عنوان: پردازش‌های Async در Entity framework 6
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۲۹ ۰:۲۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, Asynchronous Programming

اجرای Async اعمال نسبتا طولانی، در برنامه‌های مبتنی بر داده، عموما این مزایا را به همراه دارد:

الف) مقیاس پذیری سمت سرور

در اعمال سمت سرور متداول، تردهای متعددی جهت پردازش درخواست‌های کلاینت‌ها تدارک دیده می‌شوند. هر زمانیکه یکی از این تردها، یک عملیات blocking را انجام می‌دهد (مانند دسترسی به شبکه یا اعمال I/O)، ترد مرتبط با آن تا پایان کار این عملیات معطل خواهد شد. با بالا رفتن تعداد کاربران یک برنامه و در نتیجه بیشتر شدن تعداد درخواست‌هایی که سرور باید پردازش کند، تعداد تردهای معطل مانده نیز به همین ترتیب بیشتر خواهند شد. مشکل اصلی اینجا است که نمونه سازی تردها بسیار هزینه بر است (با اختصاص 1MB of virtual memory space) و منابع سرور محدود. با زیاد شدن تعداد تردهای معطل اعمال I/O یا شبکه، سرور مجبور خواهد شد بجای استفاده مجدد از تردهای موجود، تردهای جدیدی را ایجاد کند. همین مساله سبب بالا رفتن بیش از حد مصرف منابع و حافظه برنامه می‌گردد. یکی از روش‌های رفع این مشکل بدون نیاز به بهبودهای سخت افزاری، تبدیل اعمال blocking نامبرده شده به نمونه‌های non-blocking است. به این ترتیب ترد پردازش کننده‌ی این اعمال Async بلافاصله آزاد شده و سرور می‌تواند از آن جهت پردازش درخواست دیگری استفاده کند؛ بجای اینکه ترد جدیدی را وهله سازی نماید.

ب) بالا بردن پاسخ دهی کلاینت‌ها

کلاینت‌ها نیز اگر مدام درخواست‌های blocking را به سرور جهت دریافت پاسخی ارسال کنند، به زودی به یک رابط کاربری غیرپاسخگو خواهند رسید. برای رفع این مشکل نیز می‌توان از توانمندی‌های Async دات نت 4.5 جهت آزاد سازی ترد اصلی برنامه یا همان ترد UI استفاده کرد.

و ... تمام این‌ها یک شرط را دارند. نیاز است یک چنین API خاصی که اعمال Async واقعی را پشتیبانی می‌کنند، فراهم شده باشد. بنابراین صرفا وجود متد Task.Run، به معنای اجرای واقعی Async یک متد خاص نیست. برای این منظور ADO.NET 4.5 به همراه متدهای Async ویژه کار با بانک‌های اطلاعاتی است و پس از آن Entity framework 6 از این زیر ساخت استفاده کرده‌است که در ادامه جزئیات آن‌را بررسی خواهیم کرد.

پیشنیازها

برای کار با امکانات جدید Async موجود در EF 6 نیاز است از VS 2012 به بعد که به همراه کامپایلری است که واژه‌های کلیدی async و await را پشتیبانی می‌کند و همچنین دات نت 4.5 استفاده کرد. چون ADO.NET 4.5 اعمال async واقعی را پشتیبانی می‌کند، دات نت 4 در اینجا قابل استفاده نخواهد بود.

متدهای الحاقی جدید Async در EF 6.x

جهت متدهای الحاقی متداول EF مانند Min، Max، ToList و غیره، نمونه‌های Async آن‌ها نیز اضافه شده‌اند:

```
QueryableExtensions:
AllAsync
AnyAsync
AverageAsync
ContainsAsync
CountAsync
FirstAsync
FirstOrDefaultAsync
ForEachAsync
LoadAsync
LongCountAsync
MaxAsync
```

```
MinAsync
SingleAsync
SingleOrDefaultAsync
SumAsync
ToArrayAsync
ToDictionaryAsync
ToListAsync

DbSet:
FindAsync

DbContext:
SaveChangesAsync

Database:
ExecuteSqlCommandAsync
```

بنابراین اولین قدم تبدیل کدهای قدیمی به Async، استفاده از متدهای الحاقی فوق است.

