عنوان: استفاده از Async و Await در برنامه‌های ASP.NET MVC
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۰۹ ۱۴:۲۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Asynchronous Programming

از ASP.NET MVC 4 به بعد، امکان استفاده از اکشن متدهای async در ASP.NET MVC میسر شده‌است. البته همانطور که پیشتر نیز ذکر شد ، شرط استفاده از امکانات async در نگارش‌های پیش از دات نت 4.5، استفاده از کامپایلری است که بتواند کدهای async را تولید کند و این مورد تنها از VS 2012 به بعد ممکن شده‌است.

علت استفاده از اکشن متدهای async در ASP.NET MVC

اگر نیاز دارید که برنامه‌ی وبی، به شدت مقیاس پذیر را تولید کنید، باید بتوانید مجموعه تردهای سیستم را تا حد ممکن مشغول به کار و سرویس دهی نگه دارید. در برنامه‌های وب ASP.NET تنها تعداد مشخصی ترد، برای پاسخ دهی به درخواست‌های رسیده، همواره مشغول به کار می‌باشند. در اینجا اگر این تردها را برای مدت زمان زیادی جهت اعمال IO مشغول نگه داریم، دست آخر به سیستمی خواهیم رسید که تردهای مفید آن، جهت پایان عملیات مرتبط بیکار شده‌اند و دیگر ASP.NET قادر نیست از آن‌ها جهت پاسخ‌دهی به سایر درخواست‌های رسیده استفاده کند.
برای مثال یک اکشن متد را درنظر بگیرید که نیاز است با یک وب سرویس، برای دریافت نتیجه کار کند. اگر این عملیات اندکی طول بکشد، به همین میزان ترد جاری درحال پردازش این درخواست، بیکار شده و منتظر دریافت پاسخ خواهد ایستاد و اگر به همین ترتیب تعداد تردهای بیکار، بیشتر و بیشتر شوند، دیگر سیستم قادر نخواهد بود به درخواست‌های جدید رسیده پاسخ دهد و ASP.NET مجبور خواهد شد این درخواست‌ها را در صف قرار دهد تا بالاخره زمانی این تردها آزاد شده و قابل استفاده‌ی مجدد گردند. برای رفع این مشکل، استفاده از اعمال غیرهمزمان ابداع گردیدند تا در آن‌ها ترد مورد استفاده جهت پردازش درخواست رسیده را آزاد کرده و به این ترتیب دیگر نیازی نباشد تا تردجاری، تا پایان عملیات IO بلاک شده و بدون استفاده باقی بماند.
در ASP.NET MVC 3 برای نوشتن اکشن متدهای async می‌بایستی از روش قدیمی مدل‌های Async در دات نت مانند APM استفاده می‌شد و همچنین کنترلر جاری بجای ارث بری از کلاس Controller می‌بایستی از کلاس AsyncController مشتق می‌شد. به علت سخت بودن استفاده از آن، این روش و کنترلرهای async در نگاش 3 آن آنچنان مقبولیت و استفاده‌ی گسترده‌ای نیافتند. چون هر اکشن متد تبدیل می‌شد به دو قسمت Begin و End متداول روش‌های APM. سپس در کشن متد دومی، نتیجه‌ی این عملیات به View بازگشت داده می‌شد.
از ASP.NET MVC 4 به بعد، خالی کردن تردهای سیستم و استفاده‌ی مجدد و مشغول به کار نگه داشتن مداوم آن‌ها با استفاده از امکانات توکار زبان‌هایی مانند سی‌شارپ 5، ساده‌تر و خواناتر شده‌است.
البته باید دقت داشت که این بحث صرفا سمت سرور بوده و ارتباطی به مباحث غیرهمزمان سمت کلاینت، مانند Ajax ندارد (A در Ajax نیز به معنای Async است) و از دید مصرف کننده‌ی نهایی، نامرئی می‌باشد. کار Aajx در سمت کلاینت نیز خالی کردن ترد UI مرورگر است (و نه سرور) و در نهایت تهیه‌ی برنامه‌هایی با قابلیت پاسخ‌دهی بهتر.

