عنوان: **Implementing second level caching in EF code first**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۵ ۱۱:۱۱
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

هدف اصلی از انواع و اقسام مباحث caching اطلاعات، فراهم آوردن روش‌هایی جهت میسر ساختن دسترسی سریعتر به داده‌هایی است که به صورت متناوب در برنامه مورد استفاده قرار می‌گیرند، بجای مراجعه مستقیم به بانک اطلاعاتی و خواندن اطلاعات از دیسک سخت.

عموما در ORMها دو سطح کش می‌تواند وجود داشته باشد:

الف) سطح اول کش

که نمونه بارز آن در EF Code first استفاده از متد context.Entity.Find است. در بار اول فراخوانی این متد، مراجعه‌ای به بانک اطلاعاتی صورت گرفته تا بر اساس primary key ذکر شده در آرگومان آن، رکورد متناظری بازگشت داده شود. در بار دوم فراخوانی متد Find، دیگر مراجعه‌ای به بانک اطلاعاتی صورت نخواهد گرفت و اطلاعات از سطح اول کش (یا همان Context جاری) خوانده می‌شود.
بنابراین سطح اول کش در طول عمر یک تراکنش معنا پیدا می‌کند و به صورت خودکار توسط EF مدیریت می‌شود.

ب) سطح دوم کش

سطح دوم کش در ORMها طول عمر بیشتری داشته و سراسری است. هدف از آن کش کردن اطلاعات عمومی و پر مصرفی است که در دید تمام کاربران قرار دارد و همچنین تمام کاربران می‌توانند به آن دسترسی داشته باشند. بنابراین محدود به یک Context نیست.
عموما پیاده سازی سطح دوم کش خارج از ORM مورد استفاده قرار می‌گیرد و توسط اشخاص و شرکت‌های ثالث تهیه می‌شود.
در حال حاضر پیاده سازی توکاری از سطح دوم کش در EF Code first وجود ندارد و قصد داریم در مطلب جاری به یک پیاده سازی نسبتا خوب از آن برسیم.

تلاش‌های صورت گرفته

تا کنون دو پیاده سازی نسبتا خوب از سطح دوم کش در EF صورت گرفته:

Entity Framework Code First Caching
Caching the results of LINQ queries

مورد اول برای ایده گرفتن خوب است. بحث اصلی پیاده سازی سطح دوم کش، یافتن کلیدی است که معادل کوئری LINQ در حال فراخوانی است. سطح دوم کش را به صورت یک Dictionary تصور کنید. هر آیتم آن تشکیل شده است از یک کلید و یک مقدار. از کلید برای یافتن مقدار متناظر استفاده می‌شود.
اکنون مشکل چیست؟ در یک برنامه ممکن است صدها کوئری لینک وجود داشته باشد. چطور باید به ازای هر کوئری LINQ یک کلید منحصربفرد تولید کرد؟
در مطلب « Entity Framework Code First Caching » از متد ToString استفاده شده است. اگر این متد، بر روی یک عبارت LINQ در EF Code first فراخوانی شود، معادل SQL آن نمایش داده می‌شود. بنابراین یک قدم به تولید کلید منحصربفرد متناظر با یک کوئری نزدیک شده‌ایم. اما ... مشکل اینجا است که متد ToString پارامترها را لحاظ نمی‌کند. بنابراین این روش اصلا قابل استفاده نیست. چون کاربر به ازای تمام پارامترهای ارسالی، همواره یک نتیجه را دریافت خواهد کرد.
در مقاله « Caching the results of LINQ queries » این مشکل برطرف شده است. با parse کامل یک expression tree عبارت LINQ کلید منحصربفرد معادل آن یافت می‌شود. سپس بر این اساس می‌توان نتیجه کوئری را به نحو صحیحی کش کرد. در این روش پارامترها هم لحاظ می‌شوند و مشکل مقاله قبلی را ندارد.
اما این مقاله دوم یک مشکل مهم را به همراه دارد: روشی را برای حذف آیتم‌ها از کش ارائه نمی‌دهد. فرض کنید مقالات سایت را در سطح دوم کش قرار داده‌اید. اکنون یک مقاله جدید در سایت ثبت شده است. اصطلاحا برای invalidating کش در این روش،

راهکاری پیشنهاد نشده است.

