عنوان: آشنایی با مدل برنامه نویسی TAP
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۰۲ ۱۳:۳۹
آدرس: www.dotnettips.info
گروه‌ها: C#, Asynchronous Programming

تاریخچه‌ی اعمال غیر همزمان در دات نت فریم ورک

دات نت فریم ورک، از زمان ارائه نگارش یک آن، از اعمال غیرهمزمان و API خاص آن پشتیبانی می‌کرده‌است. همچنین این مورد یکی از ویژگی‌های Win32 نیز می‌باشد. نوشتن کدهای همزمان متداول بسیار ساده است. در این نوع کدها هر عملیات خاص، پس از پایان عملیات قبلی انجام می‌شود.

```
public string TestNoneAsync()
        {
            var webClient = new WebClient();
            return webClient.DownloadString("http://www.google.com");
        }
```

در این مثال متداول، متد DownloadString به صورت همزمان یا synchronous عمل می‌کند. به این معنا که تا پایان عملیات دریافت اطلاعات از وب، منتظر مانده و ترد جاری را قفل می‌کند. مشکل از جایی آغاز می‌شود که مدت زمان دریافت اطلاعات، طولانی باشد. چون این عملیات در ترد UI در حال انجام است، کل رابط کاربری برنامه تا پایان عملیات نیز قفل شده و دیگر پاسخگوی سایر اعمال رسیده نخواهد بود. در این حالت عموما ویندوز در نوار عنوان برنامه، واژه‌های Not responding را نمایش می‌دهد.
این مورد همچنین در برنامه‌های سمت سرور نیز حائز اهمیت است. با قفل شدن تعداد زیادی ترد در حال اجرا، عملا قدرت پاسخ‌دهی سرور نیز کاهش می‌یابد. بنابراین در این نوع موارد، برنامه‌های چند ریسمانی هرچند در سمت کلاینت ممکن است مفید واقع شوند و برای مثال ترد UI را آزاد کنند، اما اثر آنچنانی بر روی برنامه‌های سمت سرور ندارند. زیرا در آن‌ها می‌توان هزاران ترد را ایجاد کرد که همگی دارای کدهای اصطلاحا blocking باشند. برای حل این مساله استفاده از API غیرهمزمان توصیه می‌شود.
برای نمونه کلاس WebClient توکار دات نت، دارای متدی به نام DownloadStringAsync نیز می‌باشد. این متد به محض فراخوانی، ترد جاری را آزاد می‌کند. به این معنا که فراخوانی آن سبب توقف ترد جاری برای دریافت نتیجه‌ی دریافت اطلاعات از وب نمی‌شود. به این نوع API، یک Asynchronous API گفته می‌شود؛ زیرا با سایر کدهای نوشته شده، هماهنگ و همزمان اجرا نمی‌شود.
هر چند این کد جدید مشکل عدم پاسخ دهی برنامه را برطرف می‌کند، اما مشکل دیگری را به همراه دارد؛ چگونه باید حاصل عملیات آن‌را پس از پایان کار دریافت کرد؟ چگونه باید خطاها و مشکلات احتمالی را مدیریت کرد؟
برای مدیریت این مساله، رخدادی به نام DownloadStringCompleted تعریف شده‌است. روال رویدادگردان آن پس از پایان کار دریافت اطلاعات از وب، فراخوانی می‌گردد.

```
public void TestAsync()
        {
            var webClient = new WebClient();
            webClient.DownloadStringAsync(new Uri("http://www.google.com"));
            webClient.DownloadStringCompleted += webClientDownloadStringCompleted;
        }

        void webClientDownloadStringCompleted(object sender, DownloadStringCompletedEventArgs e)
        {
            // use e.Result
        }
```

در اینجا همچنین توسط آرگومان DownloadStringCompletedEventArgs، موفقیت یا شکست عملیات نیز گزارش می‌شود و مقدار e.Result حاصل عملیات است.

مشکل! ما سادگی یک عملیات همزمان را از دست دادیم. متد TestNoneAsync از لحاظ پیاده سازی و همچنین خواندن و نگهداری آن در طول زمان، بسیار ساده‌تر است از نمونه‌ی TestAsync نوشته شده. در کدهای غیرهمزمان فوق، یک متد ساده، به دو متد مجزا خرد شده‌است و نتیجه‌ی نهایی، درون یک روال رخدادگردان بدست می‌آید.

