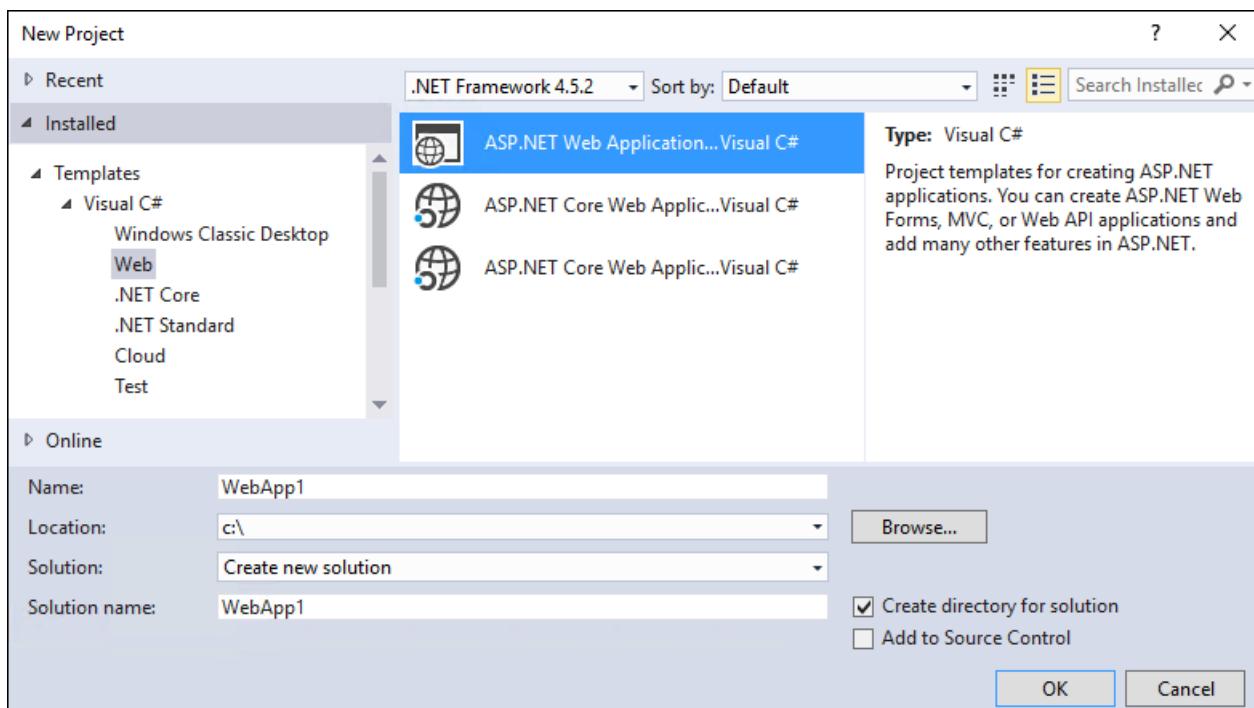
Este artigo mostra como começar a migrar um projeto ASP.NET MVC para [ASP.NET Core MVC](#). No processo, ele destaca muitas das coisas que mudaram do ASP.NET MVC. Migrando do ASP.NET MVC é um processo de várias etapas e este artigo aborda a configuração inicial, básicas controladores e modos de exibição, conteúdo estático e as dependências do lado do cliente. Artigos adicionais abrangem a migração de configuração e o código de identidade encontrados em muitos projetos do ASP.NET MVC.

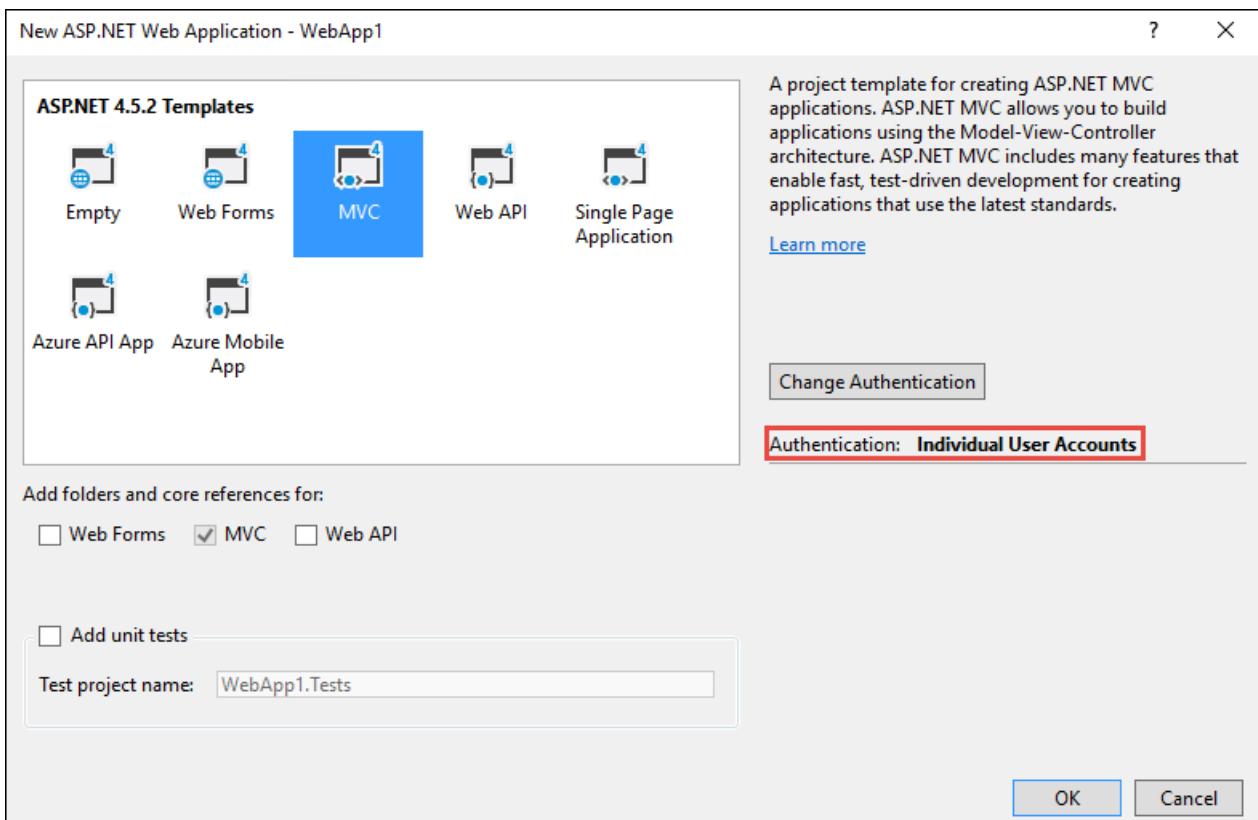
## NOTE

Os números de versão nos exemplos podem não ser atuais. Talvez você precise atualizar seus projetos de acordo.

## Criar o projeto do MVC do ASP.NET starter

Para demonstrar a atualização, vamos começar criando um aplicativo ASP.NET MVC. Criá-lo com o nome *WebApp1* para o namespace coincida com o projeto do ASP.NET Core que criamos na próxima etapa.

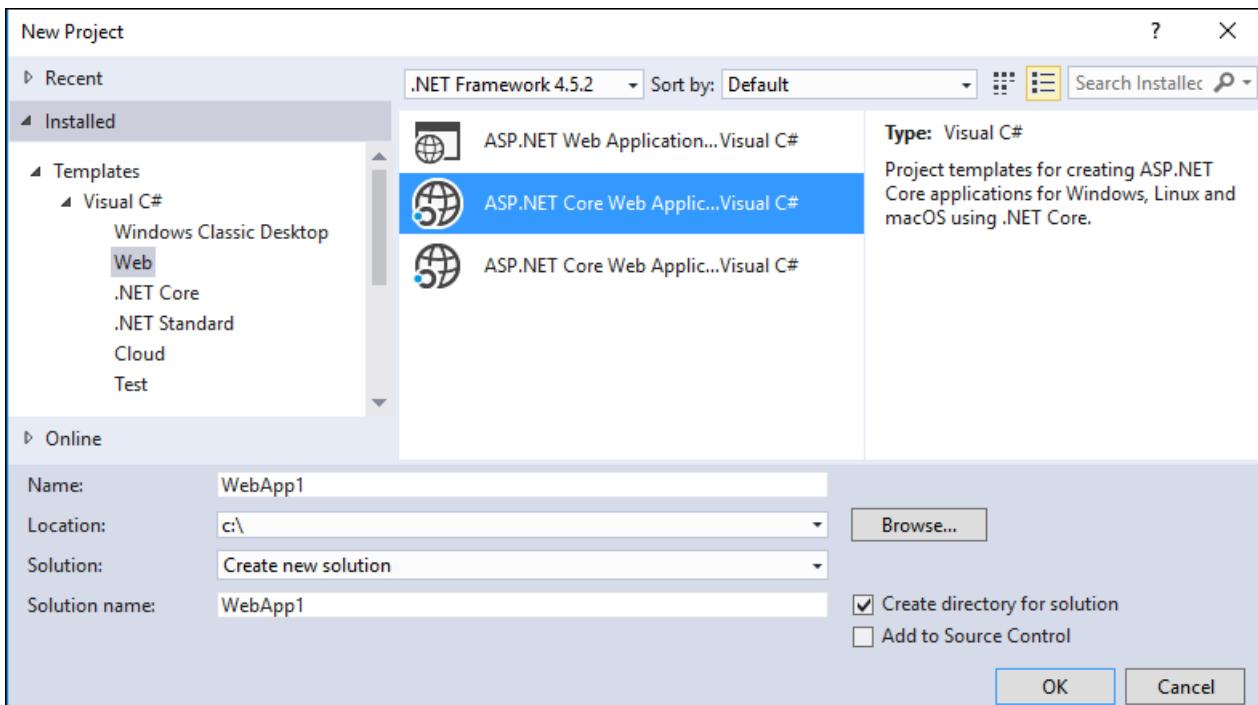


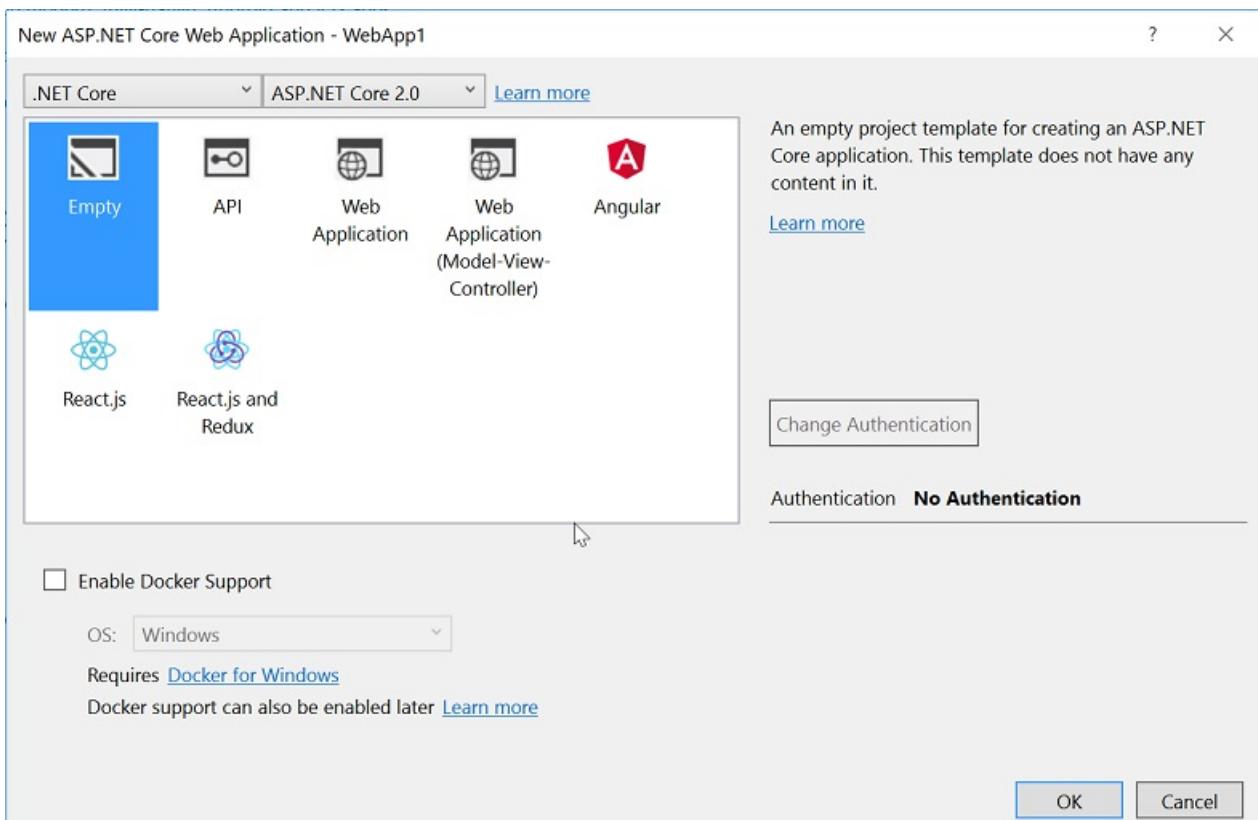


Opcional: alterar o nome da solução do *WebApp1* à *Mvc5*. O Visual Studio exibe o novo nome da solução (*Mvc5*), que torna mais fácil de informar deste projeto no próximo projeto.

## Criar o projeto do ASP.NET Core

Criar um novo *vazio* aplicativo de web do ASP.NET Core com o mesmo nome que o projeto anterior (*WebApp1*) para que os namespaces em dois projetos sejam correspondentes. Ter o mesmo namespace torna mais fácil copiar o código entre os dois projetos. Você precisará criar esse projeto em um diretório diferente do projeto anterior para usar o mesmo nome.