چند مثال

فرض کنید، مدل‌های برنامه، رابطه‌ی one-to-many ذیل را بین یک کاربر و مقالات او دارند:

```
public class User
    {
        public int Id { get; set; }
        public string Name { get; set; }

        public virtual ICollection<BlogPost> BlogPosts { get; set; }
    }

    public class BlogPost
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        [ForeignKey("UserId")]
        public virtual User User { get; set; }
        public int UserId { get; set; }
    }
```

همچنین Context برنامه نیز جهت در معرض دید قرار دادن این کلاس‌ها، به نحو ذیل تشکیل شده‌است:

```
public class MyContext : DbContext
    {
        public DbSet<User> Users { get; set; }
        public DbSet<BlogPost> BlogPosts { get; set; }

        public MyContext()
            : base("Connection1")
        {
            this.Database.Log = sql => Console.Write(sql);
        }
    }
```

بر این اساس مثالی که دو رکورد را در جداول کاربران و مقالات به صورت async ثبت می‌کند، به نحو ذیل خواهد بود:

```
private async Task<User> addUserAsync(CancellationToken cancellationToken = default(CancellationToken))
        {
            using (var context = new MyContext())
            {
                var user = context.Users.Add(new User
                {
                    Name = "Vahid"
                });
                context.BlogPosts.Add(new BlogPost
                {
                    Content = "Test",
                    Title = "Test",
```

```
                User = user
            });
            await context.SaveChangesAsync(cancellationToken);
            return user;
        }
    }
```

چند نکته جهت یادآوری مباحث Async

- به امضای متد واژه‌ی کلیدی async اضافه شده‌است، زیرا در بدنه‌ی آن از کلمه‌ی کلیدی await استفاده کرده‌ایم (لازم و ملزوم هستند).
- به انتهای نام متد، کلمه‌ی Async اضافه شده‌است. این مورد ضروری نیست؛ اما به یک استاندارد و قرارداد تبدیل شده‌است.
- مدل Async دات نت 4.5 مبتنی بر Taskها است. به همین جهت اینبار خروجی‌های توابع نیاز است از نوع Task باشند و آرگومان جنریک آن‌ها، بیانگر نوع مقداری که باز می‌گردانند.
- تمام متدهای الحاقی جدیدی که نامبرده شدند، دارای پارامتر اختیاری لغو عملیات نیز هستند. این مورد را با مقدار دهی cancellationToken در کدهای فوق ملاحظه می‌کنید.

نمونه‌ای از نحوه‌ی مقدار دهی این پارامتر در ASP.NET MVC به صورت زیر می‌تواند باشد:

```
[AsyncTimeout(8000)]
public async Task<ActionResult> Index(CancellationToken cancellationToken)
```

در اینجا به امضای اکشن متد جاری، async اضافه شده‌است و خروجی آن نیز به نوع Task تغییر یافته است. همچنین یک پارامتر cancellationToken نیز تعریف شده‌است. این پارامتر به صورت خودکار توسط ASP.NET MVC پس از زمانیکه توسط ویژگی AsyncTimeout تعیین شده‌است، تنظیم خواهد شد. به این ترتیب، اعمال async در حال اجرا به صورت خودکار لغو می‌شوند.
- برای اجرا و دریافت نتیجه‌ی متدهای Async دار EF، نیاز است از واژه‌ی کلیدی await استفاده گردد.

استفاده کننده نیز می‌تواند متد addUserAsync را به صورت زیر فراخوانی کند:

```
var user = await addUserAsync();
Console.WriteLine("user id: {0}", user.Id);
```

شبیه به همین اعمال را نیز جهت به روز رسانی و یا حذف اطلاعات خواهیم داشت:

```
private async Task<User> updateAsync(CancellationToken cancellationToken = default(CancellationToken))
        {
            using (var context = new MyContext())
            {
                var user1 = await context.Users.FindAsync(cancellationToken, 1);
                if (user1 != null)
                    user1.Name = "Vahid N.";

                await context.SaveChangesAsync(cancellationToken);
                return user1;
            }
        }

        private async Task<int> deleteAsync(CancellationToken cancellationToken =
default(CancellationToken))
        {
            using (var context = new MyContext())
            {
                var user1 = await context.Users.FindAsync(cancellationToken, 1);
                if (user1 != null)
                    context.Users.Remove(user1);

                return await context.SaveChangesAsync(cancellationToken);
            }
        }
```

کدهای Async تقلبی!