به این مدل، EAP یا Event based asynchronous pattern نیز گفته می‌شود. EAP در دات نت 2 معرفی شد. روال‌های رخدادگردان در این حالت، در ترد اصلی برنامه اجرا می‌شوند. اما اگر به حالت اصلی اعمال غیرهمزمان موجود از دات نت یک کوچ کنیم، اینطور نیست. در WinForms و WPF برای به روز رسانی رابط کاربری نیاز است اطلاعات دریافت شده در همان تردی که رابط کاربری ایجاد شده است، تحویل گرفته شده و استفاده شوند. در غیراینصورت استثنایی صادر شده و برنامه خاتمه می‌یابد.

آشنایی با Synchronization Context

ابتدا یک برنامه‌ی WinForms ساده را آغاز کرده و یک دکمه‌ی جدید را به نام btnGetInfo و یک تکست باکس را به نام txtResults، به آن اضافه کنید. سپس کدهای فرم اصلی آن‌را به نحو ذیل تغییر دهید:

```
using System;
using System.Linq;
using System.Net;
using System.Windows.Forms;

namespace Async02
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnGetInfo_Click(object sender, EventArgs e)
        {
            var req = (HttpWebRequest)WebRequest.Create("http://www.google.com");
            req.Method = "HEAD";
            req.BeginGetResponse(
                asyncResult =>
                {
                    var resp = (HttpWebResponse)req.EndGetResponse(asyncResult);
                    var headersText = formatHeaders(resp.Headers);
                    txtResults.Text = headersText;
                }, null);
        }

        private string formatHeaders(WebHeaderCollection headers)
        {
            var headerString = headers.Keys.Cast<string>()
                                        .Select(header => string.Format("{0}:{1}", header,
headers[header]));
            return string.Join(Environment.NewLine, headerString.ToArray());
        }
    }
}
```

در اینجا از روش دیگری برای دریافت اطلاعات از وب استفاده کرده‌ایم. با استفاده از امکانات HttpWebRequest، کوئری‌های پیشرفته‌تری را می‌توان تهیه کرد. برای مثال می‌توان نوع متد را به HEAD تنظیم نمود؛ تا صرفا مقادیر هدر آدرس درخواستی از سرور، دریافت شوند.

همچنین در این مثال از متد غیرهمزمان BeginGetResponse نیز استفاده شده‌است. در این نوع API خاص، کار با BeginGetResponse آغاز شده و سپس در callback نهایی توسط EndGetResponse، نتیجه‌ی عملیات به دست می‌آید.

اگر برنامه را اجرا کنید، با استثنای زیر مواجه خواهید شد:

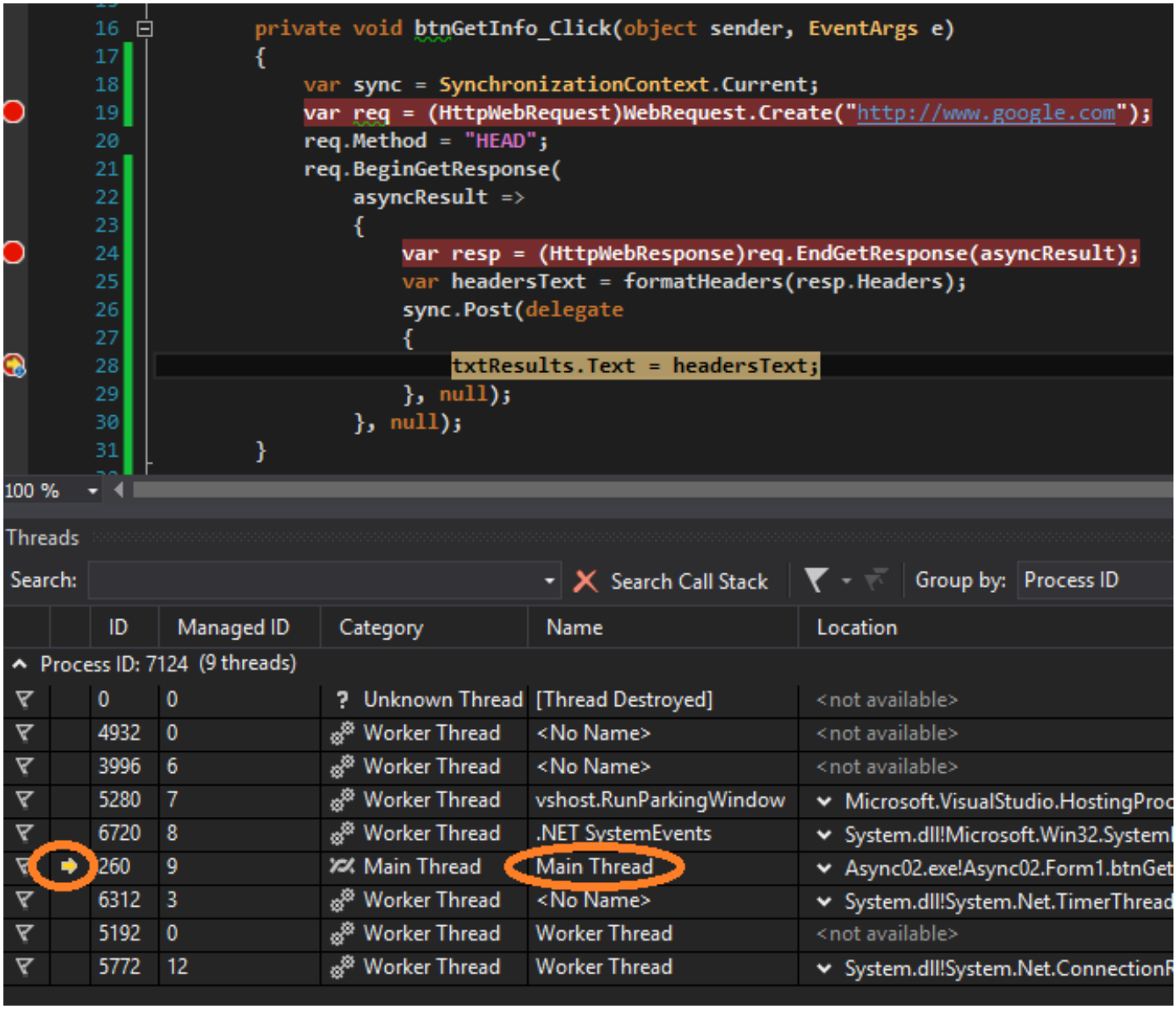
```
An exception of type 'System.InvalidOperationException' occurred in System.Windows.Forms.dll but was
not handled in user code
Additional information: Cross-thread operation not valid: Control 'txtResults' accessed from a thread
other than the thread it was created on.
```

علت اینجا است که asyncResult دریافتی، در تردی دیگر نسبت به ترد اصلی برنامه که UI را اداره می‌کند، اجرا می‌شود. یکی از راه حل‌های این مشکل و انتقال اطلاعات به ترد اصلی برنامه، استفاده از Synchronization Context است:

```
private void btnGetInfo_Click(object sender, EventArgs e)
        {
```

```
        var sync = SynchronizationContext.Current;
        var req = (HttpWebRequest)WebRequest.Create("http://www.google.com");
        req.Method = "HEAD";
        req.BeginGetResponse(
            asyncResult =>
            {
                var resp = (HttpWebResponse)req.EndGetResponse(asyncResult);
                var headersText = formatHeaders(resp.Headers);
                sync.Post(delegate { txtResults.Text = headersText; }, null);
            }, null);
    }
```

SynchronizationContext.Current در اینجا چون در ابتدای متد دریافت اطلاعات اجرا می‌شود، به ترد UI، یا ترد اصلی برنامه اشاره می‌کند. به همین جهت این زمینه را نباید داخل Async callback دریافت کرد؛ زیرا ترد جاری آن، ترد UI مدنظر ما نیست. سپس همانطور که ملاحظه می‌کنید، توسط متد Post آن می‌توان اطلاعات را در زمینه‌ی تردی که SynchronizationContext به آن اشاره می‌کند اجرا کرد.