نوشتن اکشن متدهای Async در ASP.NET MVC

اولین کاری که باید صورت گیرد، اندکی تغییر امضای اکشن متدهای متداول است:

```
public ActionResult Index()
```

این اکشن متد متداول، در یک ترد اجرا شده و این ترد تا پایان کار آن بلاک خواهد شد. برای مثال اگر قرار است مانند قسمت قبل، متد GetStringAsync در آن پردازش شود، تا پایان مدت زمان پردازش این متد، ترد جاری بلاک شده و سیستم قادر به استفاده‌ی مجدد از آن جهت پاسخ‌دهی به سایر درخواست‌های رسیده نخواهد بود. برای تبدیل آن به یک اکشن متد async باید به نحو ذیل عمل کرد:

```
public async Task<ActionResult> Index()
```

ابتدا واژه‌ی کلیدی async به ابتدای امضای متد اضافه می‌شود. سپس خروجی آن اینبار بجای ActionResult، نسخه‌ی جنریک

Task of T خواهد بود. همچنین دیگر نیازی نیست مانند MVC 3، کنترلر جاری از کلاس AsyncController مشتق شود.
زمانیکه به امضای متدی، async اضافه می‌شود، یعنی جایی در داخل بدنه‌ی آن باید await بکار رود:

```
using System.Net.Http;
using System.Threading.Tasks;
using System.Web.Mvc;

namespace Async11.Controllers
{
    public class HomeController : Controller
    {
        public async Task<ActionResult> Index()
        {
            var url = "http://www.dotnettips.info";
            var client = new HttpClient(); // make sure you have an assembly reference to
System.Net.Http.dll
            client.DefaultRequestHeaders.UserAgent.ParseAdd("Test Async");
            var result = await client.GetStringAsync(url);
            return View(result);
        }
    }
}
```

بنابراین اگر داخل اکشن متد جاری، جایی از await استفاده نمی‌شود، async کردن آن بی‌معنا است. این await است که سبب آزاد شدن ترد جاری جهت استفاده‌ی مجدد از آن برای پاسخ‌دهی به سایر درخواست‌های رسیده می‌شود.

یک نکته در مورد WCF 4.5

از WCF 4.5 به بعد، در صفحه‌ی معروف Add service references آن، با کلیک بر روی گزینه‌ی advanced و تنظیمات سرویس، امکان انتخاب گزینه‌ی Create task based operations نیز وجود دارد. این مورد دقیقا برای سهولت استفاده از آن با async و await سی‌شارپ 5 و مدل TAP آن طراحی شده‌است.

تعیین timeout در اکشن متدهای async

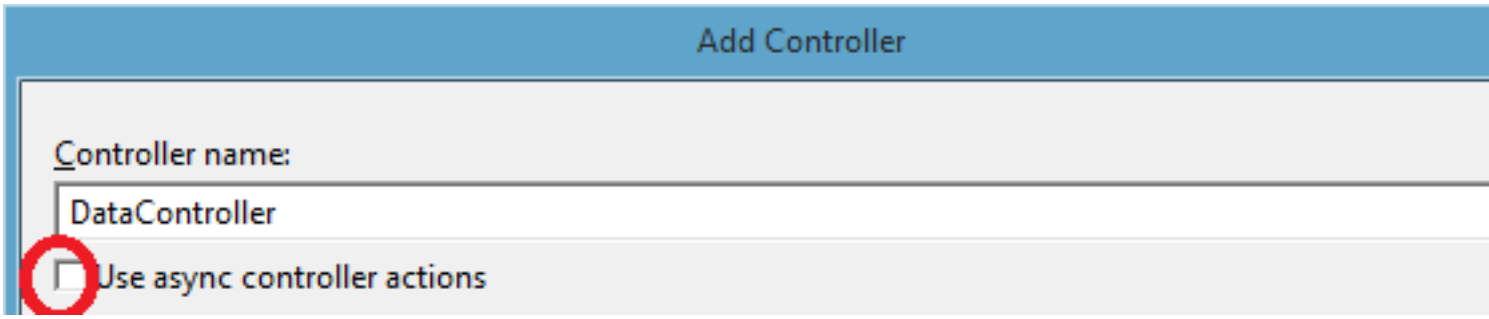
برای مشخص سازی صریح timeout در اکشن متدهای غیرهمزمان، می‌توان از ویژگی خاصی به نام AsyncTimeout به نحو ذیل استفاده کرد:

```
[AsyncTimeout(duration: 1200)]
  public async Task<ActionResult> Index(CancellationToken ct)
```

در مورد لغو اعمال غیرهمزمان پیشتر صحبت شد . در اینجا پارامتر CancellationToken توسط فریم ورک جاری تنظیم شده و می‌توان آن‌را به متدهایی که قادرند اعمال غیر همزمان خود را بر اساس درخواست رسیده CancellationToken لغو کنند، ارسال کرد.