به قطعه کد ذیل دقت کنید:

```
public async Task<List<TEntity>> GetAllAsync()
   {
    return await Task.Run(() => _tEntities.ToList());
   }
```

این متد از یکی از Generic repositoryهای فله‌ای رها شده در اینترنت انتخاب شده‌است.
به این نوع متدها که از Task.Run برای فراخوانی متدهای همزمان قدیمی مانند ToList جهت Async جلوه دادن آن‌ها استفاده می‌شود، کدهای Async تقلبی می‌گویند! این عملیات هر چند در یک ترد دیگر انجام می‌شود اما هم سربار ایجاد یک ترد جدید را به همراه دارد و هم عملیات ToList آن کاملا blocking است.
معادل صحیح Async واقعی این عملیات را در ذیل مشاهده می‌کنید:

```
private async Task<List<User>> getUsersAsync(CancellationToken cancellationToken =
default(CancellationToken))
        {
            using (var context = new MyContext())
            {
                return await context.Users.ToListAsync(cancellationToken);
            }
        }
```

متد ToListAsync یک متد Async واقعی است و نه شبیه سازی شده توسط Task.Run. متدهای Async واقعی کار با شبکه و اعمال I/O، از ترد استفاده نمی‌کنند و توسط سیستم عامل به نحو بسیار بهینه‌ای اجرا می‌گردند.
برای مثال پشت صحنه‌ی متد الحاقی SaveChangesAsync به یک چنین متدی ختم می‌شود:

```
internal override async Task<long> ExecuteAsync(
//...
rowsAffected = await
command.ExecuteNonQueryAsync(cancellationToken).ConfigureAwait(continueOnCapturedContext: false);
//...
```

متد ExecuteNonQueryAsync جزو متدهای ADO.NET 4.5 است و برای اجرا نیاز به هیچ ترد جدیدی ندارد.
و یا برای شبیه سازی ToListAsync با ADO.NET 4.5 و استفاده از متدهای Async واقعی آن، به یک چنین کدهایی نیاز است:

```
var connectionString = ".........";
    var sql = @"......"";
    var users = new List<User>();

    using (var cnx = new SqlConnection(connectionString))
    {
      using (var cmd = new SqlCommand(sql, cnx))
      {
       await cnx.OpenAsync();
       using (var reader = await cmd.ExecuteReaderAsync(CommandBehavior.CloseConnection))
       {
        while (await reader.ReadAsync())
        {
          var user = new User
          {
           Id = reader.GetInt32(0),
           Name = reader.GetString(1),
          };
         users.Add(user);
        }
       }
      }
    }
```

محدودیت پردازش موازی اعمال در EF

در متد ذیل، دو Task غیرهمزمان تعریف شده‌اند و سپس با await Task.WhenAll درخواست اجرای همزمان و موازی آن‌ها را کرده‌ایم:

```
// multiple operations
        private static async Task loadAllAsync(CancellationToken cancellationToken =
default(CancellationToken))
        {
            using (var context = new MyContext())
            {
                var task1 = context.Users.ToListAsync(cancellationToken);
                var task2 = context.BlogPosts.ToListAsync(cancellationToken);

                await Task.WhenAll(task1, task2);
                // use task1.Result
            }
        }
```

این متد ممکن است اجرا شود؛ یا در بعضی از مواقع با استثنای ذیل خاتمه یابد:

```
An unhandled exception of type 'System.NotSupportedException' occurred in mscorlib.dll
 Additional information: A second operation started on this context before a previous asynchronous
operation completed.
Use 'await' to ensure that any asynchronous operations have completed before calling another method on
this context.
Any instance members are not guaranteed to be thread safe.
```

متن استثنای ارائه شده بسیار مفید است و توضیحات کامل را به همراه دارد. در EF در طی یک Context اگر عملیات Async شروع شده‌ای خاتمه نیافته باشد، مجاز به شروع یک عملیات Async دیگر، به موازت آن نخواهیم بود. برای رفع این مشکل یا باید از چندین Context استفاده کرد و یا await Task.WhenAll را حذف کرده و بجای آن واژه‌ی کلیدی await را همانند معمول، جهت صبر کردن برای دریافت نتیجه‌ی یک عملیات غیرهمزمان استفاده کنیم.