برای درک بهتر آن، سه break point را پیش از متد BeginGetResponse، داخل Async calback و داخل delegate متد Post قرار

دهید. پس از اجرای برنامه، از منوی دیباگ در VS.NET گزینه‌ی Windows و سپس Threads را انتخاب کنید.
در اینجا همانطور که مشخص است، کد داخل delegate تعریف شده، در ترد اصلی برنامه اجرا می‌شود و نه یکی از Worker threadهای ثانویه.
هر چند استفاده از متدهای تو در تو و lambda syntax، نیاز به تعریف چندین متد جداگانه را برطرف کرده‌است، اما باز هم کد ساده‌ای به نظر نمی‌رسد. در سی شارپ 5، برای مدیریت بهتر تمام مشکلات یاد شده، پشتیبانی توکاری از اعمال غیرهمزمان، به هسته‌ی زبان اضافه شده‌است.

Syntax ابتدایی یک متد Async

در ابتدا کلاس و متد Async زیر را در نظر بگیرید:

```
using System;
using System.Threading.Tasks;

namespace Async01
{
    public class AsyncExample
    {
        public async Task DoWorkAsync(int parameter)
        {
            await Task.Delay(parameter);
            Console.WriteLine(parameter);
        }
    }
}
```

شیوه‌ی نگارش آن بر اساس راهنمای نوشتن برنامه‌های Async یا Task asynchronous programming model یا به اختصار TAP است:
- در مدل برنامه نویسی TAP، متدهای غیرهمزمان باید یک Task را بازگشت دهند؛ یا نمونه‌ی جنریک آن‌را. البته کامپایلر، async void را نیز پشتیبانی می‌کند ولی در قسمت‌های بعدی بررسی خواهیم کرد که چرا استفاده از آن مشکل‌زا است و باید از آن پرهیز شود.
- همچنین مطابق TAP، اینگونه متدها باید به پسوند Async ختم شوند تا استفاده کننده در حین کار با Intellisense، بتواند آن‌ها را از متدهای معمولی سریعتر تشخیص دهد.
- از واژه‌ی کلیدی async نیز استفاده می‌گردد تا کامپایلر از وجود اعمال غیر همزمان مطلع گردد.
- await به کامپایلر می‌گوید، عبارت پس از من، یک وظیفه‌ی غیرهمزمان است و ادامه‌ی کدهای نوشته شده، تنها زمانی باید اجرا شوند که عملیات غیرهمزمان معرفی شده، تکمیل گردد.

در متد DoWorkAsync، ابتدا به اندازه‌ای مشخص توقف حاصل شده و سپس سطر بعدی یعنی Console.WriteLine اجرا می‌شود.

یک اشتباه عمومی! استفاده از واژه‌های کلیدی async و await متد شما را async نمی‌کنند.

برخلاف تصور ابتدایی از بکارگیری واژه‌های کلیدی async و await، این کلمات نحوه‌ی اجرای متد شما را async نمی‌کنند. این کلمات صرفا برای تشکیل متدهایی که هم اکنون غیرهمزمان هستند، مفید می‌باشند. برای توضیح بیشتر آن به مثال ذیل دقت کنید:

```
public async Task<double> GetNumberAsync()
        {
            var generator = new Random();
            await Task.Delay(generator.Next(1000));

            return generator.NextDouble();
        }
```

در این متد با استفاده از Task.Delay، انجام یک عملیات طولانی شبیه سازی شده‌است؛ مثلا دریافت یک عدد یا نتیجه از یک وب سرویس. سپس در نهایت، عددی را بازگشت داده است. برای بازگشت یک خروجی double، در اینجا از نمونه‌ی جنریک Task استفاده شده‌است.