- *Opcional:* criar um novo aplicativo de ASP.NET Core usando o *aplicativo Web* modelo de projeto. Nomeie o projeto *WebApp1* e selecione uma opção de autenticação do **contas de usuário individuais**. Renomear este aplicativo seja *FullAspNetCore*. Isso poupa de projeto tempo criando a conversão. Você pode examinar o código gerado pelo modelo para ver o resultado final ou copiar o código para o projeto de conversão. Também é útil quando você ficar preso em uma etapa de conversão a ser comparado com o projeto de modelo gerado.

## Configurar o site para usar o MVC

- Ao direcionar o .NET Core, o [metapacote Microsoft](#) é referenciado por padrão. Este pacote contém pacotes de pacotes usados pelos aplicativos MVC. Se o destino do .NET Framework, referências de pacote devem estar listadas individualmente no arquivo de projeto.
- Ao direcionar o .NET Core, o [metapacote Microsoft.AspNetCore.All](#) é referenciado por padrão. Este pacote contém pacotes de pacotes usados pelos aplicativos MVC. Se o destino do .NET Framework, referências de pacote devem estar listadas individualmente no arquivo de projeto.
- Ao direcionar o .NET Core ou .NET Framework, pacotes de pacotes usados pelos aplicativos MVC estão listados individualmente no arquivo de projeto.

`Microsoft.AspNetCore.Mvc` é a estrutura do ASP.NET Core MVC. `Microsoft.AspNetCore.StaticFiles` é o manipulador de arquivo estático. O tempo de execução do ASP.NET Core é modular e você deve explicitamente optar por fornecer arquivos estáticos (consulte [arquivos estáticos](#)).

- Abra o *Startup.cs* de arquivo e altere o código para corresponder ao seguinte:

```

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;

namespace WebApp1
{
    public class Startup
    {
        // This method gets called by the runtime. Use this method to add services to the container.
        // For more information on how to configure your application, visit
        https://go.microsoft.com/fwlink/?LinkID=398940
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddMvc();
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request
        pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseStaticFiles();

            app.UseMvc(routes =>
            {
                routes.MapRoute(
                    name: "default",
                    template: "{controller=Home}/{action=Index}/{id?}");
            });
        }
    }
}

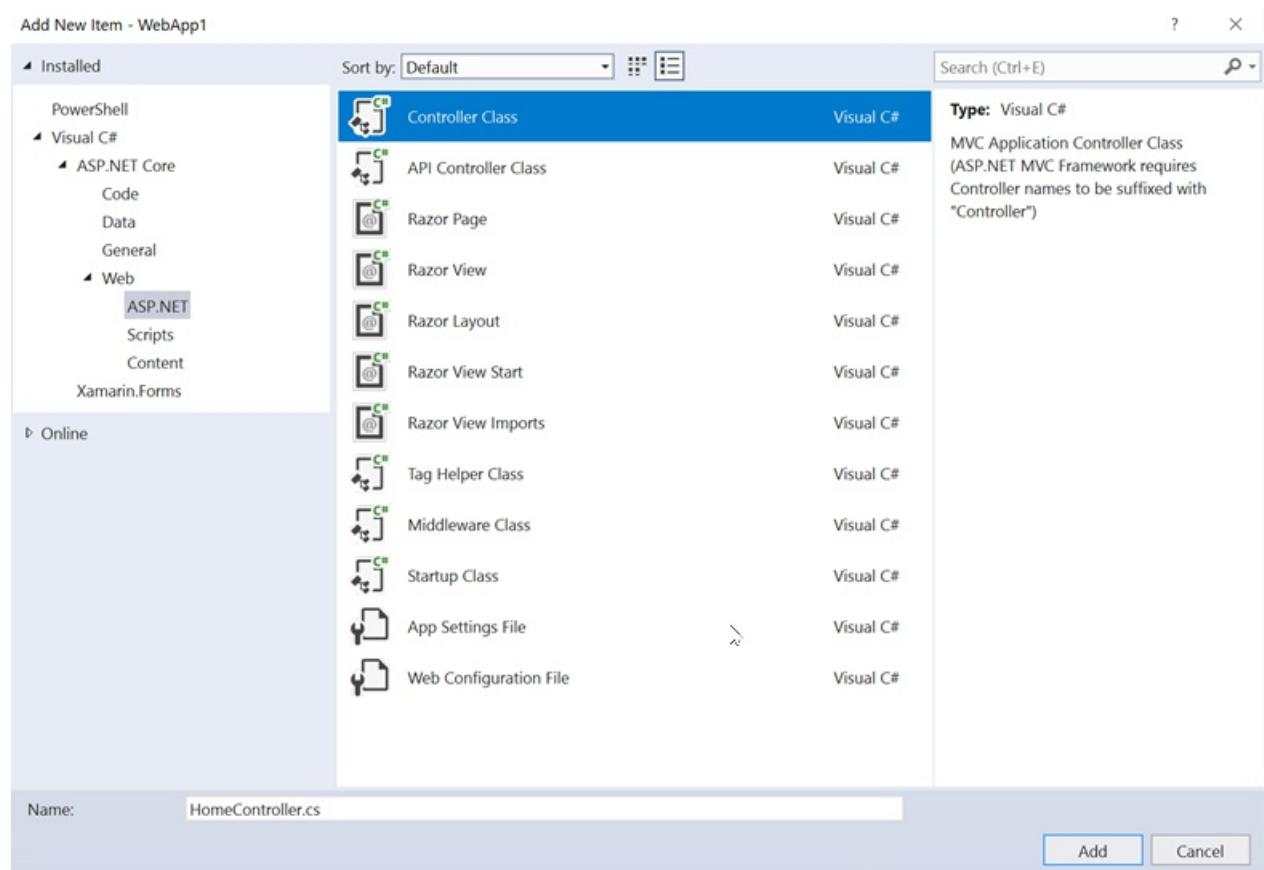
```

O `UseStaticFiles` método de extensão adiciona o manipulador de arquivo estático. Conforme mencionado anteriormente, o tempo de execução do ASP.NET é modular e você deve explicitamente optar por fornecer arquivos estáticos. O `UseMvc` adiciona o método de extensão roteamento. Para obter mais informações, consulte [inicialização do aplicativo e roteamento](#).

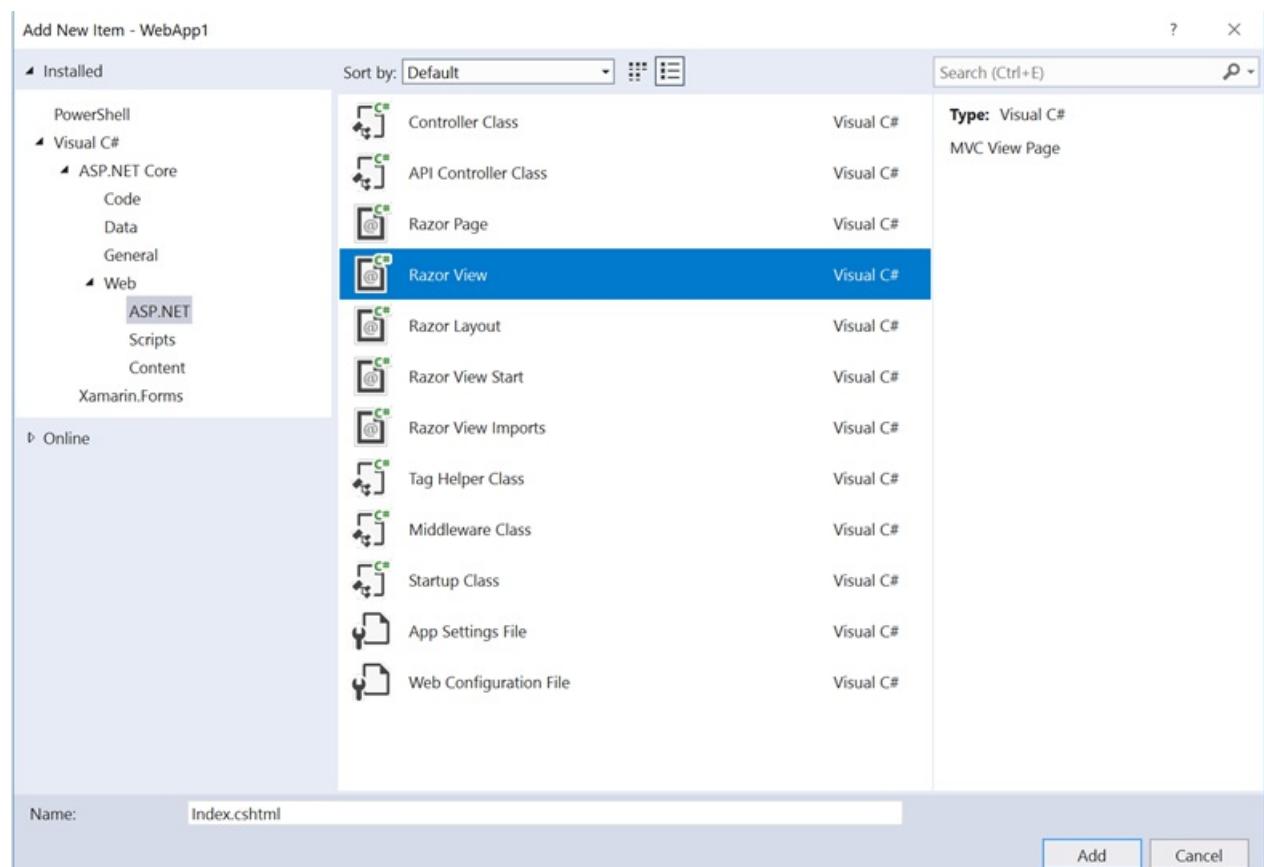
## Adicionar um controlador e o modo de exibição

Nesta seção, você adicionará um controlador de mínimo e o modo de exibição para servir como espaços reservados para o controlador do ASP.NET MVC e exibições que na próxima seção, você migrará.

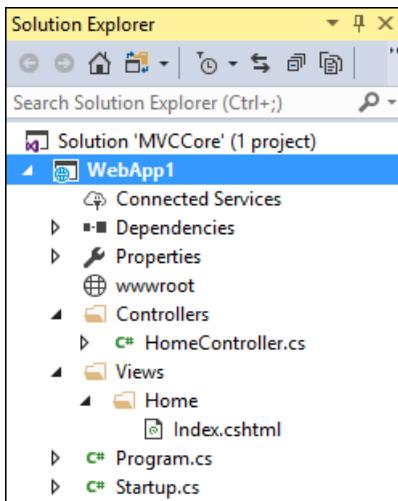
- Adicionar um *controladores* pasta.
- Adicionar um **classe Controller** denominado *HomeController.cs* para o *controladores* pasta.



- Adicionar um *modos de exibição* pasta.
- Adicionar um *exibições/inicial* pasta.
- Adicionar um **modo de exibição do Razor** denominado *index.cshtml* para o *exibições/inicial* pasta.



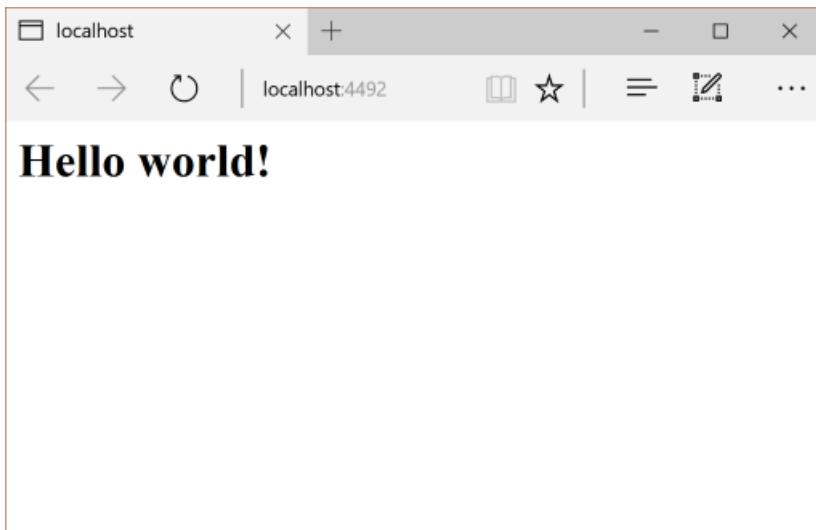
A estrutura do projeto é mostrada abaixo:



Substitua o conteúdo do *Views/Home/Index.cshtml* arquivo com o seguinte:

```
<h1>Hello world!</h1>
```

Execute o aplicativo.



Ver [controladores e modos de exibição](#) para obter mais informações.

