

Actions assíncronas no ASP .NET MVC

por José Barbosa | fev 5, 2016 | Desenvolvimento | 4 comentários

O *.NET Framework 4.5* apresenta uma abordagem simplificada à programação assíncrona fazendo uso das palavras-chaves *async* e *await* para criar métodos assíncronos. Esse modelo é chamado de *Task-based Asynchronous Pattern (TAP)* e torna muito mais fácil escrever trechos de código assíncrono se comparado aos métodos anteriores.

Desde o *ASP NET MVC 4* você pode escrever *actions methods* assíncronas, veja o código abaixo:

```
public async Task<IActionResult> GetAsync(long id)
{
    var foo = await repository.GetAsync(id);
    var bar = await otherRepository.GetAsync(foo.Id);
}
```

Repare que essa *action* retorna um objeto do tipo

Task<ActionResult>; faz uso dos modificadore **async** e **await** e seu nome termina com **Async**.

As palavras-chave *async* e *await* e o tipo *Task* são o coração da programação assíncrona no *.NET*. Você deve utilizar essa poderosa combinação para métodos assíncronos, mas lembre-se sempre de seguir as seguintes regras e convenções:

- O nome de um método assíncrono, por convenção, termina com um sufixo *Async*.

- O método deve retornar: `Task<TResult>`, `Task` ou `void`.
- A assinatura do método deve incluir o modificador `async`.

O web server trata de forma diferente as requisições assíncronas e síncronas, vamos entender essa diferença.

O que acontece em uma action síncrona

O servidor web de uma aplicação *.NET* possui um *pool* de *threads* que são usadas para servir as requisições que chegam em uma *action method*. Quando uma *request* chega, uma *thread* desse *pool* é encarregada de processar esse *request*. Se o *request* for processado de forma síncrona (uma *action method* sem *async*), a thread encarregada ficará bloqueada até o *request* finalizar, bloquear a thread enquanto uma requisição ocorre é um termo conhecido como uma operação *thread blocking*.

O que acontece de fato é que no *.NET* a execução de métodos é contínua, isso significa que se uma *thread* começar a executar um determinado método, ela irá ficar ocupada até a execução do método terminar. Em alguns casos a *thread* está completamente ocupada sem fazer nada.

O que acontece em uma action assíncrona

Você já deve ter ouvido falar no [node.js](https://nodejs.org/), certo? O Node.js é um *runtime* JavaScript que possui um modelo de *I/O assíncrono non-blocking singlethread*. Isso é possível porque o *node.js* faz uso de *apis/syscalls* para fazer *I/O assíncrono non-blocking singlethread* nativamente nos sistemas operacionais. Tudo

graças a [libuv](#), uma biblioteca *cross-platform* para abstrair I/O assíncrono.

Curiosidade: A biblioteca [libuv](#) também é usada no web server [Kestrel](#).

Assim como no *node.js* os métodos *async* do C# fazem I/O *non-blocking singlethread*. Quando você usa *async* e *await* você não está criando novas *threads*, mas sim fazendo trabalhos assíncronos em uma mesma *thread*.

Quando o *ASP .NET* encontra uma expressão *await* em uma *action async* ele não bloqueia o *thread* enquanto aguardada a execução. Em vez de isso a *thread* é devolvida para o *pool*, para servir novas *requests*. Apenas quando a operação assíncrona terminar a *thread* é notificada retoma o controle da execução dessa *action*.

Uma boa maneira de pensar sobre isso é imaginar que métodos assíncronos tem um botão de *pause* e *play*. Quando a *thread* em execução encontra uma expressão *await*, ela aperta o botão de *pause* e a execução do método é suspensa, fazendo com que a *thread* retorne para o *pool*. Quando a *task* que ela estava aguardando completa, é apertado o botão de *play*, e *thread* assume a execução do método é retomada.

Escolhendo *actions methods* síncronos e assíncronos

Existem algumas recomendações para quando usar *actions*

síncronas e assíncronas.

Em geral, use *actions* síncronas quando:

- A operação é simples e rápida.
- Simplicidade é mais importante que eficiência.
- A operação é baseada em CPU ao invés de envolver I/O. Usar assincronia em operações de CPU não prove benefícios e você está apenas gerando mais overhead.

Em geral, use *actions* assíncronas:

- A operação é baseada em I/O.
- Paralelismo é mais importante que simplicidade.
- Testes mostram que operações bloqueantes são o gargalo na performance de um site e o IIS pode servir mais requests usando métodos assíncronos para essas operações bloqueantes.

Conclusão

Uma *action async* demora o mesmo tempo de processamento do que uma *action* síncrona. Entretanto durante uma *action async* a *thread* não é bloqueada e pode responder outras *requests* enquanto espera a primeira *request* estar completa.