در ادامه برای استفاده از آن خواهیم داشت:

```
public async Task<double> GetSumAsync()
        {
            var leftOperand = await GetNumberAsync();
            var rightOperand = await GetNumberAsync();

            return leftOperand + rightOperand;
        }
```

خروجی این متد تنها زمانی بازگشت داده می‌شود که نتایج leftOperand و rightOperand از وب سرویس فرضی، دریافت شده باشند و در اختیار مصرف کننده قرارگیرند. بنابراین همانطور که ملاحظه می‌کنید از واژه‌ی کلیدی await جهت تشکیل یک عملیات غیرهمزمان و مدیریت ساده‌تر کدهای نهایی، شبیه به کدهای معمولی همزمان استفاده شده‌است.
در کدهای همزمان متداول، سطر اول ابتدا انجام می‌شود و بعد سطر دوم و الی آخر. با استفاده از واژه‌ی کلیدی await یک چنین عملکردی را با اعمال غیرهمزمان خواهیم داشت. پیش از این برای مدیریت اینگونه اعمال از یک سری callback و یا رخداد استفاده می‌شد. برای مثال ابتدا عملیات همزمانی شروع شده و سپس نتیجه‌ی آن در یک روال رخداد گردان جایی در کدهای برنامه دریافت می‌شد (مانند مثال ابتدای بحث). اکنون تصور کنید که قصد داشتید جمع نهایی حاصل دو عملیات غیرهمزمان را از دو روال رخدادگردان جدا از هم، جمع آوری کرده و بازگشت دهید. هرچند اینکار غیرممکن نیست، اما حاصل کار به طور قطع آنچنان زیبا نبوده و قابلیت نگهداری پایینی دارد. واژه‌ی کلیدی await، انجام اینگونه امور غیرهمزمان را طبیعی و همزمان جلوه می‌دهد. به این ترتیب بهتر می‌توان بر روی منطق و الگوریتم‌های مورد استفاده تمرکز داشت، تا اینکه مدام درگیر مکانیک اعمال غیرهمزمان بود.

امکان استفاده از واژه‌ی کلیدی await در هر جایی از کدها وجود دارد. برای نمونه در مثال زیر، برای ترکیب دو عملیات غیرهمزمان، از await در حین تشکیل عملیات ضرب نهایی، دقیقا در جایی که مقدار متد باید بازگشت داده شود، استفاده شده‌است:

```
public async Task<double> GetProductOfSumAsync()
        {
            var leftOperand = GetSumAsync();
            var rightOperand = GetSumAsync();

            return await leftOperand * await rightOperand;
        }
```

اگر await را از این مثال حذف کنیم، خطای کامپایل زیر را دریافت خواهیم کرد:

```
Operator '*' cannot be applied to operands of type 'System.Threading.Tasks.Task<double>' and
'System.Threading.Tasks.Task<double>'
```

خروجی متد GetSumAsync صرفا یک Task است و نه یک عدد. پس از استفاده از await، عملیات آن انجام شده و بازگشت داده می‌شود.

اگر متد DownloadString همزمان ابتدای بحث را نیز بخواهیم تبدیل به نمونه‌ی async سی‌شارپ 5 کنیم، می‌توان از متد الحاقی جدید آن به نام DownloadStringTaskAsync کمک گرفت:

```
public async Task<string> DownloadAsync()
        {
            var webClient = new WebClient();
            return await webClient.DownloadStringTaskAsync("http://www.google.com");
        }
```

نکته‌ی مهم این کد علاوه بر ساده سازی اعمال غیر همزمان، برای استفاده از نتیجه‌ی نهایی آن، نیازی به SynchronizationContext معرفی شده در تاریخچه‌ی ابتدای بحث نیست. نتیجه‌ی دریافتی از آن در ترد اصلی برنامه تحویل داده شده و به سادگی قابل استفاده است.

سؤال: آیا استفاده از await نیز ترد جاری را قفل می‌کند؟

اگر به کدها دقت کنید، استفاده از await به معنای صبر کردن تا پایان عملیات async است. پس اینطور به نظر می‌رسد که در اینجا نیز ترد اصلی، همانند قبل قفل شده‌است.

```
public void TestDownloadAsync()
        {
            Debug.WriteLine("Before DownloadAsync");
            DownloadAsync();
            Debug.WriteLine("After DownloadAsync");
        }
```

اگر این متد را اجرا کنید (در آن await بکار نرفته)، بلافاصله خروجی ذیل را مشاهده خواهید کرد:

```
Before DownloadAsync
After DownloadAsync
```

به این معنا که در اصل، همانند سایر روش‌های async موجود از دات نت یک، در اینجا نیز فراخوانی متد async ترد اصلی را بلافاصله آزاد می‌کند و ترد آن‌را قفل نخواهد کرد. استفاده از await نیز عملکرد کدها را تغییر نمی‌دهد. تنها کامپایلر در پشت صحنه همان کدهای لازم جهت مدیریت روال‌های رخدادگردان و callbackها را تولید می‌کند، به نحوی که صرفا نحوه‌ی کدنویسی ما همزمان به نظر می‌رسد، اما در پشت صحنه، نحوه‌ی اجرای آن غیرهمزمان است.