استفاده از قابلیت‌های غیرهمزمان EF 6 به همراه ASP.NET MVC 5

EF 6 به همراه یک سری متد و همچنین متد الحاقی جدید است که اعمال Async را پشتیبانی می‌کنند. اگر در حین انتخاب گزینه‌ی ایجاد کنترلر جدید، گزینه‌ی MVC 5 Controller with views, using EF را انتخاب کنید، امکان تولید اکشن متدهای async نیز به صورت پیش فرض پیش بینی شده‌است:




```
public async Task<ActionResult> Index()
  {
     var model = await db.Books.ToListAsync();
     return View(model);
  }
```

در اینجا نیز امضای اکشن متد، همانند توضیحاتی است که در ابتدای بحث ارائه شد. فقط بجای متد ToList معمولی EF، از نگارش Async آن استفاده شده‌است و همچنین برای دریافت نتیجه‌ی آن از کلمه‌ی کلیدی await استفاده گردیده است.
به علاوه متد Find اکنون معادل FindAsync نیز دارد و همچنین SaveChanges دارای معادل غیرهمزمانی شده‌است به نام SaveChangesAsync .
البته باید دقت داشت که برای Where معادل Async ایی طراحی نشده‌است؛ زیرا نوع IQueryable صرفا یک عبارت است و اجرای آن تا زمانیکه First، ToList و امثال آن فراخوانی نشوند، به تعویق خواهد افتاد.

نظرات خوانندگان

نویسنده: فواد عبداللهی
تاریخ: ۱۳۹۳/۰۱/۱۶ ۱۱:۱۶

سلام؛ اگر

```
var model = await db.Books.ToListAsync();
```

همزمان اجرا میشه ولی بازم برای return باید منتظر پاسخ از db بمونه! پس اینجا فایده ای نداره؟
مشکل من اینجاست که فکر میکنم این روش تنها برای قسمت هایی بدرد میخوره که به هم وابسته نیستن. برای مثال وقتی یه فایل رو آپلود میکنی و بعد آدرس فایل رو ذخیره کنیم فایده نداره. چون تا فایل آپلود نشه ذخیره آدرس تو db بی معنیه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۱۶ ۱۱:۴۲

- پیشنیاز مطالعه قسمت جاری، مطالعه 6 قسمت اول این دوره است.
- «همزمان اجرا میشه»
خیر. متدهای Async واقعی مثل نمونه ارائه شده در EF غیرهمزمان اجرا می‌شوند. یعنی، ترد جاری را آزاد کرده و ASP.NET می‌تواند از آن ترد برای پاسخ دهی به یک درخواست رسیده دیگر استفاده کند.
- «باید منتظر پاسخ از db بمونه»
استفاده از await و async سبب بازنویسی بدنه متد توسط یک state machine در پشت صحنه می‌شوند. یعنی اینطور نیست که روش اجرای آن blocking است و تا رسیدن پاسخ از بانک اطلاعاتی، از این ترد دیگر نمی‌شود استفاده کرد. جایی که await فراخوانی می‌شود، ترد جاری برای استفاده بعدی آزاد خواهد شد. در ادامه مابقی کدها تبدیل به یک IEnumerator می‌شوند که هر دستور آن شامل یک yield return است. هر مرحله که تمام شد، MoveNext این IEnumerator فراخوانی می‌شود تا به مرحله‌ی بعدی برسد. به این روش استفاده از coroutines هم گفته می‌شود که در سی شارپ 5، کامپایلر کار تولید کدهای آن‌را انجام می‌دهد. برای مطالعه بیشتر:
- انجام پی در پی اعمال Async به کمک Iterators - قسمت اول
- انجام پی در پی اعمال Async به کمک Iterators - قسمت دوم
- «چون تا فایل آپلود نشه ذخیره آدرس تو db بی معنیه»
ذخیره آدرس هم یک قسمت از کار است و اتفاقا وابسته به سیستم جاری هم نیست. وابسته است به یک بانک اطلاعاتی که خارج از مرزهای سیستم، به صورت مستقل در حال فعالیت است (عموما البته؛ مثلا اگر از SQL Server استفاده می‌شود).
برای ذخیره فایل‌ها در سیستم هم متدهای Async به کلاس Stream در دات نت 4.5 اضافه شده‌اند؛ مثل WriteAsync . در این حالت هم می‌توان از await WriteAsync برای ذخیره اطلاعات و بازهم آزاد کردن ترد جاری استفاده کرد.