پیاده سازی بهتری از سطح دوم کش در EF Code fist

می‌توان از همان روش یافتن کلید منحصربفرد معادل با یک کوئری LINQ، که در مقاله دوم فوق، یاد شد، کار را شروع کرد و سپس آن‌را به مرحله‌ای رساند که مباحث حذف کش نیز به صورت خودکار مدیریت شود. پیاده سازی آن را برای برنامه‌های وب در ذیل ملاحظه می‌کنید:

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Data.Objects;
using System.Diagnostics;
using System.Linq;
using System.Web;
using System.Web.Caching;

namespace EfSecondLevelCaching.Core
{
    public static class EfHttpRuntimeCacheProvider
    {
        #region Methods (6)

        // Public Methods (2)

        public static IList<TEntity> ToCacheableList<TEntity>(
                            this IQueryable<TEntity> query,
                            int durationMinutes = 15,
                            CacheItemPriority priority = CacheItemPriority.Normal)
        {
            return query.Cacheable(x => x.ToList(), durationMinutes, priority);
        }

        /// <summary>
        /// Returns the result of the query; if possible from the cache, otherwise
        /// the query is materialized and the result cached before being returned.
        /// The cache entry has a one minute sliding expiration with normal priority.
        /// </summary>
        public static TResult Cacheable<TEntity, TResult>(
                            this IQueryable<TEntity> query,
                            Func<IQueryable<TEntity>, TResult> materializer,
                            int durationMinutes = 15,
                            CacheItemPriority priority = CacheItemPriority.Normal)
        {
            // Gets a cache key for a query.
            var queryCacheKey = query.GetCacheKey();

            // The name of the cache key used to clear the cache. All cached items depend on this key.
            var rootCacheKey = typeof(TEntity).FullName;

            // Try to get the query result from the cache.
            printAllCachedKeys();
            var result = HttpRuntime.Cache.Get(queryCacheKey);
            if (result != null)
            {
                debugWriteLine("Fetching object '{0}__{1}' from the cache.", rootCacheKey,
queryCacheKey);
                return (TResult)result;
            }

            // Materialize the query.
            result = materializer(query);

            // Adding new data.
            debugWriteLine("Adding new data: queryKey={0}, dependencyKey={1}", queryCacheKey,
rootCacheKey);
            storeRootCacheKey(rootCacheKey);
            HttpRuntime.Cache.Insert(
                    key: queryCacheKey,
                    value: result,
                    dependencies: new CacheDependency(null, new[] { rootCacheKey }),
                    absoluteExpiration: DateTime.Now.AddMinutes(durationMinutes),
                    slidingExpiration: Cache.NoSlidingExpiration,
```

```
                    priority: priority,
                    onRemoveCallback: null);

            return (TResult)result;
        }

        /// <summary>
        /// Call this method in `public override int SaveChanges()` of your DbContext class
        /// to Invalidate Second Level Cache automatically.
        /// </summary>
        public static void InvalidateSecondLevelCache(this DbContext ctx)
        {
            var changedEntityNames = ctx.ChangeTracker
                                        .Entries()
                                        .Where(x => x.State == EntityState.Added ||
                                                    x.State == EntityState.Modified ||
                                                    x.State == EntityState.Deleted)
                                        .Select(x =>
ObjectContext.GetObjectType(x.Entity.GetType()).FullName)
                                        .Distinct()
                                        .ToList();

            if (!changedEntityNames.Any()) return;

            printAllCachedKeys();
            foreach (var item in changedEntityNames)
            {
                item.removeEntityCache();
            }
            printAllCachedKeys();
        }
        // Private Methods (4)

        private static void debugWriteLine(string format, params object[] args)
        {
            if (!Debugger.IsAttached) return;
            Debug.WriteLine(format, args);
        }