برنامه‌های Async و نگارش‌های مختلف دات نت

شاید در ابتدا به نظر برسد که قابلیت‌های جدید async و await صرفا متعلق هستند به دات نت 4.5 به بعد؛ اما خیر. اگر کامپایلری را داشته باشید که از این واژه‌های کلیدی را پشتیبانی کند، امکان استفاده از آن‌ها را با دات نت 4 نیز خواهید داشت. برای این منظور تنها کافی است از VS 2012 به بعد استفاده نمائید. سپس در کنسول پاورشل نیوگت دستور ذیل را اجرا نمائید (فقط برای برنامه‌های دات نت 4 البته):

```
PM> Install-Package Microsoft.Bcl.Async
```

این روال متداول VS.NET بوده است تا به امروز. برای مثال اگر VS 2010 را نصب کنید و سپس یک برنامه‌ی دات نت 3.5 را ایجاد کنید، امکان استفاده‌ی کامل از تمام امکانات سی‌شارپ 4، مانند آرگومان‌های نامدار و یا مقادیر پیش فرض آرگومان‌ها را در یک برنامه‌ی دات نت 3.5 نیز خواهید داشت. همین نکته در مورد async نیز صادق است. VS 2012 (یا نگارش‌های جدیدتر) را نصب کنید و سپس یک پروژه‌ی دات نت 4 را آغاز کنید. امکان استفاده از async و await را خواهید داشت. البته در این حالت دسترسی به متدهای الحاقی جدید را مانند DownloadStringTaskAsync نخواهید داشت. برای رفع این مشکل باید بسته‌ی Microsoft.Bcl.Async را نیز توسط نیوگت نصب کنید.

نظرات خوانندگان

نویسنده: علی رضایی
تاریخ: ۱۳۹۳/۰۱/۰۲ ۱۸:۶

سلام
بسیار بسیار تشکر برای آموزش این بحث جالب.
یک سوال:
موضوع کاربردی که من از این مطلب فهمیدم به شکل زیر است، لطفاً اگر اشتباه است بفرمایید:
در پروژهٔ واقعی که حجم دیتابیس زیاد میشود، ممکن است اندکی زمان برای ذخیره اطلاعات، جستجو و غیره لازم باشد که این باعث عدم پاسخ سرور به سایر درخواست‌ها میشود، حال با استفاده از Async مثلاً در زمان context.SaveChanges این مشکل رفع شده و پس از ثبت اطلاعات آی دی رکورد جدید برگشت داده میشود.
ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۰۲ ۱۸:۳۱