Agora que temos um projeto de núcleo do ASP.NET mínimo do trabalho, podemos começar a migrar a funcionalidade do projeto ASP.NET MVC. É preciso mover o seguinte:

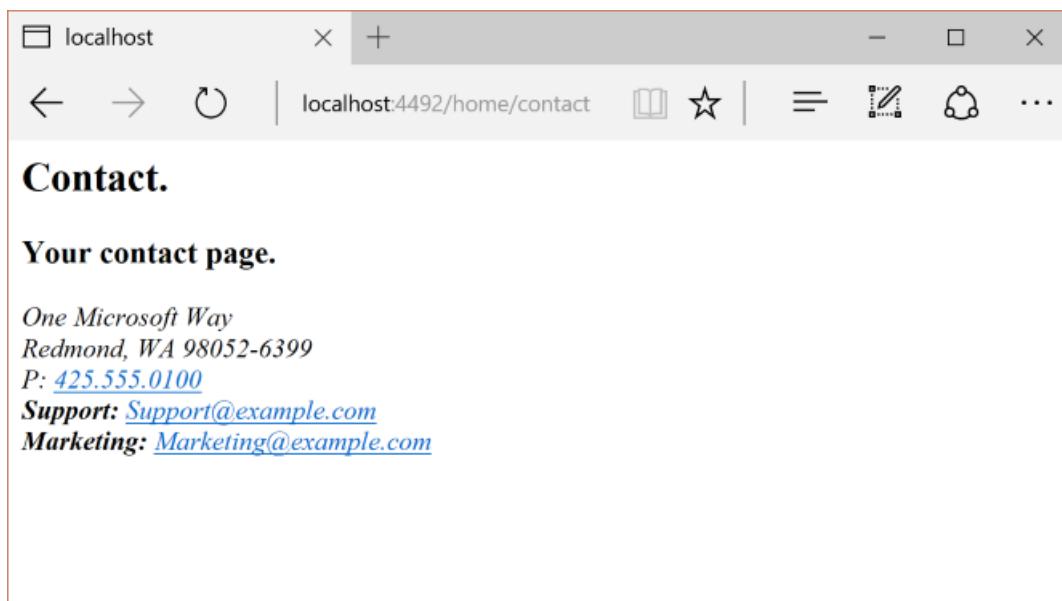
- conteúdo do lado do cliente (CSS, fontes e scripts)
- controladores
- modos de exibição
- modelos
- Agrupamento
- filtros
- Log de entrada/saída, a identidade (Isso é feito no próximo tutorial.)

## Controladores e exibições

- Copie cada um dos métodos do ASP.NET MVC `HomeController` para o novo `HomeController`. Observe que,

no ASP.NET MVC, tipo de método de ação retorno controlador interno do modelo é `ActionResult`; no ASP.NET Core MVC, os retorno de métodos de ação `IActionResult` em vez disso. `ActionResult` implementa `IActionResult`, portanto, não é necessário alterar o tipo de retorno de métodos de ação.

- Cópia de `About.cshtml`, `Contact.cshtml`, e `index.cshtml` arquivos de exibição do Razor do projeto ASP.NET MVC para o projeto ASP.NET Core.
- Execute o aplicativo ASP.NET Core e cada método de teste. Ainda não migramos o arquivo de layout ou estilos ainda, portanto, os modos de exibição renderizados contêm apenas o conteúdo nos arquivos de exibição. Você não terá os links de arquivo gerado de layout para o `About` e `Contact` modos de exibição, portanto, você precisará invocá-los a partir do navegador (substitua **4492** com o número da porta usado em seu projeto).
  - `http://localhost:4492/home/about`
  - `http://localhost:4492/home/contact`



Observe a falta de estilo e itens de menu. Corrigiremos isso na próxima seção.

## Conteúdo estático

Nas versões anteriores do ASP.NET MVC, o conteúdo estático foi hospedado da raiz do projeto da web e foi intercalado com arquivos do lado do servidor. No ASP.NET Core, conteúdo estático é hospedado na `wwwroot` pasta. Você desejará copiar o conteúdo estático de seu aplicativo ASP.NET MVC antigo para o `wwwroot` pasta em seu projeto ASP.NET Core. Nessa conversão de exemplo:

- Cópia de `/favicon.ico` arquivo de projeto MVC antigo para o `wwwroot` pasta no projeto do ASP.NET Core.

O ASP.NET MVC antigo projeto usa `Bootstrap` para seu estilo e armazena a inicialização de arquivos no `conteúdo` e `Scripts` pastas. O modelo, que gerou o antigo projeto do ASP.NET MVC, faz referência a inicialização no arquivo de layout (`Views/Shared/_Layout.cshtml`). Você pode copiar o `bootstrap.js` e `Bootstrap` arquivos do ASP.NET MVC de projeto para o `wwwroot` pasta no novo projeto. Em vez disso, vamos adicionar suporte para o Bootstrap (e outras bibliotecas do lado do cliente) usando as CDNs na próxima seção.

## Migrar o arquivo de layout

- Cópia de `viewstart` arquivo do antigo do projeto ASP.NET MVC `exibições` pasta para do projeto ASP.NET Core `modos de exibição` pasta. O `viewstart` arquivo não foi alterado no ASP.NET Core MVC.
- Criar uma `Views/Shared` pasta.

- *Opcional:* cópia `viewimports.cshtml` da `FullAspNetCore` do projeto MVC *exibições* pasta para do projeto ASP.NET Core *Modos de exibição* pasta. Remover qualquer declaração de namespace na `viewimports.cshtml` arquivo. O `viewimports.cshtml` fornece os namespaces para todos os arquivos de exibição de arquivo e traz **auxiliares de marca**. Os auxiliares de marca são usados no novo arquivo de layout. O `viewimports.cshtml` arquivo é novo para o ASP.NET Core.
- Cópia de `layout.cshtml` arquivo do antigo do projeto ASP.NET MVC *Views/Shared* pasta para do projeto ASP.NET Core *Views/Shared* pasta.

Abra `layout.cshtml` de arquivo e faça as seguintes alterações (o código completo é mostrado abaixo):

- Substitua `@Styles.Render("~/Content/css")` com um `<link>` elemento do qual carregar *Bootstrap* (veja abaixo).
- Remova `@Scripts.Render("~/bundles/modernizr")`.
- Comente a `@Html.Partial("_LoginPartial")` linha (envolvem a linha com `@*...*@`). Para obter mais informações, consulte [migrar autenticação e identidade para o ASP.NET Core](#)
- Substitua `@Scripts.Render("~/bundles/jquery")` com um `<script>` elemento (veja abaixo).
- Substitua `@Scripts.Render("~/bundles/bootstrap")` com um `<script>` elemento (veja abaixo).

A marcação de substituição para inclusão de CSS do Bootstrap:

```
<link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
      integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
      crossorigin="anonymous">
```

A marcação de substituição para o jQuery e Bootstrap JavaScript inclusão:

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"
      integrity="sha384-Tc5IQib027qvyjSMfHj0MaLkfWVxZxUPnPnCJA7l2mCwNIpG9mGCD8wGNiCpd7Txa"
      crossorigin="anonymous"></script>
```

Atualizada `layout.cshtml` arquivo é mostrado abaixo:

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET Application</title>
    <link rel="stylesheet"
        href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
        integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg320mUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
        crossorigin="anonymous">
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                @Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li>@Html.ActionLink("Home", "Index", "Home")</li>
                    <li>@Html.ActionLink("About", "About", "Home")</li>
                    <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
                </ul>
                @* @Html.Partial("_LoginPartial") *@
            </div>
        </div>
    </div>
    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
        </footer>
    </div>

    <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"
        integrity="sha384-Tc5IQib027qvyjSMFhJ0MaLkfWxZxUPnCJA712mCWNIpG9mGCD8wGNICPD7Txz"
        crossorigin="anonymous"></script>
    @RenderSection("scripts", required: false)
</body>
</html>

```

Exiba o site no navegador. Ele agora deve carregar corretamente, com os estilos esperados em vigor.

- *Opcional:* você talvez queira usar o novo arquivo de layout. Para este projeto, você pode copiar o arquivo de layout do *FullAspNetCore* projeto. O novo arquivo de layout utiliza [auxiliares de marca](#) e tem outras melhorias.

## Configurar o agrupamento e minificação

Para obter informações sobre como configurar o agrupamento e minificação, consulte [agrupamento e Minificação](#).

## Resolver erros HTTP 500

Há muitos problemas que podem causar uma mensagem de erro HTTP 500 que não contém informações sobre a origem do problema. Por exemplo, se o *viewimports* arquivo contém um namespace que não existe em seu

projeto, você obterá um erro HTTP 500. Por padrão em aplicativos ASP.NET Core, o `UseDeveloperExceptionPage` extensão é adicionada para o `IApplicationBuilder` e executados quando a configuração é *desenvolvimento*. Isso é detalhado no código a seguir:

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;

namespace WebApp1
{
    public class Startup
    {
        // This method gets called by the runtime. Use this method to add services to the container.
        // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddMvc();
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseStaticFiles();

            app.UseMvc(routes =>
            {
                routes.MapRoute(
                    name: "default",
                    template: "{controller=Home}/{action=Index}/{id?}");
            });
        }
    }
}
```

ASP.NET Core converte as exceções sem tratamento em um aplicativo web em respostas de erro HTTP 500. Normalmente, os detalhes do erro não estão incluídos nessas respostas para evitar a divulgação de informações possivelmente confidenciais sobre o servidor. Ver **usando a página de exceção do desenvolvedor** na [tratar erros](#) para obter mais informações.

## Recursos adicionais

- [Desenvolvimento do lado do cliente](#)
- [Auxiliares de marcação](#)

# Migrar da API Web ASP.NET para ASP.NET Core

11/12/2018 • 13 minutes to read • [Edit Online](#)

Por [Scott Addie](#) e [Steve Smith](#)

Uma API da Web do ASP.NET 4.x é um serviço HTTP que atinja uma ampla gama de clientes, incluindo navegadores e dispositivos móveis. Unifica o ASP.NET Core MVC do ASP.NET do 4.x e modelos de aplicativo de API da Web em um modelo de programação mais simples conhecido como o ASP.NET Core MVC. Este artigo demonstra as etapas necessárias para migrar da API de Web do ASP.NET 4.x para ASP.NET Core MVC.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Pré-requisitos

- [Visual Studio 2017 versão 15.9 ou posterior com a carga de trabalho ASP.NET e desenvolvimento para a Web](#)
- [SDK 2.2 ou posterior do .NET Core](#)

## Examine o projeto de API da Web do ASP.NET 4.x

Como ponto de partida, este artigo usa o *ProductsApp* projeto criado no [Introdução à API Web ASP.NET 2](#). Nesse projeto, um projeto de API Web simples ASP.NET 4.x está configurado da seguinte maneira.

Na *Global.asax.cs*, é feita uma chamada para `WebApiConfig.Register` :

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

namespace ProductsApp
{
    public class WebApiApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            GlobalConfiguration.Configure(WebApiConfig.Register);
        }
    }
}
```

O `WebApiConfig` classe é encontrada na *App\_Start* pasta e tem um estático `Register` método:

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

namespace ProductsApp
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            // Web API configuration and services

            // Web API routes
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}
```

Essa classe configura [roteamento de atributo](#), embora na verdade não está sendo usado no projeto. Ele também configura a tabela de roteamento, que é usada pela API Web ASP.NET. Nesse caso, a API da Web do ASP.NET 4.x espera que as URLs para corresponder ao formato `/api/{controller}/{id}`, com `{id}` opcionais.

O `ProductsApp` projeto inclui um controlador. O controlador herda de `ApiController` e contém duas ações:

```

using ProductsApp.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Web.Http;

namespace ProductsApp.Controllers
{
    public class ProductsController : ApiController
    {
        Product[] products = new Product[]
        {
            new Product
            {
                Id = 1, Name = "Tomato Soup", Category = "Groceries", Price = 1
            },
            new Product
            {
                Id = 2, Name = "Yo-yo", Category = "Toys", Price = 3.75M
            },
            new Product
            {
                Id = 3, Name = "Hammer", Category = "Hardware", Price = 16.99M
            }
        };

        public IEnumerable<Product> GetAllProducts()
        {
            return products;
        }

        public IHttpActionResult GetProduct(int id)
        {
            var product = products.FirstOrDefault((p) => p.Id == id);
            if (product == null)
            {
                return NotFound();
            }
            return Ok(product);
        }
    }
}

```

O `Product` modelo usado pelo `ProductsController` é uma classe simples:

```

namespace ProductsApp.Models
{
    public class Product
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Category { get; set; }
        public decimal Price { get; set; }
    }
}

```

As seções a seguir demonstram a migração do projeto API da Web para ASP.NET Core MVC.

## Criar o projeto de destino

Conclua as seguintes etapas no Visual Studio:

- Vá para **arquivo > nova > projeto > outros tipos de projeto > Soluções do visual Studio**. Selecione

**solução em branco** o nome da solução *WebAPIMigration*. Clique o **Okey** botão.

- Adicionar existente *ProductsApp* projeto à solução.
- Adicione um novo **aplicativo Web ASP.NET Core** projeto à solução. Selecione o **.NET Core** estrutura na lista suspensa de destino e, em seguida, selecione o **API** modelo de projeto. Nomeie o projeto *ProductsCore* e clique no **Okey** botão.

Agora, a solução contém dois projetos. As seções a seguir explicam a migrar os *ProductsApp* o conteúdo do projeto para o *ProductsCore* projeto.

## Migrar configuração

ASP.NET Core não usa o *App\_Start* pasta ou o *global.asax* arquivo e o *Web.config* arquivo for adicionado no momento da publicação. *Startup.cs* é o substituto do *global.asax* e está localizado na raiz do projeto. O **Startup** classe manipula todas as tarefas de inicialização do aplicativo. Para obter mais informações, consulte [Inicialização de aplicativo no ASP.NET Core](#).

No ASP.NET Core MVC, o roteamento de atributo é incluído por padrão quando **UseMvc** é chamado no **Startup.Configure**. O seguinte **UseMvc** chamar substitui o *ProductsApp* do projeto *app\_start* arquivo:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseMvc();
}
```

## Migrar modelos e controladores

Copiar sobre a *ProductApp* controlador do projeto e o modelo utilizado. Siga estas etapas:

1. Cópia *Controllers/ProductsController.cs* do projeto original para o novo.
2. Copiar todo o *modelos* pasta do projeto original para o novo.
3. Alterar os namespaces dos arquivos copiados para coincidir com o novo nome do projeto (*ProductsCore*). Ajustar a **using ProductsApp.Models;** instrução no *ProductsController.cs* muito.

Neste ponto, criando os resultados do aplicativo em um número de erros de compilação. Os erros ocorrem porque os seguintes componentes não existem no ASP.NET Core:

- Classe **ApiController**
- Namespace **System.Web.Http**
- **IHttpActionResult** Interface

Corrija os erros da seguinte maneira:

1. Alteração **ApiController** para  **ControllerBase**. Adicione **using Microsoft.AspNetCore.Mvc;** para resolver o  **ControllerBase** referência.
2. Excluir **using System.Web.Http;** .
3. Alterar o **GetProduct** tipo de retorno da ação de **IHttpActionResult** para **ActionResult<Product>** .