        private static void printAllCachedKeys()
        {
            if (!Debugger.IsAttached) return;
            debugWriteLine("Available cached keys list:");
            int count = 0;
            var enumerator = HttpRuntime.Cache.GetEnumerator();
            while (enumerator.MoveNext())
            {
                if (enumerator.Key.ToString().StartsWith("__")) continue; // such as
__System.Web.WebPages.Deployment
                debugWriteLine("queryKey: {0}", enumerator.Key.ToString());
                count++;
            }
            debugWriteLine("count: {0}", count);
        }

        private static void removeEntityCache(this string rootCacheKey)
        {
            if (string.IsNullOrWhiteSpace(rootCacheKey)) return;
            debugWriteLine("Removing items with dependencyKey={0}", rootCacheKey);
            // Removes all cached items depend on this key.
            HttpRuntime.Cache.Remove(rootCacheKey);
        }

        private static void storeRootCacheKey(string rootCacheKey)
        {
            // The cacheKeys of a cacheDependency that are not already in cache ARE NOT inserted into
the cache
            // on the Insert of the item in which the dependency is used.
            if (HttpRuntime.Cache.Get(rootCacheKey) != null)
                return;

            HttpRuntime.Cache.Add(
                rootCacheKey,
                rootCacheKey,
                null,
                Cache.NoAbsoluteExpiration,
                Cache.NoSlidingExpiration,
                CacheItemPriority.Default,
                null);
        }
```

```
        #endregion Methods
    }
}
```

توضیحات کدهای فوق

در اینجا یک متدالحاقی به نام Cacheable توسعه داده شده است که می‌تواند در انتهای کوئری‌های LINQ شما قرار گیرد. مثلا:

```
var data = context.Products.AsQueryable().Cacheable(x => x.FirstOrDefault());
```

کاری که در این متد انجام می‌شود به این شرح است:
الف) ابتدا کلید منحصربفرد معادل کوئری LINQ فراخوانی شده محاسبه می‌شود.
ب) بر اساس نام کامل نوع Entity در حال استفاده، کلید دیگری به نام rootCacheKey تولید می‌گردد.
شاید بپرسید اهمیت این کلید چیست؟
فرض کنید در حال حاضر 1000 آیتم در کش وجود دارند. چه روشی را برای حذف آیتم‌های مرتبط با کش Entity1 پیشنهاد می‌دهید؟ احتمالا خواهید گفت تمام کش را بررسی کرده و آیتم‌ها را یکی یکی حذف می‌کنیم.
این روش بسیار کند است (و جواب هم نمی‌دهد؛ چون کلیدی که در اینجا تولید شده، هش MD5 معادل کوئری است و نمی‌توان آن‌را به موجودیتی خاص ربط داد) و ... نکته جالبی در متد HttpRuntime.Cache.Insert برای مدیریت آن پیش بینی شده است: استفاده از CacheDependency.
توسط CacheDependency می‌توان گروهی از آیتم‌های هم‌خانواده را تشکیل داد. سپس برای حذف کل این گروه کافی است کلید اصلی CacheDependency را حذف کرد. به این ترتیب به صورت خودکار کل کش مرتبط خالی می‌شود.
ج) مراحل بعدی آن هم یک سری اعمال متداول هستند. ابتدا توسط HttpRuntime.Cache.Get بررسی می‌شود که آیا بر اساس کلید متناظر با کوئری جاری، اطلاعاتی در کش وجود دارد یا خیر. اگر بله، نتیجه از کش خوانده می‌شود. اگر خیر، کوئری اصطلاحا materialized می‌شود تا بر روی بانک اطلاعاتی اجرا شده و نتیجه بازگشت داده شود. سپس این نتیجه را در کش قرار می‌دهیم.

مورد بعدی که باید به آن دقت داشت، خالی کردن کش، پس از به روز رسانی اطلاعات توسط کاربران است. این کار در متد InvalidateSecondLevelCache صورت می‌گیرد. به کمک ChangeTracker می‌توان نام نوع‌های موجودیت‌های تغییر کرده را یافت. چون کلید اصلی CacheDependency را بر مبنای همین نام نوع‌های موجودیت‌ها تعیین کرده‌ایم، به سادگی می‌توان کش مرتبط با موجودیت یافت شده را خالی کرد.
استفاده از متد InvalidateSecondLevelCache یاد شده به نحو زیر است:

```
using System.Data.Entity;
using EfSecondLevelCaching.Core;
using EfSecondLevelCaching.Test.Models;

namespace EfSecondLevelCaching.Test.DataLayer
{
    public class ProductContext : DbContext
    {
        public DbSet<Product> Products { get; set; }

        public override int SaveChanges()
        {
            this.InvalidateSecondLevelCache();
            return base.SaveChanges();
        }
    }
}
```

در اینجا با تحریف متد SaveChanges، می‌توان درست در زمان اعمال تغییرات به بانک اطلاعاتی، قسمتی از کش را غیرمعتبر کرد.

نحوه استفاده از سطح دوم کش توسعه داده شده

مثالی از کاربرد متدهای الحاقی توسعه داده شده را در ذیل مشاهده می‌کنید:

```
using System.Data.Entity;
using System.Linq;
using EfSecondLevelCaching.Core;
using EfSecondLevelCaching.Test.DataLayer;
using EfSecondLevelCaching.Test.Models;
using System;

namespace EfSecondLevelCaching
{
    public static class TestUsages
    {
        public static void RunQueries()
        {
            using (ProductContext context = new ProductContext())
            {
                var isActive = true;
                var name = "Product1";

                // reading from db
                var list1 = context.Products
                                   .OrderBy(one => one.ProductNumber)
                                   .Where(x => x.IsActive == isActive && x.ProductName == name)
                                   .ToCacheableList();

                // reading from cache
                var list2 = context.Products
                                   .OrderBy(one => one.ProductNumber)
                                   .Where(x => x.IsActive == isActive && x.ProductName == name)
                                   .ToCacheableList();

                // reading from cache
                var list3 = context.Products
                                   .OrderBy(one => one.ProductNumber)
                                   .Where(x => x.IsActive == isActive && x.ProductName == name)
                                   .ToCacheableList();

                // reading from db
                var list4 = context.Products
                                   .OrderBy(one => one.ProductNumber)
                                   .Where(x => x.IsActive == isActive && x.ProductName == "Product2")
                                   .ToCacheableList();
            }

            // removes products cache
            using (ProductContext context = new ProductContext())
            {
                var p = new Product()
                {
                    IsActive = false,
                    ProductName = "P4",
                    ProductNumber = "004"
                };
                context.Products.Add(p);
                context.SaveChanges();
            }

            using (ProductContext context = new ProductContext())
            {
                var data = context.Products.AsQueryable().Cacheable(x => x.FirstOrDefault());
                var data2 = context.Products.AsQueryable().Cacheable(x => x.FirstOrDefault());
                context.SaveChanges();
            }
        }
    }
}
```

در این حالت اگر برنامه را اجرا کنیم به یک چنین خروجی در پنجره Debug ویژوال استودیو خواهیم رسید:

```
Adding new data: queryKey=72AF5DA1BA9B91E24DCCF83E88AD1C5F,
dependencyKey=EfSecondLevelCaching.Test.Models.Product
```

```
Available cached keys list:
queryKey: EfSecondLevelCaching.Test.Models.Product
queryKey: 72AF5DA1BA9B91E24DCCF83E88AD1C5F
count: 2

Fetching object 'EfSecondLevelCaching.Test.Models.Product__72AF5DA1BA9B91E24DCCF83E88AD1C5F' from the
cache.

Available cached keys list:
queryKey: EfSecondLevelCaching.Test.Models.Product
queryKey: 72AF5DA1BA9B91E24DCCF83E88AD1C5F
count: 2

Fetching object 'EfSecondLevelCaching.Test.Models.Product__72AF5DA1BA9B91E24DCCF83E88AD1C5F' from the
cache.