استفاده از async به معنای خالی کردن ترد جاری کدهای مدیریت شده‌ی دات نت است و انجام سایر کارهای برنامه و صبر کردن برای دریافت پاسخی است که در سمت کدهای مدیریت شده نیازی به پردازش و محاسبه ندارد.
برای مثال در حالت کار با یک دیتابیس، این موتور بانک اطلاعاتی است که کوئری رسیده را پردازش می‌کند و برنامه‌ی ما صرفا درخواستی را به آن ارائه داده است. به این ترتیب در اینجا استفاده از async برای خالی کردن ترد جاری و صبر کردن جهت دریافت نتیجه‌ی اطلاعات از سرور مفید است و میزان پاسخ‌دهی برنامه را بالا می‌برد.
بنابراین استفاده از async در سمت کدهای دات نتی، تاثیری بر روی عملکرد یک بانک اطلاعاتی ندارد. فقط در سمت کدهای ما است که برنامه تا رسیدن و محاسبه‌ی درخواست توسط بانک اطلاعاتی، هنگ نمی‌کند.
در این حالت اگر برنامه‌ی شما ویندوزی است، ترد UI آن آزاد شده و برنامه مدام در حال هنگ به نظر نمی‌رسد. اگر برنامه‌ی وب است، ترد جاری آن آزاد شده و thread pool برنامه می‌تواند از این ترد آزاد شده، برای پردازش سایر درخواست‌های رسیده توسط کاربران استفاده کند. به این ترتیب بازدهی و اصطلاحا throughput سرور افزایش پیدا می‌کند.
در حال حاضر تمام APIهای جدید مایکروسافت نسخه‌ی async را هم اضافه کرده‌اند. برای مثال اگر از EF استفاده می‌کنید، از نسخه‌ی 6 آن به بعد، متدهایی مانند ToListAsync برای کوئری گرفتن معمولی غیرهمزمان و SaveChangesAsync برای ذخیره سازی اطلاعات به صورت غیرهمزمان، اضافه شده‌اند. یک مثال کامل در این مورد در اینجا
البته بدیهی است تمام ORMهای دات نتی در سطح پایین خودشان از ADO.NET استفاده می‌کنند. ADO.NET نیز Async API سازگار با دات نت 4.5 به بعد را مدتی است که اضافه کرده‌است. برای مثال متدهایی مانند GetFieldValueAsync ، ExecuteReaderAsync و ExecuteNonQueryAsync و امثال آن به زیر ساخت ADO.NET اضافه شده‌اند. اطلاعات بیشتر

نویسنده: شهروز جعفری
تاریخ: ۱۳۹۳/۰۱/۰۸ ۱۶:۴۷

من یکم گیج شدم:شما فرمودید که : - await به کامپایلر می‌گوید، عبارت پس از من، یک وظیفه‌ی غیرهمزمان است و ادامه‌ی کدهای نوشته شده، تنها زمانی باید اجرا شوند که عملیات غیرهمزمان معرفی شده، تکمیل گردد.
این آیا بدان معنا نیست که ترد اصلی برنامه باید قفل شود؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۰۸ ۱۷:۰

خیر. در پشت صحنه از یک ماشین حالت (state machine) برای پیاده سازی async استفاده می‌کند. کل سطرهای بعدی 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

نویسنده: مصطفی عسگری
تاریخ: ۱۳۹۳/۰۱/۰۹ ۱۳:۴۵

سلام
من متد DownloadStringAsync و رویداد مرتبط با آن یعنی DownloadStringCompleted رو تست کردم و به دلیل اینکه متد DownloadStringAsync را در UI Thread صدا میزدم رویداد DownloadStringCompleted نیز همیشه در UI Thread فراخوانی میشد.
من در یک پروژه یک کتابخانه درست کرده بودم که یکی از متدها باید کاری رو به صورت Async انجام میداد و وقتی که کار این متد تمام میشد نتیجه را با Raise کردن یک event به اطلاع استفاده کننده میرسوندم.
اما مشکل اینجا بود که به کنترل‌های روی فرم دسترسی نداشتم و داخل این رویداد ابتدا شرط InvokeRequired و سپس Invoke رو نوشته بودم.این کار مشکل رو حل کرده بود.
اما به نظر من این کار درست نیست.چون من در واقع یکسری API نوشته ام و در اختیار برنامه نویسان دیگر گذاشته ام و آنها باید بتوانند کدهای خود را بدون InvokeRequired درون رویداد بنویسند.
آیا راهی هست که بشه متد من در هر Thread یی اجرا شود رویداد اتمام آن نیز در همان Thread صدازننده ، فراخوانی شود؟
من در این کتابخانه از async و await استفاده نکرده بودم.
ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۰۹ ۱۳:۵۵

- SynchronizationContext از دات نت 2 در دسترس است. بنابراین اجازه دهید مصرف کننده از متد Post آن در صورت صلاحدید، در هر جایی که لازم داشت برای ارسال نتیجه‌ی دریافتی به تردی خاص، مثلا ترد UI استفاده کند. در این مورد در مطلب « استفاده از Async و Await در برنامه‌های دسکتاپ » بیشتر بحث شده‌است.
- SynchronizationContext.Current را اگر پیش از آغاز ترد دریافت کنید، به ترد جاری فراخوان اشاره می‌کند. در پایان ترد، می‌توانید از متد Post آن برای بازگشت به ترد قبلی کمک بگیرید.