O *async*

A instrução ***async*** faz com que um método possa ser executado de forma assíncrona.

Você deve usar o modificador ***async*** para especificar que um método, *lambda expression* ou um método anônimo é assíncrono.

O *await*

A palavra reservada ***await*** é um ***syntax sugar*** que indica que um

pedaço de código deve esperar por algum outro pedaço de código de maneira não bloqueante.

Quando uma *action method* executa um *await*, a *thread* ao invés de ficar aguardando bloqueada, é liberada para atender outras *requests*.

A classe Task

A classe **Task<TResult>** representa uma **única operação** que **retorna** um valor; e essa **operação** pode¹ ser executada de forma assíncrona.

É possível usar **Task.Run** para mover o trabalho de CPU associado a uma nova thread, mas nesse caso o multithreading não ajuda com um processo que está apenas aguardando que os resultados tornem-se disponíveis (*await*).

A classe **Task<TResult>** representa uma **única operação** que **retorna** um valor; e essa **operação** pode¹ ser executada de forma assíncrona.

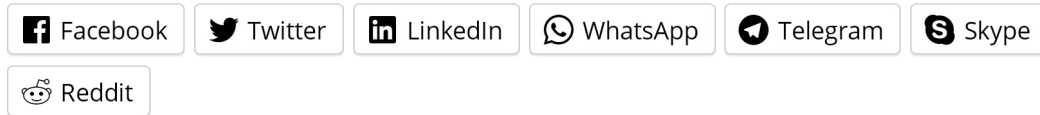
¹A instrução **async** faz com que um método **possa** ser chamado de forma assíncrona ~ não bloqueando a *thread* ~, mas sem sempre isso ocorre, então o método é chamado de forma síncrona ~ bloqueando a *thread* ~.

Para mais informações sobre *async* e *await* você pode ver essas referências.

- [Whitepaper: Asynchrony in .NET](#)

- [Async/Await FAQ](#)
- [Visual Studio Asynchronous Programming](#)

Compartilhe isso:



Related

4 Comentários

Lambda3

 Entrar ▾ Recomendar Tweet Compartilhar

Ordenar por Mais recentes ▾




Participe da discussão...

FAZER LOGIN COM

OU REGISTRE-SE NO DISQUS **Haruno Kenobi** • um ano atrás

Artigo bem esclarecedor!



  • Responder • Compartilhar ›**Edmar Munhoz** • 3 anos atrás

Em alguns exemplos que já vi na internet usava-se AsyncController neste artigo que não foi usado, o detalhe é que todos os artigos que vi que usava AsyncController eram exemplos antigos, então não sei se pode ser alguma mudança no framework e tal. Você saberia me dizer José.



  • Responder • Compartilhar ›**Allan Gomes** • 3 anos atrás

Existe alguma utilidade em utilizar Actions Async em uma WebAPI?

No cliente vou utilizar Angular (Web) e Delphi (Windows)

1   • Responder • Compartilhar ›**Lauro L. Nunes** ➔ Allan Gomes • um ano atrás

Boa pergunta Allan, também gostaria de saber..

  • Responder • Compartilhar ›

Eventos

Não há eventos previstos.

Próximos Posts

Nenhum post planejado no momento

 Don't miss it - **Subscribe by RSS**

Recent Comments

Tiago Bernardo em 404NotFound

Cesar Lemos em 404NotFound

Jefferson Mateus em Entendendo (de verdade) a criptografia RSA

Digite seu endereço de e-mail para assinar este blog e receber notificações de novas publicações por e-mail.

Junte-se a 246 outros assinantes

Autores

Categorias

Agilidade (36)

ALM (239)

Análise de Negócios (14)

Cases (7)

Dados (5)

Desenvolvimento (249)

Empreendedorismo (14)

Gestão (113)

Infra (63)

Inteligência Artificial (4)

Java (11)

JavaScript (1)

Lambda3 (6)

Microsoft (393)

Mobile (38)

Open Source (101)

Outros (105)

Pessoas (8)

Podcast (146)

Práticas de Engenharia (95)

Tecnologia (7)

UX (1)

Web (58)

Xamarin (24)

Arquivos

Selecionar o mês

Tags

.NET .NET Core .NET Magazine Agile ALM Arquitetura ASP.Net ASP.NET Core
 ASP.Net MVC Azure Blog Build C# C#7 Carreira Certificação Cloud DDD
 Democracia organizacional DevOps Diversidade Docker Evento Eventos
 Gestão de projeto git Indicação de conteúdo Javascript liderança Microsoft Mobile
 nodejs Outros podcast podcast não técnico Powershell Scrum SharePoint
 Source Control Testes Automatizados TFS Visual Studio VSTS Webcast xamarin

Lambda3 - Direitos Reservados 2017