Available cached keys list:
queryKey: EfSecondLevelCaching.Test.Models.Product
queryKey: 72AF5DA1BA9B91E24DCCF83E88AD1C5F
count: 2

Adding new data: queryKey=11A2C33F9AD7821A0A31003BFF1DF886,
dependencyKey=EfSecondLevelCaching.Test.Models.Product

Available cached keys list:
queryKey: 72AF5DA1BA9B91E24DCCF83E88AD1C5F
queryKey: 11A2C33F9AD7821A0A31003BFF1DF886
queryKey: EfSecondLevelCaching.Test.Models.Product
count: 3

Removing items with dependencyKey=EfSecondLevelCaching.Test.Models.Product
Available cached keys list:
count: 0
Available cached keys list:
count: 0

Adding new data: queryKey=02E6FE403B461E45C5508684156C1D10,
dependencyKey=EfSecondLevelCaching.Test.Models.Product

Available cached keys list:
queryKey: 02E6FE403B461E45C5508684156C1D10
queryKey: EfSecondLevelCaching.Test.Models.Product
count: 2


Fetching object 'EfSecondLevelCaching.Test.Models.Product__02E6FE403B461E45C5508684156C1D10' from the
cache.
```

توضیحات:

در زمان تولید `list1` چون اطلاعاتی در کش سطح دوم وجود ندارد، پیغام `Adding new data` قابل مشاهده است. اطلاعات از بانک اطلاعاتی دریافت شده و سپس در کش قرار داده می‌شود.

حین فراخوانی `list2` که دقیقا همان کوئری `list1` را یکبار دیگر فراخوانی می‌کند، به عبارت `Fetching object` خواهیم رسید که بر دریافت اطلاعات از کش سطح دوم بجای مراجعه به بانک اطلاعاتی دلالت دارد.

در `list4` چون پارامترهای کوئری تغییر کرده‌اند، بنابراین دیگر کلید منحصربفرد معادل آن با `list1` و `lis2` یکی نیست و اینبار پیغام `Adding new data` مشاهده می‌شود؛ چون برای دریافت اطلاعات آن نیاز است که به بانک اطلاعاتی مراجعه شود.

در ادامه یک `context` دیگر باز شده و در آن رکوردی به بانک اطلاعاتی اضافه می‌شود. به همین دلیل اینبار پیام `Removing items with dependencyKey` قابل مشاهده است. به عبارتی متد `InvalidateSecondLevelCache` وارد عمل شده است و بر اساس تغییری که صورت گرفته، کش را غیرمعتبر کرده است.

سپس در `context` بعدی تعریف شده، دوبار متد `FirstOrDefault` فراخوانی شده است. اولین مورد `Adding new data` است و دومین فراخوانی به `Fetching object` ختم شده است (دریافت اطلاعات از کش).

کدهای کامل این پروژه را از اینجا می‌توانید دریافت کنید:

EfSecondLevelCaching.zip

نظرات خوانندگان

نویسنده: مجتبی کاویانی
تاریخ: ۱۳۹۱/۰۴/۰۵ ۱۲:۴۴

ممنون، من با executesqlcommand چندین رکور را حذف می‌کنم اما هنور در dbset کش شده است چگونه بدون ایجاد نمونه جدید از context آن را رفرش کنم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۵ ۱۲:۴۹

InvalidateSecondLevelCache فقط بر اساس اطلاعات موجود در کش سطح اول یا همان Context جاری کار می‌کند. بنابراین اگر از عبارات sql مستقیما استفاده کنید، در Context جاری لحاظ نخواهد شد مگر اینکه از متد context.Entry(entity1).Reload استفاده کنید.
در قسمت 14 سری EF code first این سایت به این مطلب پرداخته شده.

نویسنده: مجتبی کاویانی
تاریخ: ۱۳۹۱/۰۴/۰۵ ۱۳:۰

اگر برای اینکه تنها از یک context در برنامه استفاده کنیم در Global.cs یک static DbContext بسازم و در application_start , application_end آن را ایجاد و حذف کنیم روش خوبی است؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۵ ۱۳:۱۲

خیر. context باید به ازای هر request ایجاد و تخریب شود. در این مورد در قسمت 12 سری Ef Code first سایت جاری توضیح دادم (پیاده سازی الگوی Context Per Request در برنامه‌های مبتنی بر EF Code first).

نویسنده: میثم
تاریخ: ۱۳۹۱/۰۴/۲۹ ۹:۵۰

سلام و خسته نباشید
به این نتیجه رسیدم که اگر متد ToCacheableList() را در انتها اضافه نکنیم چیزی شبیه به این