Simplificar a `GetProduct` da ação `return` instrução para o seguinte:

```
return product;
```

## Configurar o roteamento

Configure o roteamento da seguinte maneira:

1. Decore o `ProductsController` classe com os seguintes atributos:

```
[Route("api/[controller]")]
[ApiController]
```

Precedente `[Route]` configura o atributo padrão de roteamento de atributo do controlador. O `[ApiController]` atributo faz o roteamento de um requisito para todas as ações nesse controlador de atributo.

Roteamento de atributo oferece suporte a tokens, como `[controller]` e `[action]`. Em tempo de execução, cada token é substituído com o nome do controlador ou ação, respectivamente, para o qual o atributo foi aplicado. Os tokens de reduzem o número de cadeias de caracteres mágicas no projeto. Os tokens de também garantem rotas permanecerem sincronizadas com os correspondentes controladores e ações quando automático renomear refatorações são aplicadas.

2. Defina o modo de compatibilidade do projeto para o ASP.NET Core 2.2:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc()
        .SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
}
```

A alteração anterior:

- É necessário para usar o `[ApiController]` atributo no nível do controlador.
- Aceita o potencialmente significativa comportamentos introduzidos no ASP.NET Core 2.2.

3. Habilitar as solicitações HTTP Get para o `ProductController` ações:

- Aplicar a `[HttpGet]` atributo para o  `GetAllProducts` ação.
- Aplicar a `[HttpGet("{id}")]` de atributo para o `GetProduct` ação.

Após as alterações anteriores e a remoção do não utilizado `using` instruções `ProductsController.cs` arquivo tem esta aparência:

```

using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using ProductsCore.Models;

namespace ProductsCore.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ProductsController : ControllerBase
    {
        Product[] products = new Product[]
        {
            new Product
            {
                Id = 1, Name = "Tomato Soup", Category = "Groceries", Price = 1
            },
            new Product
            {
                Id = 2, Name = "Yo-yo", Category = "Toys", Price = 3.75M
            },
            new Product
            {
                Id = 3, Name = "Hammer", Category = "Hardware", Price = 16.99M
            }
        };

        [HttpGet]
        public IEnumerable<Product> GetAllProducts()
        {
            return products;
        }

        [HttpGet("{id}")]
        public ActionResult<Product> GetProduct(int id)
        {
            var product = products.FirstOrDefault((p) => p.Id == id);
            if (product == null)
            {
                return NotFound();
            }
            return product;
        }
    }
}

```

Execute o projeto migrado e navegue até `/api/products`. É exibida uma lista completa dos três produtos. Navegue para `/api/products/1`. O primeiro produto será exibida.

## Correção de compatibilidade

O [Microsoft.AspNetCore.Mvc.WebApiCompatShim](#) biblioteca fornece um shim de compatibilidade para mover projetos de API da Web do ASP.NET 4.x ASP.NET Core. O shim de compatibilidade se estende do ASP.NET Core para dar suporte a uma série de convenções da API de Web do ASP.NET 4.x 2. O exemplo portado anteriormente neste documento é bastante básico que o shim de compatibilidade era desnecessário. Para projetos maiores, usem o shim de compatibilidade pode ser útil para temporariamente preenchendo a lacuna de API entre o ASP.NET Core e ASP.NET 4.x Web API 2.

O shim de compatibilidade de API da Web destina-se a ser usado como uma medida temporária para dar suporte à migração grande API da Web projetos do ASP.NET 4.x ASP.NET Core. Ao longo do tempo, os projetos devem ser atualizados para usar os padrões do ASP.NET Core em vez de usar o shim de compatibilidade.

Recursos de compatibilidade incluídos no `Microsoft.AspNetCore.Mvc.WebApiCompatShim` incluem:

- Adiciona um `ApiController` tipo de forma que os tipos de base dos controladores não precisam ser atualizados.
- Habilita a associação de modelo de estilo de API da Web. Funções de associação de maneira semelhante do ASP.NET de modelo do ASP.NET Core MVC 4.x/MVC 5, por padrão. As alterações de correção de compatibilidade associação de modelo para ser mais semelhante a convenções de associação de modelo do ASP.NET Web API 2 de 4.x. Por exemplo, tipos complexos são vinculados automaticamente do corpo da solicitação.
- Estende a associação de modelo para que as ações do controlador podem usar parâmetros de tipo `HttpRequestMessage`.
- Adiciona os formatadores de mensagem, permitindo que as ações para retornar os resultados do tipo `HttpResponseMessage`.
- Adiciona métodos de resposta adicionais que ações de API Web 2 podem ter usado para servir as respostas:
  - `HttpResponseMessage` geradores de:
    - `CreateResponse<T>`
    - `CreateErrorResponse`
  - Métodos de ação de resultados:
    - `BadRequestErrorMessageResult`
    - `ExceptionResult`
    - `InternalServerErrorResult`
    - `InvalidModelStateResult`
    - `NegotiatedContentResult`
    - `ResponseMessageResult`
- Adiciona uma instância do `IContentNegotiator` para o aplicativo disponibiliza os tipos de conteúdo relacionado a negociação do e do contêiner de DI (injeção) de dependência `Microsoft.AspNet.WebApi.Client`. Exemplos de tais tipos `DefaultContentNegotiator` e `MediaTypeFormatter`.

Para usar o shim de compatibilidade:

1. Instalar o `Microsoft.AspNetCore.Mvc.WebApiCompatShim` pacote do NuGet.
2. Registrar os serviços do shim de compatibilidade com o contêiner de injeção de dependência do aplicativo chamando `services.AddMvc().AddWebApiConventions()` em `Startup.ConfigureServices`.
3. Definir web específicos de API roteia usando `MapWebApiRoute` sobre o `IRouteBuilder` no aplicativo do `IApplicationBuilder.UseMvc` chamar.

## Recursos adicionais

- [Criar APIs Web com o ASP.NET Core](#)
- [Tipos de retorno de ação do controlador na API Web ASP.NET Core](#)
- [Versão de compatibilidade do ASP.NET Core MVC](#)

# Migrar a configuração para o ASP.NET Core

30/10/2018 • 4 minutes to read • [Edit Online](#)

Por [Steve Smith](#) e [Scott Addie](#)

No artigo anterior, começamos [migrar um projeto ASP.NET MVC para ASP.NET Core MVC](#). Neste artigo, vamos migrar a configuração.

[Exibir ou baixar código de exemplo \(como baixar\)](#)

## Configuração da instalação

O ASP.NET Core não usa o *global.asax* e *Web.config* arquivos utilizadas de versões anteriores do ASP.NET. Em versões anteriores do ASP.NET, a lógica de inicialização do aplicativo foi colocada em uma `Application_StartUp` método dentro *global.asax*. Posteriormente, no ASP.NET MVC, uma *Startup.cs* arquivo foi incluído na raiz do projeto; e ele foi chamado quando o aplicativo foi iniciado. ASP.NET Core tem adotado essa abordagem completamente colocando toda lógica de inicialização na *Startup.cs* arquivo.

O *Web.config* arquivo também foi substituído no ASP.NET Core. Configuração em si agora pode ser configurada como parte do procedimento de inicialização de aplicativo descrito em *Startup.cs*. Configuração ainda pode utilizar arquivos XML, mas normalmente projetos do ASP.NET Core serão coloca valores de configuração em um arquivo formatado em JSON, como *appSettings.JSON*. Sistema de configuração do ASP.NET Core também poderá acessar facilmente as variáveis de ambiente, que podem fornecer um [mais local seguro e robusto](#) para valores específicos do ambiente. Isso é especialmente verdadeiro para segredos, como cadeias de caracteres de conexão e as chaves de API que não devem ser verificadas no controle de origem. Ver [configuração](#) para saber mais sobre a configuração no ASP.NET Core.

Neste artigo, estamos começando com o projeto ASP.NET Core migrado parcialmente desde [o artigo anterior](#). A configuração da instalação, adicione o seguinte construtor e propriedade como o *Startup.cs* arquivo localizado na raiz do projeto:

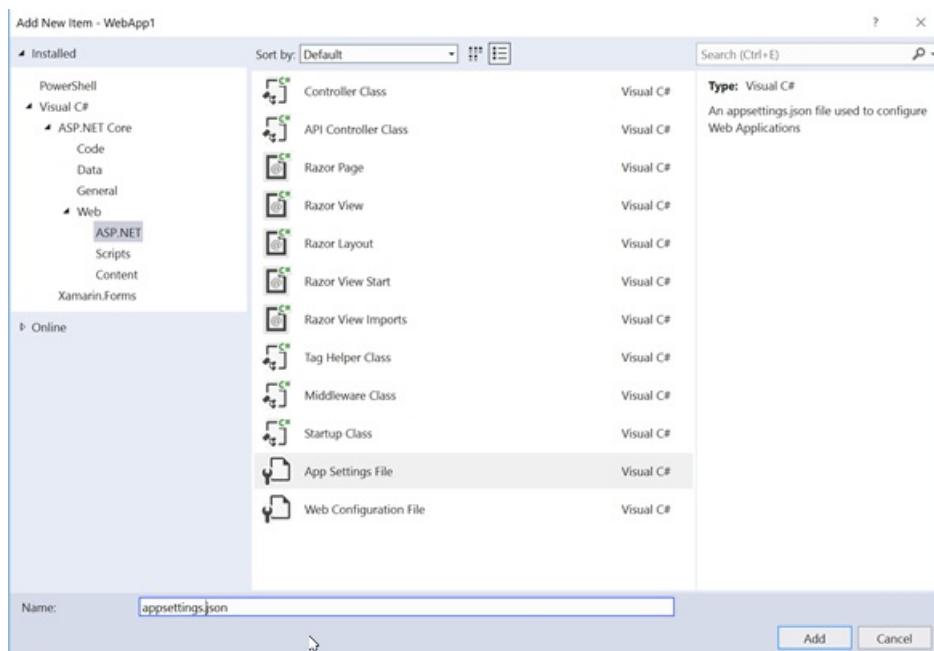
```
public Startup(IConfiguration configuration)
{
    Configuration = configuration;
}

public IConfiguration Configuration { get; }
```

Observe que neste ponto, o *Startup.cs* arquivo não será compilado, pois precisamos adicionar o seguinte `using` instrução:

```
using Microsoft.Extensions.Configuration;
```

Adicionar um *appSettings.JSON* arquivo na raiz do projeto usando o modelo de item apropriado:



## Migrar definições da configuração da Web. config

Nosso projeto ASP.NET MVC incluído a cadeia de caracteres de conexão de banco de dados necessários no *Web.config*, no `<connectionStrings>` elemento. Em nosso projeto do ASP.NET Core, vamos armazenar essas informações na *appSettings.JSON* arquivo. Abra *appSettings.JSON* e observe que ele já inclui o seguinte:

```
{
  "Data": {
    "DefaultConnection": {
      "ConnectionString": "Server=(localdb)\\MSSQLLocalDB;Database=_CHANGE_ME;Trusted_Connection=True;"
    }
  }
}
```

Na linha realçada descrita acima, altere o nome do banco de dados **\_CHANGE\_ME** para o nome do seu banco de dados.

## Resumo

ASP.NET Core coloca toda lógica de inicialização para o aplicativo em um único arquivo, em que os serviços necessários e dependências podem ser definidas e configuradas. Ele substitui o *Web.config* arquivo com um recurso de configuração flexíveis que pode aproveitar uma variedade de formatos de arquivo, como JSON, bem como as variáveis de ambiente.

# Migrar autenticação e identidade para o ASP.NET Core

13/02/2019 • 5 minutes to read • [Edit Online](#)

Por [Steve Smith](#)

No artigo anterior, podemos [migrados de configuração de um projeto ASP.NET MVC para ASP.NET Core MVC](#). Neste artigo, vamos migrar os recursos de gerenciamento de registro, logon e usuário.

## Configurar identidade e associação

No ASP.NET MVC, os recursos de autenticação e identidade são configurados usando o ASP.NET Identity no *Startup.Auth.cs* e *IdentityConfig.cs*, localizado na *App\_Start* pasta. No ASP.NET Core MVC, esses recursos são configurados na *Startup.cs*.

Instalar o `Microsoft.AspNetCore.Identity.EntityFrameworkCore` e `Microsoft.AspNetCore.Authentication.Cookies` pacotes do NuGet.

Em seguida, abra *Startup.cs* e atualize o `Startup.ConfigureServices` método para usar os serviços de identidade e o Entity Framework:

```
public void ConfigureServices(IServiceCollection services)
{
    // Add EF services to the services container.
    services.AddDbContext<ApplicationContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationContext>()
        .AddDefaultTokenProviders();

    services.AddMvc();
}
```

Neste ponto, há dois tipos referenciados no código acima que nós ainda não tiver migrado do projeto ASP.NET MVC: `ApplicationContext` e  `ApplicationUser`. Criar um novo *modelos* pasta no ASP.NET Core de projeto e adicionar duas classes a ele correspondente a esses tipos. Você encontrará o ASP.NET MVC versões dessas classes em */Models/IdentityModels.cs*, mas usaremos um arquivo por classe no projeto migrado, já que é mais clara.

*ApplicationUser.cs*:

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

namespace NewMvcProject.Models
{
    public class ApplicationUser : IdentityUser
    {
    }
}
```

*ApplicationContext.cs*:

```

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace NewMvcProject.Models
{
    public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {}

        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);
            // Customize the ASP.NET Core Identity model and override the defaults if needed.
            // For example, you can rename the ASP.NET Core Identity table names and more.
            // Add your customizations after calling base.OnModelCreating(builder);
        }
    }
}

```

O projeto Web do ASP.NET Core MVC Starter não inclui muita personalização de usuários, ou o `ApplicationDbContext`. Ao migrar um aplicativo real, você também precisará migrar todas as propriedades personalizadas e métodos de usuário do seu aplicativo e `DbContext` classes, bem como outras classes de modelo utilizadas no seu aplicativo. Por exemplo, se sua `DbContext` tem uma `DbSet<Album>`, você precisa migrar o `Album` classe.

