عنوان: **نحوه کاهش مصرف حافظه EF Code first حین گزارشگیری از اطلاعات**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۰۶ ۱۹:۶
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

تمام ORMهای خوب، دارای سطح اول کش هستند. از این سطح جهت نگهداری اطلاعات تغییرات صورت گرفته روی اشیاء و سپس اعمال نهایی آن‌ها در پایان یک تراکنش استفاده می‌شود. بدیهی است جمع آوری این اطلاعات اندکی بر روی سرعت انجام کار و همچنین بر روی میزان مصرف حافظه برنامه تاثیرگذار است. به علاوه یک سری از اعمال مانند گزارشگیری نیازی به این سطح اول کش ندارند. اطلاعات مورد استفاده در آن‌ها مانند نمایش لیستی از اطلاعات در یک گرید، حالت فقط خواندنی دارد. در EF Code first برای یک چنین مواردی استفاده از متد الحاقی AsNoTracking تدارک دیده شده است که سبب خاموش شدن سطح اول کش می‌شود. در ادامه در طی یک مثال، اثر این متد را بر روی سرعت و میزان مصرف حافظه برنامه بررسی خواهیم کرد.

کدهای کامل این مثال را در ذیل ملاحظه می‌کنید:

```
using System;
using System.Data.Entity;
using System.Data.Entity.Migrations;
using System.Diagnostics;
using System.Linq;

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

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

    public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(MyContext context)
        {
            for (int i = 0; i < 21000; i++)
            {
                context.Users.Add(new User { Name = "name " + i });
                if (i % 1000 == 0)
                    context.SaveChanges();
            }
            base.Seed(context);
        }
    }

    public class PerformanceHelper
    {
        public static string RunActionMeasurePerformance(Action action)
        {
            GC.Collect();
            long initMemUsage = Process.GetCurrentProcess().WorkingSet64;

            var stopwatch = new Stopwatch();
            stopwatch.Start();

            action();

            stopwatch.Stop();

            var currentMemUsage = Process.GetCurrentProcess().WorkingSet64;
            var memUsage = currentMemUsage - initMemUsage;
            if (memUsage < 0) memUsage = 0;
```

```
            return string.Format("Elapsed time: {0}, Memory Usage: {1:N2} KB", stopwatch.Elapsed,
memUsage / 1024);
        }
    }

    public static class Test
    {
        public static void RunTests()
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyContext, Configuration>());
            StartDb();

            for (int i = 0; i < 3; i++)
            {
                Console.WriteLine("\nRun {0}", i + 1);

                var memUsage = PerformanceHelper.RunActionMeasurePerformance(() => LoadWithTracking());
                Console.WriteLine("LoadWithTracking:\n{0}", memUsage);

                memUsage = PerformanceHelper.RunActionMeasurePerformance(() => LoadWithoutTracking());
                Console.WriteLine("LoadWithoutTracking:\n{0}", memUsage);
            }
        }

        private static void StartDb()
        {
            using (var ctx = new MyContext())
            {
                var user = ctx.Users.Find(1);
                if (user != null)
                {
                    // keep the object in memory
                }
            }
        }

        private static void LoadWithTracking()
        {
            using (var ctx = new MyContext())
            {
                var list = ctx.Users.ToList();
                if (list.Any())
                {
                    // keep the list in memory
                }
            }
        }

        private static void LoadWithoutTracking()
        {
            using (var ctx = new MyContext())
            {
                var list = ctx.Users.AsNoTracking().ToList();
                if (list.Any())
                {
                    // keep the list in memory
                }
            }
        }
    }
}
```

توضیحات:

مدل برنامه یک کلاس ساده کاربر است به همراه id و نام او.

سپس این کلاس توسط Context برنامه در معرض دید EF Code first قرار می‌گیرد.

در کلاس Configuration تعدادی رکورد را در ابتدای کار برنامه در بانک اطلاعاتی ثبت خواهیم کرد. قصد داریم میزان مصرف حافظه بارگذاری این اطلاعات را بررسی کنیم.

کلاس PerformanceHelper معرفی شده، دو کار اندازه گیری میزان مصرف حافظه برنامه در طی اجرای یک فرمان خاص و همچنین مدت زمان سپری شدن آن‌را اندازه‌گیری می‌کند.