```
var list2 = context.Products
        .OrderBy(one => one.ProductNumber)
        .Where(x => x.IsActive == isActive && x.ProductName == name);
```

روی کش هیچ تاثیری نداره یعنی نه چیزی رو کش می‌کنه و نه چیزی رو از کش می‌خونه و نه کش رو پاک می‌کنه ولی list3 دوباره اطلاعات رو از کش می‌خونه آیا این موضوع صحیح است ؟
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۲۹ ۹:۵۴

بهترین راه جهت تصدیق یا رد کل مطالب عنوان شده استفاده از SQL Server Profiler و مشاهده SQL خروجی است و همچنین شمارش تعداد بار رفت و برگشت به بانک اطلاعاتی (بر اساس حداقل موارد لاگ شده در پروفایلر).
+
کوئری شما فقط یک expression است. هنوز اجرا نشده. اجرای یک عبارت با فراخوانی متدهایی مانند ToList، FirstOrDefault و امثال آن رخ می‌دهد. به این مورد deferred execution گفته می‌شود (قسمت دهم سری ef code first

سایت جاری).

نویسنده: جلال
تاریخ: ۱۳۹۱/۰۶/۱۹ ۱۰:۴۹

با تشکر،
فقط مسئله ای که هست، اینه که از Cache مربوط به ASP.NET استفاده می‌کنه و برای یه نرم افزار Desktop مناسب نیست. آیا پیاده سازی ای با MemoryCache نداره؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۹ ۱۰:۵۷

- همین پیاده سازی فوق رو با یک برنامه کنسول ویندوزی هم تست کردم، کار می‌کنه. می‌خواهید یک امتحانی بکنید. به نظر در پشت صحنه به صورت خودکار به memory cache سوئیچ میشه. فقط باید ارجاعی را به اسمبلی System.Web اضافه کنید.
- ضمن اینکه در برنامه‌های دسکتاپ این مساله اهمیت آنچنانی نداره؛ چون سطح دوم کش بیشتر جهت ارائه محتوایی یکسان و با دسترسی عمومی، به کاربران همزمان سایت کاربرد داره. عمده اطلاعات برنامه‌های دسکتاپ با سطح دسترسی خصوصی و مخصوص به یک کاربر است؛ در یک چنین مواردی نباید از سطح دوم کش استفاده کرد وگرنه به مشکلات امنیتی و فاش سازی اطلاعاتی که نباید عمومی شوند، منتهی خواهد شد (البته اگر مثلا از یک وب سرویس استفاده شده باشه؛ اگر همه چیز لوکال است، این مساله صادق نخواهد بود؛ اما باز هم نیازی به سطح دوم کش نیست. چون مهم‌ترین هدف آن کاهش بار بانک اطلاعاتی، در مراجعات مکرر کاربران است؛ که در حالت لوکال آنچنان معنی ندارد).

نویسنده: جلال
تاریخ: ۱۳۹۱/۰۶/۱۹ ۱۱:۱۵

ممنون بابت پاسخ سریع،
ولی برنامه من، حتی در Paging هم سرعت مورد انتظار من رو نداره. توی برنامه WPF من، هر بار ورق زدن، 15 رکورد ناقابل بارگذاری میشه و طی بررسی که انجام دادم بیشتر این مدت (از نیم ثانیه، 350 میلی ثانیه به کوئری اختصاص داره و بقیش شامل کارهایی مثل اعمال DataTemplate و Render و ...) و می‌خوام این زمان رو تا حد ممکن کمتر کنم. با خودم گفتم این لیست به ندرت ویرایش میشه. فقط Insert به طور روزانه انجام میشه و عمل حذف بسیار نادر رخ میده. اطلاعات صفحه اونقدر از نظر امنیتی اهمیت ندارند.
بانک اطلاعاتی مورد استفاده من، SQL Compact 4.0 است و از Entity Framework 4.3.1 و روش Code First استفاده می‌کنم.

نویسنده: جلال
تاریخ: ۱۳۹۱/۰۶/۱۹ ۱۲:۵

سلام مجدد.
من توی زمان Stop کردن Stopwatch اشتباه داشتم، زمان query گرفتن زیاد نیست و کاملا قابل صرف نظره و بنابراین نیازی به caching ندارم، بیشترین زمان رو render به خودش اختصاص داده متاسفانه و کار زیادی نمیشه کرد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۹ ۱۲:۲۸

احتمال داره در حین نمایش گرید، lazy loading فعال است و به این ترتیب بدون اینکه متوجه باشید چند صد کوئری مجدد به بانک اطلاعاتی ارسال می‌شود. در این حالت کار نمایش بسیار کند خواهد بود. این مساله رو فقط با یک پروفایلر می‌شود تشخیص داد؛ که روش آن در مقاله ذکر شده قسمت 10 بررسی شده. همچنین مطلب کاهش مصرف حافظه را هم مدنظر داشته باشید.

نویسنده: hosseinzadeh
تاریخ: ۱۳۹۱/۰۶/۱۹ ۱۳:۵۳

ظاهرا متد GetCacheKey به ازای کوئری‌های مختلف نتیجه یکسانی رو بر میگردونه و نهایتا همیشه دیتای کش شده نمایش داده

میشه. مثلا دو کوئری زیر :