Com esses arquivos em vigor, o `Startup.cs` arquivo pode ser feito para compilar atualizando seu `using` instruções:

```

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Hosting;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;

```

Nosso aplicativo agora está pronto para dar suporte a serviços de autenticação e identidade. Ele precisa apenas ter esses recursos expostos aos usuários.

## Migrar a lógica de registro e login

Com os serviços de identidade configurados para o aplicativo e o acesso a dados configurado usando o Entity Framework e o SQL Server, estamos prontos para adicionar suporte para registro e logon para o aplicativo. Lembre-se de que [anterior no processo de migração](#) estamos comentado uma referência a `loginpartial` na `layout.cshtml`. Agora é hora de retornar a esse código, remova os comentários e adicione os controladores de necessárias e os modos de exibição para dar suporte à funcionalidade de logon.

Remova os `@Html.Partial` de linha no `layout.cshtml`:

```

<li>@Html.ActionLink("Contact", "Contact", "Home")</li>
</ul>
@* @Html.Partial("_LoginPartial") *@
</div>
</div>

```

Agora, adicione uma nova exibição do Razor chamada `loginpartial` para o `Views/Shared` pasta:

Atualização `loginpartial` com o código a seguir (substitua todo seu conteúdo):

```
@inject SignInManager< ApplicationUser > SignInManager  
@inject UserManager< ApplicationUser > UserManager  
  
@if (SignInManager.IsSignedIn(User))  
{  
    <form asp-area="" asp-controller="Account" asp-action="Logout" method="post" id="logoutForm"  
    class="navbar-right">  
        <ul class="nav navbar-nav navbar-right">  
            <li>  
                <a asp-area="" asp-controller="Manage" asp-action="Index" title="Manage">Hello  
                @UserManager.GetUserName(User)!</a>  
            </li>  
            <li>  
                <button type="submit" class="btn btn-link navbar-btn navbar-link">Log out</button>  
            </li>  
        </ul>  
    </form>  
}  
else  
{  
    <ul class="nav navbar-nav navbar-right">  
        <li><a asp-area="" asp-controller="Account" asp-action="Register">Register</a></li>  
        <li><a asp-area="" asp-controller="Account" asp-action="Login">Log in</a></li>  
    </ul>  
}
```

Neste ponto, você deve ser capaz de atualizar o site no seu navegador.

## Resumo

ASP.NET Core apresenta alterações aos recursos de identidade do ASP.NET. Neste artigo, você viu como migrar os recursos de gerenciamento de usuário e autenticação de identidade do ASP.NET para ASP.NET Core.

# Migrar de ClaimsPrincipal.Current

23/08/2018 • 5 minutes to read • [Edit Online](#)

Em projetos do ASP.NET 4.x, era comum usar `ClaimsPrincipal.Current` recuperar atual autenticado a identidade do usuário e as declarações. No ASP.NET Core, essa propriedade não está definida. Código dependia ele precisa ser atualizado para obter a identidade do usuário autenticado atual por meio de um modo diferente.

## Dados específicos ao contexto em vez de dados estáticos

Ao usar o ASP.NET Core, os valores de ambos `ClaimsPrincipal.Current` e `Thread.CurrentPrincipal` não estão definidas. Essas propriedades representam o estado estático, que normalmente evita o ASP.NET Core. Em vez disso, arquitetura do ASP.NET Core é recuperar dependências (como a atual identidade do usuário) de coleções de contexto específico de serviço (usando seu [injeção de dependência](#) modelo (DI)). Além disso, `Thread.CurrentPrincipal` é o thread estático, portanto, ele não pode persistir as alterações em alguns cenários assíncronos (e `ClaimsPrincipal.Current` apenas chama `Thread.CurrentPrincipal` por padrão).

Para entender as classificações do thread de problemas de membros estáticos podem levar a cenários assíncronos, considere o seguinte trecho de código:

```
// Create a ClaimsPrincipal and set Thread.CurrentPrincipal
var identity = new ClaimsIdentity();
identity.AddClaim(new Claim(ClaimTypes.Name, "User1"));
Thread.CurrentPrincipal = new ClaimsPrincipal(identity);

// Check the current user
Console.WriteLine($"Current user: {Thread.CurrentPrincipal?.Identity.Name}");

// For the method to complete asynchronously
await Task.Yield();

// Check the current user after
Console.WriteLine($"Current user: {Thread.CurrentPrincipal?.Identity.Name}");
```

O código do exemplo anterior define `Thread.CurrentPrincipal` e verifica seu valor antes e depois de aguardar uma chamada assíncrona. `Thread.CurrentPrincipal` é específico para o *thread* no qual ele é definido e o método provavelmente retomar a execução em um thread diferente após o `await`. Consequentemente, `Thread.CurrentPrincipal` está presente quando ele é verificado pela primeira vez, mas é nulo após a chamada para `await Task.Yield()`.

Obtendo a atual identidade do usuário da coleção de serviço de injeção de dependência do aplicativo é mais testável, também, desde que as identidades de teste podem ser injetadas com facilidade.

## Recuperar o usuário atual em um aplicativo ASP.NET Core

Há várias opções para recuperar o atual usuário autenticado `ClaimsPrincipal` no ASP.NET Core no lugar de `ClaimsPrincipal.Current`:

- **ControllerBase.User**. Controladores do MVC podem acessar o usuário atual autenticado com suas [usuário](#) propriedade.
- **HttpContext**. Componentes com acesso ao atual `HttpContext` (por exemplo, middleware) pode obter o usuário atual `ClaimsPrincipal` partir `HttpContext`.

- **Passada pelo chamador.** Bibliotecas sem acesso ao atual `HttpContext` muitas vezes são chamados de controladores ou componentes de middleware e pode ter a atual identidade do usuário passada como um argumento.
- **IHttpContextAccessor.** O projeto que está sendo migrado para o ASP.NET Core pode ser muito grande para passar facilmente a atual identidade do usuário para todos os locais necessários. Nesses casos, `IHttpContextAccessor` pode ser usado como uma solução alternativa. `IHttpContextAccessor` é capaz de acessar atual `HttpContext` (se houver). Uma solução de curto prazo para obter a identidade do usuário atual no código que ainda não foi atualizado para trabalhar com a arquitetura de orientado a DI do ASP.NET Core seria:

- Tornar `IHttpContextAccessor` disponíveis no contêiner de injeção de dependência chamando `AddHttpContextAccessor` em `Startup.ConfigureServices`.
- Obtenha uma instância de `IHttpContextAccessor` durante a inicialização e armazená-lo em uma variável estática. A instância é disponibilizada para o código que anteriormente estava recuperando o usuário atual de uma propriedade estática.
- Recuperar o usuário atual `ClaimsPrincipal` usando `HttpContextAccessor.HttpContext?.User`. Se esse código é usado fora do contexto de uma solicitação HTTP, o `HttpContext` é nulo.

O final de opção, usando `IHttpContextAccessor`, infringir os princípios do ASP.NET Core (preferindo dependências injetadas dependências estáticas). Planeje, eventualmente, remova a dependência estático `IHttpContextAccessor` auxiliar. Pode ser uma ponte útil, no entanto, quando a migração de grandes aplicativos ASP.NET existentes que estavam usando previamente `ClaimsPrincipal.Current`.

# Migrar de autenticação de associação do ASP.NET para a identidade do ASP.NET Core 2.0

11/01/2019 • 11 minutes to read • [Edit Online](#)

Por [Isaac Levin](#)

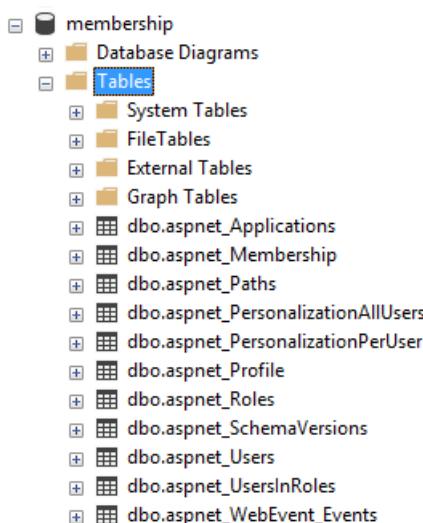
Este artigo demonstra como migrar o esquema de banco de dados para aplicativos do ASP.NET usando a autenticação de associação para a identidade do ASP.NET Core 2.0.

## NOTE

Este documento fornece as etapas necessárias para migrar o esquema de banco de dados para aplicativos baseados em associação do ASP.NET para o esquema de banco de dados usado para a identidade do ASP.NET Core. Para obter mais informações sobre como migrar da autenticação baseada em associação do ASP.NET para ASP.NET Identity, consulte [migrar um aplicativo existente da associação do SQL para o ASP.NET Identity](#). Para obter mais informações sobre a identidade do ASP.NET Core, consulte [Introdução à identidade no ASP.NET Core](#).

## Revisão do esquema de associação

Antes do ASP.NET 2.0, os desenvolvedores eram encarregados de criar todo o processo de autenticação e autorização para seus aplicativos. Com o ASP.NET 2.0, associação foi introduzida, fornecendo uma solução de texto clichê para lidar com segurança em aplicativos ASP.NET. Os desenvolvedores tiveram acesso agora inicializar um esquema em um banco de dados do SQL Server com o [aspnet\\_regsql.exe](#) comando. Depois de executar esse comando, as tabelas a seguir foram criadas no banco de dados.



Para migrar aplicativos existentes para a identidade do ASP.NET Core 2.0, os dados nessas tabelas precisam ser migrados para as tabelas usadas pelo novo esquema de identidade.

## Esquema do ASP.NET Core 2.0 de identidade

O ASP.NET Core 2.0 segue o [identidade](#) princípio introduzido no ASP.NET 4.5. Embora o princípio é compartilhado, a implementação entre as estruturas é diferente, até mesmo entre as versões do ASP.NET Core (consulte [migrar autenticação e identidade para o ASP.NET Core 2.0](#)).

É a maneira mais rápida para exibir o esquema para a identidade do ASP.NET Core 2.0 criar um novo aplicativo

ASP.NET Core 2.0. Siga estas etapas no Visual Studio 2017:

1. Selecione **Arquivo > Novo > Projeto**.
2. Criar um novo **aplicativo Web ASP.NET Core** projeto chamado `CoreIdentitySample`.
3. Selecione **ASP.NET Core 2.0** na lista suspensa e selecione **aplicativo Web**. Esse modelo produz um [páginas do Razor](#) aplicativo. Antes de clicar em **Okey**, clique em **alterar autenticação**.
4. Escolher **contas de usuário individuais** para os modelos de identidade. Por fim, clique em **Okey**, em seguida, **Okey**. Visual Studio cria um projeto usando o modelo de identidade do ASP.NET Core.
5. Selecione **ferramentas > Gerenciador de pacotes NuGet > Package Manager Console** para abrir o **Package Manager Console** Janela (PMC).
6. Navegue até a raiz do projeto no PMC e execute o `Entity Framework (EF) Core Update-Database` comando.

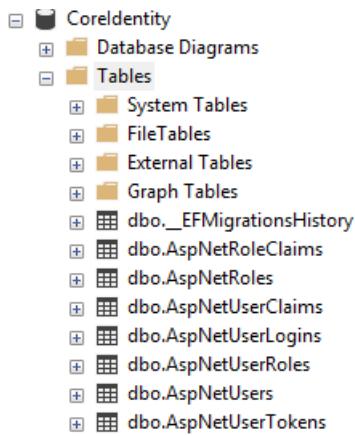
Identidade do ASP.NET Core 2.0 usa o EF Core para interagir com o banco de dados armazenar os dados de autenticação. Em ordem para o aplicativo recém-criado trabalhar, preciso haver um banco de dados para armazenar esses dados. Depois de criar um novo aplicativo, a maneira mais rápida para inspecionar o esquema em um ambiente de banco de dados é criar o banco de dados usando [migrações do EF Core](#). Esse processo cria um banco de dados, localmente ou em outro lugar, que imita o esquema. Examine a documentação anterior para obter mais informações.

Comandos do EF Core usam a cadeia de caracteres de conexão para o banco de dados especificado na `appSettings.json`. A cadeia de conexão a seguir tem como alvo um banco de dados no `localhost` denominado `asp-net-core-identity`. Nessa configuração, o EF Core está configurado para usar o `DefaultConnection` cadeia de caracteres de conexão.

```
{  
  "ConnectionStrings": {  
    "DefaultConnection": "Server=localhost;Database=aspnet-core-  
identity;Trusted_Connection=True;MultipleActiveResultSets=true"  
  }  
}
```

7. Selecione **modo de exibição > SQL Server Object Explorer**. Expanda o nó correspondente ao nome do banco de dados especificado na `ConnectionStrings:DefaultConnection` propriedade de `appSettings.json`.

O `Update-Database` comando criou o banco de dados especificado com o esquema e os dados necessários para a inicialização do aplicativo. A imagem a seguir ilustra a estrutura da tabela que é criada com as etapas anteriores.



## Migrar o esquema

Há diferenças sutis nos campos de associação e o ASP.NET Core Identity e estruturas de tabela. O padrão foi alterado significativamente para a autenticação/autorização com aplicativos ASP.NET e ASP.NET Core. Os objetos principais que ainda são utilizados com identidade são os *usuários* e *funções*. Aqui estão as tabelas de mapeamento para os *usuários*, *funções*, e *UserRoles*.