در کلاس Test فوق چندین متد به شرح زیر وجود دارند:

متد StartDb سبب می‌شود تا تنظیمات ابتدایی برنامه به بانک اطلاعاتی اعمال شوند. تا زمانیکه کوئری خاصی به بانک اطلاعاتی

ارسال نگردد، EF Code first بانک اطلاعاتی را آغاز نخواهد کرد.
در متد LoadWithTracking اطلاعات تمام رکوردها به صورت متداولی بارگذاری شده است.
در متد LoadWithoutTracking نحوه استفاده از متد الحاقی AsNoTracking را مشاهده می‌کنید. در این متد سطح اول کش به این ترتیب خاموش می‌شود.
و متد RunTests، این متدها را در سه بار متوالی اجرا کرده و نتیجه عملیات را نمایش خواهد داد.

برای نمونه این نتیجه در اینجا حاصل شده است:

```
Run 1
LoadWithTracking:
Elapsed time: 00:00:00.9332636, Memory Usage: 8,528.00 KB
LoadWithoutTracking:
Elapsed time: 00:00:00.3119418, Memory Usage: 256.00 KB

Run 2
LoadWithTracking:
Elapsed time: 00:00:00.3611471, Memory Usage: 4,100.00 KB
LoadWithoutTracking:
Elapsed time: 00:00:00.1590219, Memory Usage: 256.00 KB

Run 3
LoadWithTracking:
Elapsed time: 00:00:00.3750008, Memory Usage: 4,332.00 KB
LoadWithoutTracking:
Elapsed time: 00:00:00.1593993, Memory Usage: 216.00 KB
```

همانطور که ملاحظه کنید، بین این دو حالت، تفاوت بسیار قابل ملاحظه است؛ چه از لحاظ مصرف حافظه و چه از لحاظ سرعت.

نتیجه گیری:
اگر قصد ندارید بر روی اطلاعات دریافتی از بانک اطلاعاتی تغییرات خاصی را انجام دهید و فقط قرار است از آن‌ها به صورت فقط خواندنی گزارشگیری شود، بهتر است سطح اول کش را به کمک متد الحاقی AsNoTracking خاموش کنید.

نظرات خوانندگان

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۰/۲۳ ۱۷:۸

سلام.
وقتی از متد Where و ... برای فیلتر کردن استفاده کنیم دیگه قادر به انجام این کار (خاموش کردن سطح اول کش) نیستیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۲۳ ۱۷:۱۵

```
// Query for all users without tracking them
var users = context.Users.AsNoTracking();

// Query for some users without tracking them
var someUsers = context.Users
                       .Where(u => u.Name.EndsWith("st"))
                       .AsNoTracking()
                       .ToList();
//Or ...
var items = context.Users.AsNoTracking().Where(...);
```

نویسنده: احمد ولی پور
تاریخ: ۱۳۹۱/۱۲/۰۹ ۱۹:۴۴

با سلام
اگر کش سطح اول رو غیر فعال کنیم و توی صفحه عمل Update یا Delete انجام بدیم چه اتفاقی رخ میده؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۰۹ ۲۱:۲۸

با استفاده از متد AsNoTracking و یا غیرفعال سازی کلی این فرآیند توسط تنظیم context.Configuration.AutoDetectChangesEnabled = false ، شیء از context جدا شده در نظر گرفته می‌شود. بنابراین context از تغییرات شما بی‌خبر بوده و ... اتفاق خاصی رخ نخواهد داد.

نویسنده: پدرام جباری
تاریخ: ۱۳۹۱/۱۲/۱۰ ۲۳:۵۲

اگر نیاز دارید مدل رو از یک Context جدا کنید (کش کردن اون رو غیر فعال کنید) باید توجه داشته باشید که غبر فعال کردن AutoDetectChangesEnabled کافی نیست باید متد AsNoTracking رو هم استفاده کنید ، مخصوصا برای زمانی که لازم داشته باشید در یک شی دیگه از Context اون مدل رو Attach کنید، اگر هر دو رو غیر فعال نکنید Attach کردن مدل (بسته به پیچیدگی مدل) زمانی تا 5 ثانیه یا حتی بیشتر میبره.