```
ctx.Entity.SingleOrDefault(a=>a.ID==1);
ctx.Entity.SingleOrDefault(a=>a.ID==2);
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۹ ۱۴:۰

خیر. حداقل این مورد (بررسی ProductName با دو مقدار مختلف) در مثال‌های list1 تا list4 مطلب فوق بررسی شده (در متد RunQueries). لینک پروژه کامل هم در آخر مطلب قابل دریافت است.
+ مثال شما قابل بررسی و دیباگ نیست. لازم هست پروژه کامل باشد به همراه تعاریف تا بشود دید مشکل کار شما کجا است.

نویسنده: محسن
تاریخ: ۱۳۹۱/۰۷/۱۳ ۹:۲۶

با سلام
اگر تعداد تراکنش‌های زیادی را مدیریت کند بعد از مدتی خطای Out of Memory را می‌دهد. راه حلی برای اون موقع در نظر گرفته شده است؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۳ ۹:۳۲

خیر. از این جهت که کتابخانه فوق در اصل برای کار با کش IIS طراحی شده و زمانیکه absoluteExpiration آن‌را تنظیم می‌کنید، خود IIS به صورت خودکار موارد قدیمی را حذف می‌کند (آیتم‌های موجود در کش مدت دار خواهند شد). به علاوه IIS هر زمان که احساس کند از لحاظ مصرف حافظه زیر فشار است راسا شروع به حذف کردن آیتم‌های موجود در کش می‌کند.
جهت اطلاع اکثر قسمت‌های سایت جاری از کتابخانه فوق استفاده می‌کنند و تابحال مشکلی با مصرف حافظه مشاهده نشده.

نویسنده: محسن
تاریخ: ۱۳۹۱/۰۷/۱۵ ۹:۵۹

به شخصه با این مشکل روبرو شدم.حذف توسط iis راه حل مناسبی نیست.در یک سیستمی که تراکنش‌های زیادی در زمان کمی دریافت میکنه راه حل مناسب مدیریت این منبع توسط خود برنامه نویسه.ممکنه کاربری همزمان در حال کار بر روی این cash باشه و به علت مصرف زیاد حافظه iis اون رو حذف کنه.راه حلی که ایجاد شد :
-1 مدیریت تعداد رکوردهای مورد استفاده(مثلا برای کار ما بر روی 5000 رکورد بود)
-2 حذف رکوردهای قدیمی بر اساس زمان استفاده.
این 2 مورد در زمان ذخیره تغییرات اعمال می‌شدند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۵ ۱۰:۲۵

حق با شما است. من ندیدم کسی رو به ازای هر کاربر یا هر عملیات ریزی در سایت، 5000 رکورد را در کش ذخیره کند.