## Usuários

IDENTIDADE (DBO. ASPNETUSERS)		ASSOCIAÇÃO (DBO.ASPNET_USERS / DBO.ASPNET_MEMBERSHIP)	
Nome do campo	Tipo	Nome do campo	Tipo
Id	cadeia de caracteres	aspnet_Users.UserId	cadeia de caracteres
UserName	cadeia de caracteres	aspnet_Users.UserName	cadeia de caracteres
Email	cadeia de caracteres	aspnet_Membership.Email	cadeia de caracteres
NormalizedUserName	cadeia de caracteres	aspnet_Users.LoweredUserName	cadeia de caracteres
NormalizedEmail	cadeia de caracteres	aspnet_Membership.LoweredEmail	cadeia de caracteres
PhoneNumber	cadeia de caracteres	aspnet_Users.MobileAlias	cadeia de caracteres
LockoutEnabled	bit	aspnet_Membership.IsLockedout	bit

### NOTE

Nem todos os mapeamentos de campo se parecer com relações um para um dos membros no ASP.NET Core Identity. A tabela anterior usa o esquema de usuário da associação padrão e mapeia-os para o esquema de identidade do ASP.NET Core. Todos os outros campos personalizados que foram usados para associação precisam ser mapeados manualmente. Em que esse mapeamento, não há nenhum mapa para senhas, como critérios de senha e a senha salts não migram entre os dois. É recomendável deixar a senha como null e pedir que os usuários redefinam suas senhas. No ASP.NET Core Identity, LockoutEnd deverá ser definida para alguma data no futuro, se o usuário está bloqueado. Isso é mostrado no script de migração.

## Funções

IDENTIDADE (DBO. ASPNETROLES)		ASSOCIAÇÃO (DBO.ASPNET_ROLES)	
Nome do campo	Tipo	Nome do campo	Tipo
Id	cadeia de caracteres	RoleId	cadeia de caracteres
Name	cadeia de caracteres	RoleName	cadeia de caracteres
NormalizedName	cadeia de caracteres	LoweredRoleName	cadeia de caracteres

## Funções de usuário

<b>IDENTIDADE (DBO.ASPNETUSERROLES)</b>		<b>ASSOCIAÇÃO (DBO.ASPNET_USERSINROLES)</b>	
<b>Nome do campo</b>	<b>Tipo</b>	<b>Nome do campo</b>	<b>Tipo</b>
RoleId	cadeia de caracteres	RoleId	cadeia de caracteres
UserId	cadeia de caracteres	UserId	cadeia de caracteres

Fazer referência as tabelas de mapeamento anterior durante a criação de um script de migração para os usuários e funções. O exemplo a seguir pressupõe que você tenha dois bancos de dados em um servidor de banco de dados. Um banco de dados contém o esquema de associação do ASP.NET existente e os dados. A outra `CoreIdentitySample` banco de dados foi criado usando as etapas descritas anteriormente. Os comentários são incluídas embutidas para obter mais detalhes.

```
-- THIS SCRIPT NEEDS TO RUN FROM THE CONTEXT OF THE MEMBERSHIP DB
BEGIN TRANSACTION MigrateUsersAndRoles
USE aspnetdb

-- INSERT USERS
INSERT INTO CoreIdentitySample.dbo.AspNetUsers
    (Id,
     UserName,
     NormalizedUserName,
     PasswordHash,
     SecurityStamp,
     EmailConfirmed,
     PhoneNumber,
     PhoneNumberConfirmed,
     TwoFactorEnabled,
     LockoutEnd,
     LockoutEnabled,
     AccessFailedCount,
     Email,
     NormalizedEmail)
SELECT aspnet_Users.UserId,
       aspnet_Users.UserName,
       -- The NormalizedUserName value is upper case in ASP.NET Core Identity
       UPPER(aspnet_Users.UserName),
       -- Creates an empty password since passwords don't map between the 2 schemas
       '',
       /*
       The SecurityStamp token is used to verify the state of an account and
       is subject to change at any time. It should be initialized as a new ID.
       */
       NewID(),
       /*
       EmailConfirmed is set when a new user is created and confirmed via email.
       Users must have this set during migration to reset passwords.
       */
       1,
       aspnet_Users.MobileAlias,
       CASE
           WHEN aspnet_Users.MobileAlias IS NULL THEN 0
           ELSE 1
       END,
       -- 2FA likely wasn't setup in Membership for users, so setting as false.
       0,
       CASE
           -- Setting lockout date to time in the future (1,000 years)
           WHEN aspnet_Membership.IsLockedOut = 1 THEN Dateadd(year, 1000,
                                                               Sysutcdatetime())
           ELSE NULL
       END,
```

```

    aspnet_Membership.IsLockedOut,
/*
AccessFailedAccount is used to track failed logins. This is stored in
Membership in multiple columns. Setting to 0 arbitrarily.
*/
0,
aspnet_Membership.Email,
-- The NormalizedEmail value is upper case in ASP.NET Core Identity
UPPER(aspnet_Membership.Email)
FROM aspnet_Users
LEFT OUTER JOIN aspnet_Membership
    ON aspnet_Membership.ApplicationId =
        aspnet_Users.ApplicationId
    AND aspnet_Users.UserId = aspnet_Membership.UserId
LEFT OUTER JOIN CoreIdentitySample.dbo.AspNetUsers
    ON aspnet_Membership.UserId = AspNetUsers.Id
WHERE AspNetUsers.Id IS NULL

-- INSERT ROLES
INSERT INTO CoreIdentitySample.dbo.AspNetRoles(Id, Name)
SELECT RoleId, RoleName
FROM aspnet_Roles;

-- INSERT USER ROLES
INSERT INTO CoreIdentitySample.dbo.AspNetUserRoles(UserId, RoleId)
SELECT UserId, RoleId
FROM aspnet_UsersInRoles;

IF @@ERROR <> 0
BEGIN
    ROLLBACK TRANSACTION MigrateUsersAndRoles
    RETURN
END

COMMIT TRANSACTION MigrateUsersAndRoles

```

Após a conclusão do script anterior, o aplicativo ASP.NET Core Identity criado anteriormente é preenchido com os usuários de associação. Os usuários precisam alterar suas senhas antes de fazer logon.

#### NOTE

Se o sistema de associação tivesse os usuários com nomes de usuário que não correspondiam a seu endereço de email, as alterações são necessárias para o aplicativo criado anteriormente para acomodar isso. O modelo padrão espera `UserName` e `Email` sejam iguais. Para situações em que eles são diferentes, o processo de logon deve ser modificado para usar `UserName` em vez de `Email`.

No `PageModel` da página de logon, localizado em `Pages\Account\Login.cshtml.cs`, remova o `[EmailAddress]` de atributos dos `Email` propriedade. Renomeie-o para *nome de usuário*. Isso exige uma alteração sempre que `EmailAddress` mencionada, além de *exibição* e `PageModel`. O resultado é semelhante ao seguinte:

# Log in

Use a local account to log in.

**UserName**

**Password**

,  Remember me?

[Log in](#)

[Forgot your password?](#)

[Register as a new user](#)

## Próximas etapas

Neste tutorial, você aprendeu a porta que os usuários da associação do SQL para a identidade do ASP.NET Core 2.0. Para obter mais informações sobre a identidade do ASP.NET Core, consulte [Introdução à identidade](#).

# Migrar módulos e manipuladores HTTP para middleware do ASP.NET Core

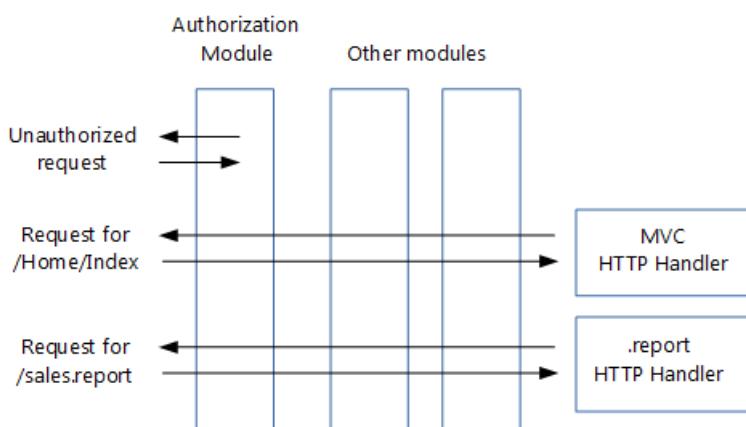
23/08/2018 • 25 minutes to read • [Edit Online](#)

Por Matt Perdeck

Este artigo mostra como migrar do ASP.NET existentes [módulos HTTP e manipuladores de System.webServer](#) para o ASP.NET Core [middleware](#).

## Módulos e manipuladores revistos

Antes de prosseguir para o middleware do ASP.NET Core, vamos primeiro recapitular como funcionam os manipuladores e módulos HTTP:



### Os manipuladores são:

- As classes que implementam [IHttpHandler](#)
- Usado para manipular solicitações com um determinado nome de arquivo ou a extensão, como `.report`
- [Configurado](#) em *Web.config*

### Módulos são:

- As classes que implementam  [IHttpModule](#)
- Chamado para cada solicitação
- Capazes de curto-circuito (interromper o processamento adicional de uma solicitação)
- Capaz de adicionar à resposta HTTP, ou criar seus próprios
- [Configurado](#) em *Web.config*

### A ordem na qual os módulos de processam solicitações de entrada é determinada por:

1. O [ciclo de vida do aplicativo](#), que é disparado pelo ASP.NET um eventos da série: `BeginRequest`, `AuthenticateRequest`, etc. Cada módulo pode criar um manipulador para um ou mais eventos.
2. Para o mesmo evento, a ordem na qual eles foram configurados na *Web.config*.

Além de módulos, você pode adicionar manipuladores para os eventos de ciclo de vida para seus `Global.asax.cs`

arquivo. Esses manipuladores executar após os manipuladores nos módulos configurados.

## De manipuladores e módulos para middleware

### Middleware são mais simples do que os manipuladores e módulos HTTP:

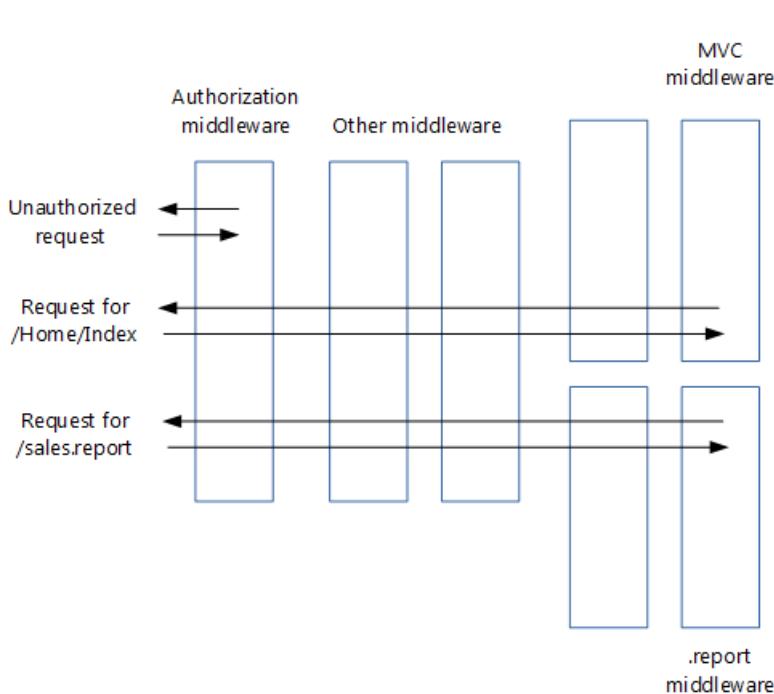
- Módulos, manipuladores *Global.asax.cs*, *Web.config* (exceto para a configuração do IIS) e o ciclo de vida do aplicativo serão excluídos
- As funções dos módulos e manipuladores foram tomadas a tecla TAB middleware
- Middleware são configurados usando o código em vez de em *Web.config*
- [Ramificação do pipeline](#) permite que você envia solicitações para o middleware específico, com base em não apenas a URL, mas também em cabeçalhos de solicitação, cadeias de caracteres de consulta, etc.

### Middleware são muito semelhantes aos módulos:

- Invocado em princípio, para cada solicitação
- Capazes de curto-circuito a uma solicitação por [não passar a solicitação para o próximo middleware](#)
- Capaz de criar sua própria resposta HTTP

### Middleware e módulos são processados em uma ordem diferente:

- Pedido de middleware é baseado na ordem em que eles são inseridos no pipeline de solicitação, enquanto a ordem dos módulos se baseia principalmente na [ciclo de vida do aplicativo](#) eventos
- Ordem de middleware para respostas é o inverso do que para solicitações, enquanto a ordem dos módulos é o mesmo para solicitações e respostas
- Consulte [criar um pipeline de middleware com o IApplicationBuilder](#)



Observe como na imagem acima, o middleware de autenticação sofreu curto-circuito a solicitação.