غیر فعال کردن کلی AutoDetectChangesEnabled بیشتر زمانی که می‌خواهید رکورد به دیتابیس اضافه کنید بسیار مورد نیاز هست، سرعت رو به مقدار قابل توجهی افزایش میده (البته برای تعداد رکورد بالا تاثیر خودش رو نشون میده)
برای آپدیت و حذف رکورد ، اگر از وجود رکورد اطمینان دارید (مخصوصا برای ویرایش مدل) بهتر هست مدل رو به Context ای که دارید Attach کنید که خوب بهتر از Select زدن از دیتابیس هست

نویسنده: ahmad.valipour
تاریخ: ۱۳۹۲/۰۲/۱۴ ۲۰:۴۸

با سلام
متد بالا رو برای DataBase First هم میشه استفاده کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۱۴ ۲۱:۵۳

برای حالت استفاده مستقیم از ObjectContext:

```
var context = new NorthwindDataContext();
context.tblCities.MergeOption = MergeOption.NoTracking;
```

واقعیت این است که یک EF بیشتر وجود خارجی ندارد. سورس EF هم در دسترس است :

```
public virtual IInternalQuery<TElement> AsNoTracking()
{
   return (IInternalQuery<TElement>) new InternalQuery<TElement>(this._internalContext, (ObjectQuery)
DbHelpers.CreateNoTrackingQuery((ObjectQuery) this._objectQuery));
}

public static IQueryable CreateNoTrackingQuery(ObjectQuery query)
{
  IQueryable queryable = (IQueryable) query;
  ObjectQuery objectQuery = (ObjectQuery) queryable.Provider.CreateQuery(queryable.Expression);
  objectQuery.MergeOption = MergeOption.NoTracking; // اینجا کار خاموش سازی ردیابی انجام شده
  return (IQueryable) objectQuery;
}
```

همانطور که مشاهده می‌کنید، متد الحاقی AsNoTracking در پشت صحنه همان کار تنظیم MergeOption = MergeOption.NoTracking رو انجام می‌ده.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۱۹ ۱۲:۴۶

اهمیت استفاده از AsNoTracking زمانیکه نیازی به آن نیست:

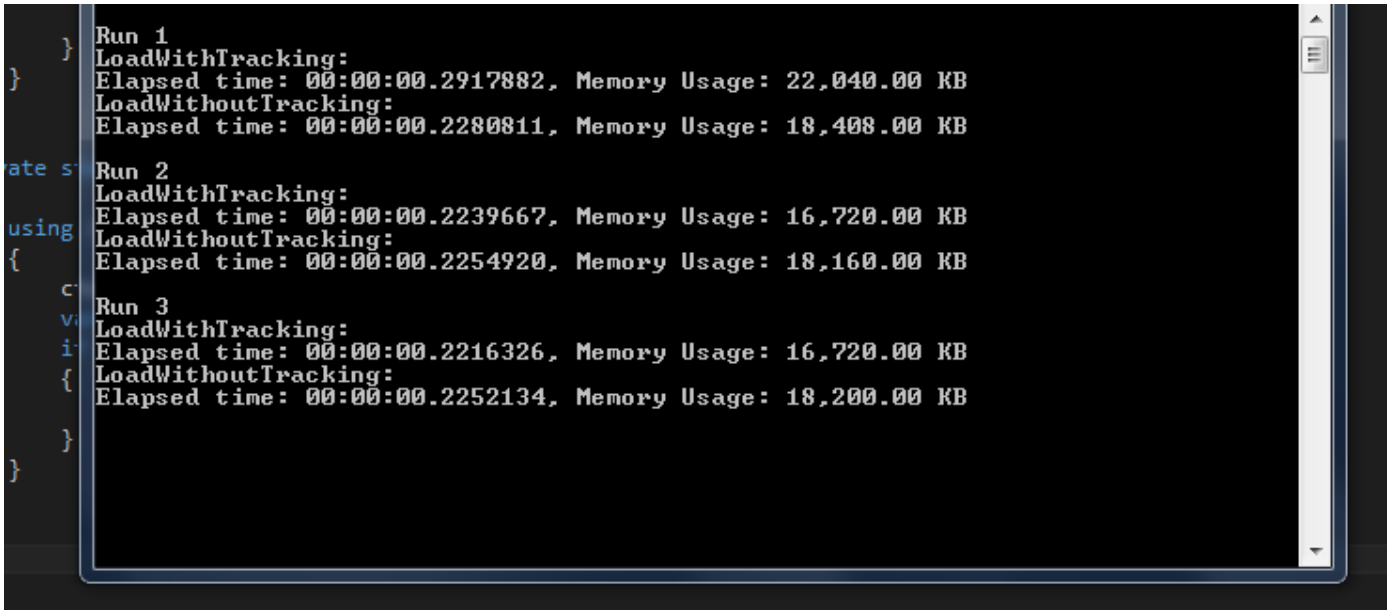
Fetch performance of various .NET ORM / Data-access frameworks