نظرات خوانندگان

نویسنده: میثم ثوامری
تاریخ: ۱۳۹۳/۰۳/۲۹ ۱:۰

با تشکر ازشما.
میخواستم بدونم متدهای async چطور در یک repository استفاده کنم
بطور مثال:

```
public class ProductRepository<T> where T : class
    {
        protected DbContext _context;

        public ProductRepository(DataContext context)
        {
            _context = context;
        }

       // GetAll
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۲۹ ۱:۲۳

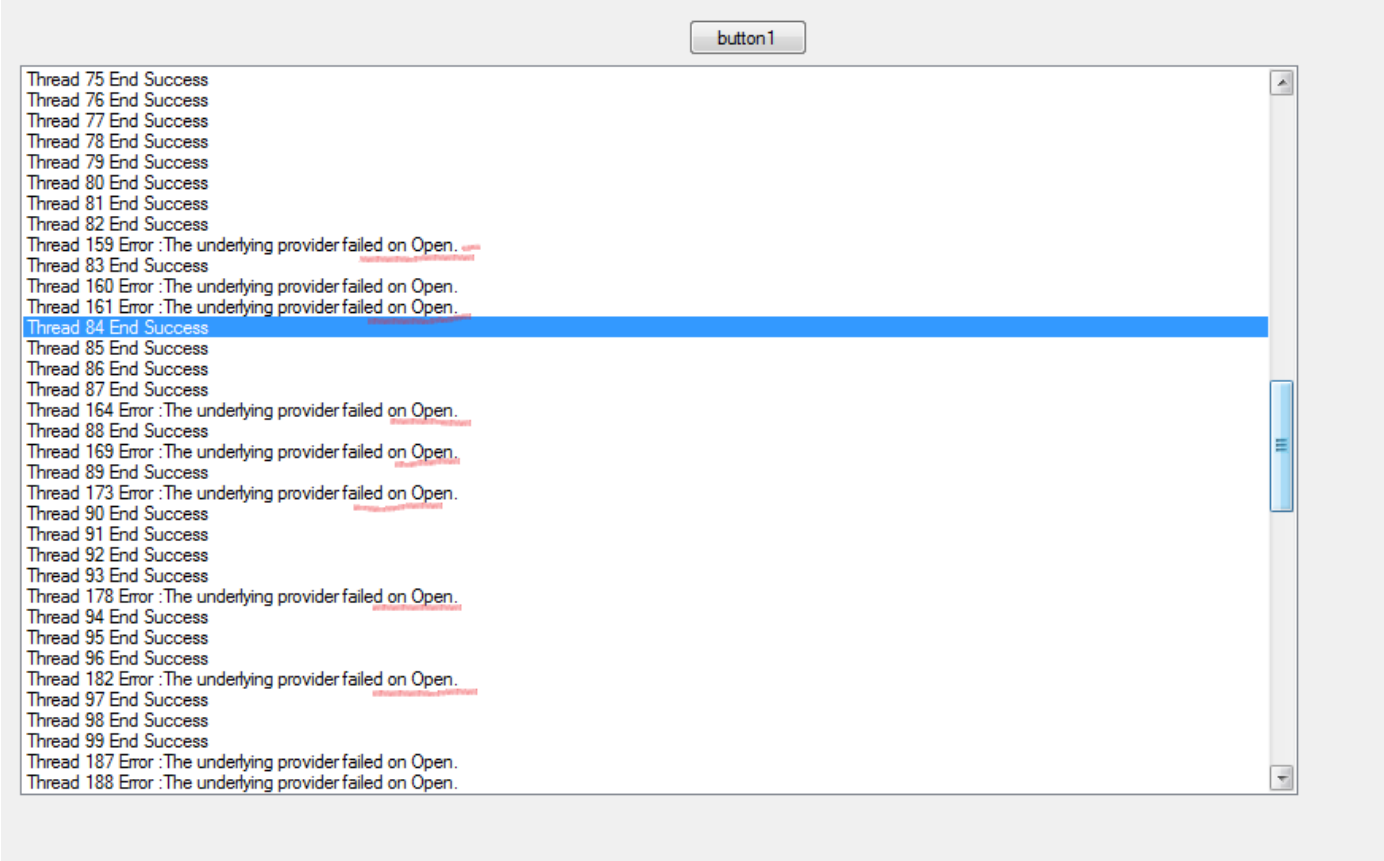
از متدهای الحاقی جدید Async که نامبرده شدند استفاده کنید (بجای متدهای قدیمی معادل) به همراه Set برای دستیابی به موجودیت‌ها؛ مثلا:

```
public async Task<List<T>> GetAllAsync()
    {
        return await _dbContext.Set<T>().ToListAsync();
    }
```

نویسنده: ج.زوسر
تاریخ: ۱۳۹۳/۰۴/۰۸ ۹:۴۹

مثلا برای همچین کدی میشه از روش بالا استفاده کرد مشکل حل میشه ؟