## Migrando o código de módulo para middleware

Um módulo HTTP existente será semelhante a este:

```
// ASP.NET 4 module

using System;
using System.Web;

namespace MyApp.Modules
{
    public class MyModule : IHttpModule
    {
        public void Dispose()
        {
        }

        public void Init(HttpApplication application)
        {
            application.BeginRequest += (new EventHandler(this.Application_BeginRequest));
            application.EndRequest += (new EventHandler(this.Application_EndRequest));
        }

        private void Application_BeginRequest(Object source, EventArgs e)
        {
            HttpContext context = ((HttpApplication)source).Context;

            // Do something with context near the beginning of request processing.
        }

        private void Application_EndRequest(Object source, EventArgs e)
        {
            HttpContext context = ((HttpApplication)source).Context;

            // Do something with context near the end of request processing.
        }
    }
}
```

Conforme mostrado na [Middleware](#) página, um middleware do ASP.NET Core é uma classe que expõe um `Invoke` levando método um `HttpContext` e retornando um `Task`. Seu novo middleware terá esta aparência:

```

// ASP.NET Core middleware

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Http;
using System.Threading.Tasks;

namespace MyApp.Middleware
{
    public class MyMiddleware
    {
        private readonly RequestDelegate _next;

        public MyMiddleware(RequestDelegate next)
        {
            _next = next;
        }

        public async Task Invoke(HttpContext context)
        {
            // Do something with context near the beginning of request processing.

            await _next.Invoke(context);

            // Clean up.
        }
    }

    public static class MyMiddlewareExtensions
    {
        public static IApplicationBuilder UseMyMiddleware(this IApplicationBuilder builder)
        {
            return builder.UseMiddleware<MyMiddleware>();
        }
    }
}

```

O modelo de middleware anterior foi obtido da seção em [escrever middleware](#).

- O `MyMiddlewareExtensions` classe auxiliar torna mais fácil de configurar seu middleware em seu `Startup` classe.
- O `UseMyMiddleware` método adiciona sua classe de middleware ao pipeline de solicitação. Serviços necessários para o middleware são injetados no construtor do middleware.

O módulo pode encerrar uma solicitação, por exemplo, se o usuário não autorizado:

```

// ASP.NET 4 module that may terminate the request

private void Application_BeginRequest(Object source, EventArgs e)
{
    HttpContext context = ((HttpApplication)source).Context;

    // Do something with context near the beginning of request processing.

    if (TerminateRequest())
    {
        context.Response.End();
        return;
    }
}

```

Um middleware lida com isso, não chamando `Invoke` no próximo middleware no pipeline. Tenha em mente que isso não encerra completamente a solicitação, porque o middleware anterior ainda será chamado quando a resposta faz seu caminho através do pipeline.

```
// ASP.NET Core middleware that may terminate the request

public async Task Invoke(HttpContext context)
{
    // Do something with context near the beginning of request processing.

    if (!TerminateRequest())
        await _next.Invoke(context);

    // Clean up.
}
```

Ao migrar a funcionalidade do módulo para seu novo middleware, você pode achar que seu código não era compilado porque o `HttpContext` classe foi alterado significativamente no ASP.NET Core. [Mais tarde](#), você verá como migrar para ASP.NET Core `HttpContext` novo.

## Migrando inserção de módulo no pipeline de solicitação

Módulos HTTP normalmente são adicionados ao pipeline de solicitação usando `Web.config`:

```
<?xml version="1.0" encoding="utf-8"?>
<!--ASP.NET 4 web.config-->
<configuration>
    <system.webServer>
        <modules>
            <add name="MyModule" type="MyApp.Modules.MyModule"/>
        </modules>
    </system.webServer>
</configuration>
```

Converter isso por [adicionando seu novo middleware](#) ao pipeline de solicitação no seu `Startup` classe:

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole(Configuration.GetSection("Logging"));
    loggerFactory.AddDebug();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseBrowserLink();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseMyMiddleware();

    app.UseMyMiddlewareWithParams();

    var myMiddlewareOptions = Configuration.GetSection("MyMiddlewareOptionsSection").Get<MyMiddlewareOptions>()
();
    var myMiddlewareOptions2 =
Configuration.GetSection("MyMiddlewareOptionsSection2").Get<MyMiddlewareOptions>();
    app.UseMyMiddlewareWithParams(myMiddlewareOptions);
    app.UseMyMiddlewareWithParams(myMiddlewareOptions2);

    app.UseMyTerminatingMiddleware();

    // Create branch to the MyHandlerMiddleware.
    // All requests ending in .report will follow this branch.
    app.MapWhen(
        context => context.Request.Path.ToString().EndsWith(".report"),
        appBranch => {
            // ... optionally add more middleware to this branch
            appBranch.UseMyHandler();
        });
    app.MapWhen(
        context => context.Request.Path.ToString().EndsWith(".context"),
        appBranch => {
            appBranch.UseHttpContextDemoMiddleware();
       });
    app.UseStaticFiles();

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}

```

O ponto exato no pipeline de onde você insere seu middleware novo depende do evento que ele tratado como um módulo (`BeginRequest`, `EndRequest`, etc.) e sua ordem na sua lista de módulos *Web.config*.

Conforme anteriormente não mencionado, há mais nenhum ciclo de vida do aplicativo no ASP.NET Core e a ordem na qual as respostas são processadas pelo middleware é diferente da ordem usada pelos módulos. Isso pode tornar sua decisão de ordenação mais desafiador.

Se a ordenação se torna um problema, você pode dividir seu módulo em vários componentes de middleware que podem ser ordenados de forma independente.

## Migrando o código de manipulador para middleware

Um manipulador HTTP é semelhante a:

```
// ASP.NET 4 handler

using System.Web;

namespace MyApp.HttpHandlers
{
    public class MyHandler : IHttpHandler
    {
        public bool IsReusable { get { return true; } }

        public void ProcessRequest(HttpContext context)
        {
            string response = GenerateResponse(context);

            context.Response.ContentType = GetContentType();
            context.Response.Output.Write(response);
        }

        // ...

        private string GenerateResponse(HttpContext context)
        {
            string title = context.Request.QueryString["title"];
            return string.Format("Title of the report: {0}", title);
        }

        private string GetContentType()
        {
            return "text/plain";
        }
    }
}
```

Em seu projeto ASP.NET Core, isso pode ser traduzido para um middleware semelhante a esta:

```

// ASP.NET Core middleware migrated from a handler

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Http;
using System.Threading.Tasks;

namespace MyApp.Middleware
{
    public class MyHandlerMiddleware
    {

        // Must have constructor with this signature, otherwise exception at run time
        public MyHandlerMiddleware(RequestDelegate next)
        {
            // This is an HTTP Handler, so no need to store next
        }

        public async Task Invoke(HttpContext context)
        {
            string response = GenerateResponse(context);

            context.Response.ContentType = GetContentType();
            await context.Response.WriteAsync(response);
        }

        // ...

        private string GenerateResponse(HttpContext context)
        {
            string title = context.Request.Query["title"];
            return string.Format("Title of the report: {0}", title);
        }

        private string GetContentType()
        {
            return "text/plain";
        }
    }

    public static class MyHandlerExtensions
    {
        public static IApplicationBuilder UseMyHandler(this IApplicationBuilder builder)
        {
            return builder.UseMiddleware<MyHandlerMiddleware>();
        }
    }
}

```

Este middleware é muito semelhante ao middleware correspondente aos módulos. A única diferença é que não há aqui nenhuma chamada para `_next.Invoke(context)`. Isso faz sentido, porque o manipulador não é no final do pipeline de solicitação, portanto, não haverá nenhum próximo middleware para invocar.

## Migrando inserção manipulador no pipeline de solicitação

Configurar um manipulador HTTP é feito na *Web.config* e pode ter esta aparência:

```
<?xml version="1.0" encoding="utf-8"?>
<!--ASP.NET 4 web.config-->
<configuration>
  <system.webServer>
    <handlers>
      <add name="MyHandler" verb="*" path="*.report" type="MyApp.HttpHandlers.MyHandler"
resourceType="Unspecified" preCondition="integratedMode"/>
    </handlers>
  </system.webServer>
</configuration>
```

Você pode converter isso adicionando seu novo middleware de manipulador para o pipeline de solicitação no seu `Startup` classe, semelhante ao middleware convertido de módulos. O problema com essa abordagem é que ele seria enviar todas as solicitações para seu novo middleware de manipulador. No entanto, você precisará apenas solicitações com uma determinada extensão para alcançar seu middleware. Essa seria a mesma funcionalidade que tinha com seu manipulador HTTP.

Uma solução é ramificar o pipeline para solicitações com uma determinada extensão, usando o `MapWhen` método de extensão. Você pode fazer isso no mesmo `configure` método onde você pode adicionar o outro middleware:

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole(Configuration.GetSection("Logging"));
    loggerFactory.AddDebug();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseBrowserLink();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseMyMiddleware();

    app.UseMyMiddlewareWithParams();

    var myMiddlewareOptions = Configuration.GetSection("MyMiddlewareOptionsSection").Get<MyMiddlewareOptions>()
();
    var myMiddlewareOptions2 =
Configuration.GetSection("MyMiddlewareOptionsSection2").Get<MyMiddlewareOptions>();
    app.UseMyMiddlewareWithParams(myMiddlewareOptions);
    app.UseMyMiddlewareWithParams(myMiddlewareOptions2);

    app.UseMyTerminatingMiddleware();

    // Create branch to the MyHandlerMiddleware.
    // All requests ending in .report will follow this branch.
    app.MapWhen(
        context => context.Request.Path.ToString().EndsWith(".report"),
        appBranch => {
            // ... optionally add more middleware to this branch
            appBranch.UseMyHandler();
        });
    app.MapWhen(
        context => context.Request.Path.ToString().EndsWith(".context"),
        appBranch => {
            appBranch.UseHttpContextDemoMiddleware();
       });
    app.UseStaticFiles();

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}

```

`MapWhen` usa esses parâmetros:

1. Um lambda que usa o `HttpContext` e retorna `true` se a solicitação deve ir para baixo a ramificação. Isso significa que você pode ramificar solicitações não apenas com base em sua extensão, mas também em cabeçalhos de solicitação, parâmetros de cadeia de caracteres de consulta, etc.
2. Um lambda que usa um `IApplicationBuilder` e adiciona o middleware para a ramificação. Isso significa que você pode adicionar outro middleware para a ramificação na frente do seu manipulador middleware.

Middleware adicionado ao pipeline antes da ramificação será invocada em todas as solicitações; a ramificação não terá impacto sobre eles.

# Opções de middleware usando o padrão de opções de carregamento

Alguns manipuladores e módulos têm opções de configuração que são armazenadas em *Web.config*. No entanto, no ASP.NET Core um novo modelo de configuração é usado no lugar de *Web.config*.

O novo [sistema de configuração](#) fornece essas opções para resolver isso:

- Injetar diretamente as opções para o middleware, como mostra a [próxima seção](#).
- Use o [padrão de opções](#):

1. Crie uma classe para manter suas opções de middleware, por exemplo:

```
public class MyMiddlewareOptions
{
    public string Param1 { get; set; }
    public string Param2 { get; set; }
}
```

2. Store os valores de opção

O sistema de configuração permite que você armazene valores de opção em qualquer local desejado. No entanto, a maioria dos locais use *appSettings.JSON*, portanto, vamos dar essa abordagem:

```
{
    "MyMiddlewareOptionsSection": {
        "Param1": "Param1Value",
        "Param2": "Param2Value"
    }
}
```

*MyMiddlewareOptionsSection* aqui é um nome de seção. Ele não precisa ser igual ao nome da sua classe de opções.

3. Associar os valores de opção com a classe de opções

O padrão de opções usa a estrutura de injeção de dependência do ASP.NET Core para associar o tipo de opções (como `MyMiddlewareOptions`) com um `MyMiddlewareOptions` objeto que tem as opções reais.

Atualização de seu `Startup` classe:

- Se você estiver usando *appSettings.JSON*, adicione-o ao construtor de configuração no `Startup` construtor:

```
public Startup(IHostingEnvironment env)
{
    var builder = new ConfigurationBuilder()
        .SetBasePath(env.ContentRootPath)
        .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
        .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true)
        .AddEnvironmentVariables();
    Configuration = builder.Build();
}
```

- Configure o serviço de opções:

```

public void ConfigureServices(IServiceCollection services)
{
    // Setup options service
    services.AddOptions();

    // Load options from section "MyMiddlewareOptionsSection"
    services.Configure<MyMiddlewareOptions>(
        Configuration.GetSection("MyMiddlewareOptionsSection"));

    // Add framework services.
    services.AddMvc();
}

```

c. Associe suas opções de sua classe de opções:

```

public void ConfigureServices(IServiceCollection services)
{
    // Setup options service
    services.AddOptions();

    // Load options from section "MyMiddlewareOptionsSection"
    services.Configure<MyMiddlewareOptions>(
        Configuration.GetSection("MyMiddlewareOptionsSection"));

    // Add framework services.
    services.AddMvc();
}

```

4. Injete as opções para o construtor de middleware. Isso é semelhante a injeção de opções em um controlador.

```

public class MyMiddlewareWithParams
{
    private readonly RequestDelegate _next;
    private readonly MyMiddlewareOptions _myMiddlewareOptions;

    public MyMiddlewareWithParams(RequestDelegate next,
        IOptions<MyMiddlewareOptions> optionsAccessor)
    {
        _next = next;
        _myMiddlewareOptions = optionsAccessor.Value;
    }

    public async Task Invoke(HttpContext context)
    {
        // Do something with context near the beginning of request processing
        // using configuration in _myMiddlewareOptions

        await _next.Invoke(context);

        // Do something with context near the end of request processing
        // using configuration in _myMiddlewareOptions
    }
}

```

O [UseMiddleware](#) método de extensão que adiciona o middleware para o `IApplicationBuilder` se encarrega de injeção de dependência.

Isso não é limitado a `IOptions` objetos. Qualquer outro objeto requer que seu middleware pode ser injetado dessa maneira.

## Opções de middleware por meio de injeção direta de carregamento

O padrão de opções tem a vantagem que ele cria o acoplamento solto entre valores de opções e seus consumidores. Depois que você associou a uma classe de opções com os valores de opções propriamente dito, qualquer outra classe pode obter acesso às opções por meio da estrutura de injeção de dependência. Não é necessário para passar ao redor de valores de opções.

Isso divide entanto se você quiser usar o mesmo middleware duas vezes, com opções diferentes. Por exemplo um middleware de autorização usado em ramificações diferentes, permitindo que diferentes funções. É possível associar dois objetos diferentes opções com a classe de opções de um.