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۹/۲۳ ۹:۵

آیا ممکن است AsNoTracking برای همه‌ی Queryها فعال شود ؟ یعنی کلا Change Tracker , سطح اول Cache خاموش شود.
در مثال پست جاری تابع LoadWithoutTracking را با کد زیر جایگزین کردم :

```
private static void LoadWithoutTracking()
        {
            using (var ctx = new MyContext())
            {
                ctx.Configuration.AutoDetectChangesEnabled = false;
                var list = ctx.Users.ToList();
                if (list.Any())
                {
                    // keep the list in memory
                }
            }
        }
```

با توجه نتیجه‌ی حاصل شده به نظر مصرف حافظه بهبود نیافته :



نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۳ ۱۰:۴۵

با توجه به سورس EF ، در متد CreateNoTrackingQuery کار MergeOption.NoTracking به ازای یک ObjectSet (و نه حتی DbSet) انجام می‌شود و الزاما معادل نیست با AutoDetectChangesEnabled = false تنها.

نویسنده: رضا گرمارودی
تاریخ: ۱۳۹۲/۱۱/۲۸ ۱۷:۲۸

سلام؛ من اینجا و اینجا و اینجا را ... مطالعه کردم. اما متوجه نشدم در زمان گزارش گیری با هر ابزاری (Stmulsoft, FastReport, ..) اطلاعات باید به صورت یک BusinesObject و یا هر عنوان دیگه ای به ابزار مورد نظر در قالب یک IEnumeable ارسال شود. در حالی که اگر ما IQueable را با .ToList() به یک IEnumerable تبدیل شود، ممکن است در برگیرنده کل اطلاعات باشد. در این موارد راهی برای کاهش حافظه و سربار کم وجود دارد؟ متد ToList را نمی‌توان به صورت Lazy پیاده سازی کرد یعنی اگر ابزار گزارش ساز فرضا صفحه 1 را نمایش دهد اطلاعات تا صفحه یک از بانک واکشی بشود. اگر گزارش ما 200 صفحه باشد در حالت عادی کل اطلاعات در سرور لود شده و برنامه‌های گزارش ساز صرفا پس از تهیه گزارش اطلاعات را به صورت صفحه بندی نمایش می‌دهند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۸ ۱۷:۴۲

مورد مدنظر شما اصطلاحا paging نام دارد و در گزارش گیری‌های خصوصا برنامه‌های تحت وب که گرید نهایی را برنامه نویس با کدنویسی و ارائه منبع داده مناسبی طراحی و پیاده سازی می‌کند، بسیار مرسوم است (یک Take و Skip است در سمت کوئری LINQ نوشته شده). مثلا:

« واکشی اطلاعات به صورت chunk chunk (تکه تکه) و نمایش در ListView »

این قابلیت اگر در نرم افزارهای گزارشگیری یاد شده، پیاده سازی شده‌است (مانند مثال یاد شده *MaximumRows* و *StartRowIndex* را هربار در اختیار برنامه نویس قرار می‌دهند)، آنگاه قابل استفاده و پیاده سازی خواهد بود. در غیراینصورت، کار خاصی را نمی‌توان انجام داد و باید مطابق نیاز تجاری آن‌ها رفتار کرد.

نویسنده: رضا گرمارودی
تاریخ: ۱۳۹۲/۱۱/۳۰ ۱۰:۱۳

سلام؛ ممنون از پاسختون. من تمام برنامه‌های گزارش سازی را که میشناختم (Telerik, Fastreport, ReportViewer, DevExpress) همه را بررسی کردم . اما هیچ کدام چنین پراپرتی و یا مشابه اون و نداشتند. یعنی هرکسی می‌خواد گزارش بگیرد یک دفعه کل اطلاعات و از بانک می‌خونه! هرچقدر هم با فیلترهای مختلف گزارش و با فیلترهای مختلف مثلا تاریخ یا شماره رکورد محدود کنیم اما در کل کاربر در هر لحظه یک صفحه را که بیشتر نمی‌تواند ببیند. مباحث مذکور به خوبی در انواع گرید و کنترل‌های مختلف پیاده سازی شده و شرکت‌های مختلف راه حل‌های مختلفی همانند مواردی که شما ذکر کردید ارائه کرده اند اما برای گزارش خیر !