```
{
            for (int i = 1; i <=500; i++)
            {
                ThreadPool.QueueUserWorkItem(Execute, i + 1);
            }
        }
void Execute(Object obj)
        {
             int thread = (int)obj;
            try
            {
                using (TestEntities ctx = new TestEntities())
                {
                    int i = 1;
                    foreach (var v in ctx.Customers)
                    {
                        Thread.Sleep(i * 1000);
                        i++;
                    }
                }
                listBox1.Items.Add("Thread " + thread.ToString() + " End Success ");
            }
            catch (Exception e)
            {
                listBox1.Items.Add("Thread " + thread.ToString() + " Error :" + e.Message);
            }
        }
```



نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۴/۰۸ ۱۰:۴۴

- بحث متدهای Async اضافه شده، ریطی به مباحث چند ریسمانی ندارد. «... متدهای *Async واقعی* کار با شبکه و اعمال I/O، از ترد استفاده نمی‌کنند ... » به همین جهت نسبت به حالت استفاده از تردها سربار کمتری دارند.
- در EF استثناءها چند سطحی هستند. نیاز است `inner exception` را جهت مشاهده‌ی اصل و علت واقعی خطا بررسی کرد. در مثال شما فقط سطح استثناء بررسی شده و نه اصل آن.
احتمالا خطای اصلی `timeout` است. این مورد به مباحث قفل گذاری روی رکوردها مرتبط است. تراکنش‌های طولانی همزمانی را آغاز کرده‌اید که دسترسی سایر کاربران را به جداول، تا پایان کار آن تراکنش‌ها، محدود می‌کنند.
- در کارهای چند ریسمانی برای دسترسی امن به عناصر UI، باید از روش‌های `Synchronization` استفاده کرد.

نویسنده: هرمز
تاریخ: ۱۳۹۳/۰۴/۰۸ ۱۴:۵۶

سلام؛ متاسفانه من نمیتونم متود ToListAsync رو پیدا کنم. آیا باید ریفرنس خاصی اضافه کنم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۴/۰۸ ۱۵:۳

- به روز رسانی خودکار وابستگی‌های پروژه:

```
PM> update-package
```

- تعریف فضای نام مرتبط:

```
using  System.Data.Entity;
```

نویسنده: رضایی
تاریخ: ۱۳۹۳/۰۸/۰۹ ۱۸:۲

با سلام؛ من از کد زیر استفاده کردم اما همواره خطا میده.

توی serviceLayer :

```
public async Task<IList<NewsSliderModel>> GetNewsSliderTable(CancellationToken cancellationToken =
default(CancellationToken))
        {
            IQueryable<News> selectednews = _news.Take(_sliderTakeCount).AsQueryable();
             ...
             ...
            return await selectednews.Select(x => new NewsSliderModel
            {
                NewsId = x.Id,
                Title = x.Title,
                ImagePath = x.ImagePath,
                ImageTitle = x.ImageTitle,
                ImageDescription = x.ImageDescription,
            }).ToListAsync();

        }
```

و توی کنترلر Home برنامه

```
public virtual async Task<ActionResult> MainSlider()
        {
            IList<NewsSliderModel> SliderList = await _newsService.GetNewsSliderTable(Order.Descending,
NewsOrderBy.Id);
            return PartialView(MVC.Home.Views._MainSlider, SliderList.ToList());
        }
```

اما همواره خطای زیر رو میده

```
HttpServerUtility.Execute blocked while waiting for an asynchronous operation to complete.
```

ممنون میشم راهنمایی فرمایید

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۰۹ ۱۸:۱۷

- اگر از اکشن متد MainSlider به صورت child action استفاده می‌شود (مثلا حین فراخوانی Html.Action یا Html.RenderAction)، این فراخوانی حتما باید همزمان باشد و حالت غیرهمزمان آن پشتیبانی نمی‌شود .
- این محدودیت در نگارش بعدی ASP.NET MVC (نگارش 6 آن) برطرف شده‌است.

نویسنده: رضایی
تاریخ: ۱۳۹۳/۰۸/۰۹ ۱۸:۳۲

از EntityFramework 6.0 استفاده میکنم. همانطور که فرمودید خطا از این خط:

```
@Html.Action(MVC.Home.ActionNames.MainSlider, MVC.Home.Name)
```

راه حل چیه؟ از لینک بالا چیزی دستگیرم نشد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۰۹ ۱۸:۳۷

عرض کردم. این مورد خاص در نگارش فعلی ASP.NET MVC (تا قبل از نگارش 6)، راه حلی ندارد. معمولی کار کنید؛ مانند قبل (خروجی ActionResult بجای async Task<ActionResult>).