A solução é obter os objetos de opções com os valores de opções propriamente dito no seu `Startup` de classe e passá-las diretamente para cada instância do seu middleware.

### 1. Adicionar uma segunda chave a ser `appSettings.json`

Para adicionar um segundo conjunto de opções para o `appSettings.json` de arquivo, use uma nova chave de para identificá-lo exclusivamente:

```
{  
  "MyMiddlewareOptionsSection2": {  
    "Param1": "Param1Value2",  
    "Param2": "Param2Value2"  
  },  
  "MyMiddlewareOptionsSection": {  
    "Param1": "Param1Value",  
    "Param2": "Param2Value"  
  }  
}
```

### 2. Recuperar valores de opções e passá-los para o middleware. O `use...` método de extensão (o que adiciona seu middleware ao pipeline) é um lugar lógico para transmitir os valores de opção:

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole(Configuration.GetSection("Logging"));
    loggerFactory.AddDebug();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseBrowserLink();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseMyMiddleware();

    app.UseMyMiddlewareWithParams();

    var myMiddlewareOptions =
Configuration.GetSection("MyMiddlewareOptionsSection").Get<MyMiddlewareOptions>();
    var myMiddlewareOptions2 =
Configuration.GetSection("MyMiddlewareOptionsSection2").Get<MyMiddlewareOptions>();
    app.UseMyMiddlewareWithParams(myMiddlewareOptions);
    app.UseMyMiddlewareWithParams(myMiddlewareOptions2);

    app.UseMyTerminatingMiddleware();

    // Create branch to the MyHandlerMiddleware.
    // All requests ending in .report will follow this branch.
    app.MapWhen(
        context => context.Request.Path.ToString().EndsWith(".report"),
        appBranch => {
            // ... optionally add more middleware to this branch
            appBranch.UseMyHandler();
        });
}

app.MapWhen(
    context => context.Request.Path.ToString().EndsWith(".context"),
    appBranch => {
        appBranch.UseHttpContextDemoMiddleware();
    });

app.UseStaticFiles();

app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});
}

```

3. Habilite o middleware para utilizar um parâmetro de opções. Forneça uma sobrecarga do `Use...` método de extensão (que usa o parâmetro `options` e passa-o para `UseMiddleware`). Quando `UseMiddleware` é chamado com parâmetros, ele passa os parâmetros para o construtor de middleware quando ele instancia o objeto de middleware.

```
public static class MyMiddlewareWithParamsExtensions
{
    public static IApplicationBuilder UseMyMiddlewareWithParams(
        this IApplicationBuilder builder)
    {
        return builder.UseMiddleware<MyMiddlewareWithParams>();
    }

    public static IApplicationBuilder UseMyMiddlewareWithParams(
        this IApplicationBuilder builder, MyMiddlewareOptions myMiddlewareOptions)
    {
        return builder.UseMiddleware<MyMiddlewareWithParams>(
            new OptionsWrapper<MyMiddlewareOptions>(myMiddlewareOptions));
    }
}
```

Observe como isso encapsula o objeto de opções em um `OptionsWrapper` objeto. Isso implementa `IOptions`, conforme o esperado pelo construtor de middleware.

## Migrando para o novo HttpContext

Você viu antes que o `Invoke` método no seu middleware usa um parâmetro de tipo `HttpContext`:

```
public async Task Invoke(HttpContext context)
```

`HttpContext` foi alterado significativamente no ASP.NET Core. Esta seção mostra como converter as propriedades mais usadas da `System.Web.HttpContext` para o novo `Microsoft.AspNetCore.Http.HttpContext`.

### HttpContext

**HttpContext.Items** se traduz em:

```
IDictionary<object, object> items = httpContext.Items;
```

### ID de solicitação exclusiva (nenhum equivalente `System.Web.HttpContext`)

Fornece uma id exclusiva para cada solicitação. Muito útil para incluir em seus logs.

```
string requestId = httpContext.TraceIdentifier;
```

### HttpContext.Request

**HttpContext.Request.HttpMethod** se traduz em:

```
string httpMethod = httpContext.Request.Method;
```

**HttpContext.Request.QueryString** se traduz em:

```

IQueryCollection queryParameters = httpContext.Request.Query;

// If no query parameter "key" used, values will have 0 items
// If single value used for a key (...?key=v1), values will have 1 item ("v1")
// If key has multiple values (...?key=v1&key=v2), values will have 2 items ("v1" and "v2")
IList<string> values = queryParameters["key"];

// If no query parameter "key" used, value will be ""
// If single value used for a key (...?key=v1), value will be "v1"
// If key has multiple values (...?key=v1&key=v2), value will be "v1,v2"
string value = queryParameters["key"].ToString();

```

**HttpContext.Request.Url** e **HttpContext.Request.RawUrl** traduzir para:

```

// using Microsoft.AspNetCore.Http.Extensions;
var url = httpContext.Request.GetDisplayUrl();

```

**HttpContext.Request.IsSecureConnection** se traduz em:

```

var isSecureConnection = httpContext.Request.IsHttps;

```

**HttpContext.Request.UserHostAddress** se traduz em:

```

var userHostAddress = httpContext.Connection.RemoteIpAddress?.ToString();

```

**HttpContext.Request.Cookies** se traduz em:

```

 IRequestCookieCollection cookies = httpContext.Request.Cookies;
 string unknownCookieValue = cookies["unknownCookie"]; // will be null (no exception)
 string knownCookieValue = cookies["cookie1name"]; // will be actual value

```

**HttpContext.Request.RequestContext.RouteData** se traduz em:

```

var routeValue = httpContext.GetRouteValue("key");

```

**HttpContext.Request.Headers** se traduz em:

```

// using Microsoft.AspNetCore.Http.Headers;
// using Microsoft.Net.Http.Headers;

IHeaderDictionary headersDictionary = httpContext.Request.Headers;

// GetTypedHeaders extension method provides strongly typed access to many headers
var requestHeaders = httpContext.Request.GetTypedHeaders();
CacheControlHeaderValue cacheControlHeaderValue = requestHeaders.CacheControl;

// For unknown header, unknownheaderValues has zero items and unknownHeaderValue is ""
IList<string> unknownheaderValues = headersDictionary["unknownheader"];
string unknownHeaderValue = headersDictionary["unknownheader"].ToString();

// For known header, knownheaderValues has 1 item and knownHeaderValue is the value
IList<string> knownheaderValues = headersDictionary[HeaderNames.AcceptLanguage];
string knownHeaderValue = headersDictionary[HeaderNames.AcceptLanguage].ToString();

```

**HttpContext.Request.UserAgent** se traduz em:

```
string userAgent = headersDictionary[HeaderNames.UserAgent].ToString();
```

**HttpContext.Request.UrlReferrer** se traduz em:

```
string urlReferrer = headersDictionary[HeaderNames.Referer].ToString();
```

**HttpContext.Request.ContentType** se traduz em:

```
// using Microsoft.Net.Http.Headers;

MediaTypeHeaderValue mediaHeaderValue = requestHeaders.ContentType;
string contentType = mediaHeaderValue?.MediaType.ToString(); // ex. application/x-www-form-urlencoded
string contentMainType = mediaHeaderValue?.Type.ToString(); // ex. application
string contentSubType = mediaHeaderValue?.SubType.ToString(); // ex. x-www-form-urlencoded

System.Text.Encoding requestEncoding = mediaHeaderValue?.Encoding;
```

**HttpContext.Request.Form** se traduz em:

```
if (httpContext.Request.HasFormContentType)
{
    IFormCollection form;

    form = httpContext.Request.Form; // sync
    // Or
    form = await httpContext.Request.ReadFormAsync(); // async

    string firstName = form["firstname"];
    string lastName = form["lastname"];
}
```

#### WARNING

Ler os valores de formulário apenas se for o tipo de conteúdo sub *x-www-form-urlencoded* ou *dados de formulário*.

**HttpContext.Request.InputStream** se traduz em:

```
string inputBody;
using (var reader = new System.IO.StreamReader(
    httpContext.Request.Body, System.Text.Encoding.UTF8))
{
    inputBody = reader.ReadToEnd();
}
```

#### WARNING

Use esse código somente em um middleware de tipo de manipulador, no final de um pipeline.

Você pode ler o corpo bruto, conforme mostrado acima apenas uma vez por solicitação. Middleware de tentativa de ler o corpo após a primeira leitura lê um corpo vazio.

Isso não se aplica à leitura de um formulário, conforme mostrado anteriormente, porque isso é feito de um buffer.

**HttpContext.Response**

**HttpContext.Response.Status** e **HttpContext.Response.StatusDescription** traduzir para:

```
// using Microsoft.AspNetCore.Http;
HttpContext.Response.StatusCode = StatusCodes.Status200OK;
```

**HttpContext.Response.ContentEncoding** e **HttpContext.Response.ContentType** traduzir para:

```
// using Microsoft.Net.Http.Headers;
var mediaType = new MediaTypeHeaderValue("application/json");
mediaType.Encoding = System.Text.Encoding.UTF8;
HttpContext.Response.ContentType = mediaType.ToString();
```

**HttpContext.Response.ContentType** em seu próprio também se traduz em:

```
HttpContext.Response.ContentType = "text/html";
```

**HttpContext.Response.Output** se traduz em:

```
string responseContent = GetResponseContent();
await HttpContext.Response.WriteAsync(responseContent);
```

**HttpContext.Response.TransmitFile**

Atendendo a um arquivo é discutida [aqui](#).

**HttpContext.Response.Headers**

Enviar cabeçalhos de resposta é complicado pelo fato de que se você defini-las depois que nada foi escrito para o corpo da resposta, eles não funcionará.

A solução é definir um método de retorno de chamada que será chamado direita antes de gravar do início da resposta. Isso é feito melhor no início do `Invoke` método em seu middleware. É esse método de retorno de chamada que define os cabeçalhos de resposta.

O código a seguir define um método de retorno de chamada chamado `SetHeaders`:

```
public async Task Invoke(HttpContext httpContext)
{
    // ...
    httpContext.Response.OnStarting(SetHeaders, state: httpContext);
```

O `SetHeaders` método de retorno de chamada teria esta aparência:

```

// using Microsoft.AspNetCore.Http.Headers;
// using Microsoft.Net.Http.Headers;

private Task SetHeaders(object context)
{
    var httpContext = (HttpContext)context;

    // Set header with single value
    httpContext.Response.Headers["ResponseHeaderName"] = "headerValue";

    // Set header with multiple values
    string[] responseHeaderValues = new string[] { "headerValue1", "headerValue1" };
    httpContext.Response.Headers["ResponseHeaderName"] = responseHeaderValues;

    // Translating ASP.NET 4's HttpContext.Response.RedirectLocation
    httpContext.Response.Headers[HeaderNames.Location] = "http://www.example.com";
    // Or
    httpContext.Response.Redirect("http://www.example.com");

    // GetTypedHeaders extension method provides strongly typed access to many headers
    var responseHeaders = httpContext.Response.GetTypedHeaders();

    // Translating ASP.NET 4's HttpContext.Response.CacheControl
    responseHeaders.CacheControl = new CacheControlHeaderValue
    {
        MaxAge = new System.TimeSpan(365, 0, 0, 0)
        // Many more properties available
    };

    // If you use .Net 4.6+, Task.CompletedTask will be a bit faster
    return Task.FromResult(0);
}

```

## HttpContext.Response.Cookies

Cookies de viagem para o navegador em um *Set-Cookie* cabeçalho de resposta. Como resultado, o envio de cookies requer mesmo retorno de chamada usado para enviar cabeçalhos de resposta:

```

public async Task Invoke(HttpContext httpContext)
{
    // ...
    httpContext.Response.OnStarting(SetCookies, state: httpContext);
    httpContext.Response.OnStarting(SetHeaders, state: httpContext);
}

```

O `SetCookies` método de retorno de chamada deve ser semelhante ao seguinte:

```

private Task SetCookies(object context)
{
    var httpContext = (HttpContext)context;

    IResponseCookies responseCookies = httpContext.Response.Cookies;

    responseCookies.Append("cookie1name", "cookie1value");
    responseCookies.Append("cookie2name", "cookie2value",
        new CookieOptions { Expires = System.DateTime.Now.AddDays(5), HttpOnly = true });

    // If you use .Net 4.6+, Task.CompletedTask will be a bit faster
    return Task.FromResult(0);
}

```

## Recursos adicionais

- Visão geral de módulos HTTP e de manipuladores HTTP
- Configuração
- Inicialização de aplicativos
- Middleware

# Migrar do Logging 2.1 para 2.2 ou 3.0

08/01/2019 • 2 minutes to read • [Edit Online](#)

Este artigo descreve as etapas comuns para a migração de um aplicativo ASP.NET Core que usa `Microsoft.Extensions.Logging` do 2.1, 2.2 ou 3.0.

## 2.1 para 2.2

Criar manualmente `ServiceCollection` e chamar `AddLogging`.

exemplo 2.1:

```
using (var loggerFactory = new LoggerFactory())
{
    loggerFactory.AddConsole();

    // use loggerFactory
}
```

exemplo de 2.2:

```
var serviceCollection = new ServiceCollection();
serviceCollection.AddLogging(builder => builder.AddConsole());

using (var serviceProvider = serviceCollection.BuildServiceProvider())
using (var loggerFactory = serviceProvider.GetService<ILoggerFactory>())
{
    // use loggerFactory
}
```

## 2.1 a 3.0

No 3.0, use `LoggingFactory.Create`.

exemplo 2.1:

```
using (var loggerFactory = new LoggerFactory())
{
    loggerFactory.AddConsole();

    // use loggerFactory
}
```

exemplo 3.0:

```
using (var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole()))
{
    // use loggerFactory
}
```

## Recursos adicionais