نویسنده: کیارش سلیمان زاده
تاریخ: ۱۳۹۱/۱۱/۲۳ ۱۲:۳۲

سلام آقای نصیری،

از سطح دوم کش باید تو لایه سرویس استفاده بشه؟
اگه تو لایه سرویس باید استفاده کرد،لایه سرویس وابسته به HttpRuntime که برای درج تو کش استفاده شده(coupling)،نمیشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۱/۲۳ ۱۲:۵۰

- «مثال» این قسمت یک برنامه ویندوزی کنسول است. در جائیکه وب سرور در دسترس نباشه به صورت خودکار به Memory Cache سوئیچ می‌کنه. (البته فرض بر این است که یکبار اجراش کردید یا حداقل خروجی درج شده رو بررسی کردید)
- زمانیکه از لایه سرویس استفاده می‌کنید، استفاده کننده نهایی فقط با یک سری اینترفیس کار می‌کنه نه الزاما پیاده سازی خاص شما. به عبارتی می‌شود mocking رو به سادگی اعمال کرد روی این لایه.
- هدف از این سایت ارائه ایده هست، نه راه حل‌های جهان شمول بی عیب و نقص قابل استفاده در تمام مسایل و مشکلات بشری. همینقدر که ایده‌ای مطرح شده، نکته‌ی جدیدی عنوان شده و کمی تونسته ذهن شما رو درگیر کنه، رسالت خودش رو انجام داده.
- اکثر کارهای این سایت سورس باز هستند. یعنی اگر به این نتیجه رسیدید که می‌تونید کیفیتش رو بهبود ببخشید، لطفا حتما اینکار رو انجام بدید و یک وصله ارائه کنید. البته بعد از اینکار هم حتما ذکر کنید که از چه cache provider جدیدی قرار هست خصوصا در برنامه‌های وب قابل اجرا در IIS که کاربرد اصلی این بحث است، استفاده بشه.

نویسنده: کیارش سلیمان زاده
تاریخ: ۱۳۹۱/۱۱/۲۳ ۱۳:۱۱

ممنون،نظر شما اینه که دو متد ToCacheableList و Cacheable رو تو یه اینترفیس تعریف کنیم و در لایه سرویس با این اینترفیس‌ها کار کنیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۱/۲۳ ۱۳:۲۳

تکرار مجدد:
- هر کلاس لایه سرویس با پیاده سازی یک اینترفیس باید تهیه شود.این مورد به نظر در قسمت 12 سری EF بحث شده با مثال و فایل و همه چیز در برنامه‌های کنسول و MVC و وب فرم‌ها.
- کلاس کمکی فوق نیازی به وب سرور برای اجرا ندارد و باعث fail آزمون‌های واحد شما نمی‌شود چون در صورت نبودن وب سرور از حافظه سیستم استفاده می‌کند نه کش IIS.
- اگر به این نتیجه رسیدید که کش پروایدر بهتری وجود دارد و نیاز به تعویض نمونه مطرح شده در اینجا هست (که من در «مثال» ارائه شده نیازی به آن نداشتم)، لطفا آن‌را معرفی کنید و همچنین پیاده سازی اصلاح شده را به صورت یک وصله ارائه کنید جهت تکمیل بحث.

نویسنده: مهتدی حسن‌پور
تاریخ: ۱۳۹۲/۰۴/۱۳ ۱۰:۲۱

من هنگام cache کردن برخی از queryها با این خطا روبرو شدم:

System.InvalidOperationException When called from 'VisitMemberInit', rewriting a node of type 'System.Linq.Expressions.NewExpression' must return a non-null value of the same type. Alternatively, override 'VisitMemberInit' and change it to not visit children of this type.

برای حل اون این کد رو به کلاس داخلی SubtreeEvaluator در کلاس Evaluator در فایل QueryResultCache.cs اضافه کردم:

```
protected override Expression VisitMemberInit(MemberInitExpression node)
 {
       if (node.NewExpression.NodeType == ExpressionType.New)
           return node;
        return base.VisitMemberInit(node);
 }
```

نویسنده: محمد شهریاری
تاریخ: ۱۳۹۲/۰۹/۰۱ ۱۸:۹

سلام
-1 در متد RunQueries از سه Context جدا استفاده کردید من همین مثال رو در یک context استفاده کردم خروجی نهایی یکی بود دلیل خاصی داشت که شما هر بخش را در یک context بلاک جداگانه فراخوانی کردید
-2 در سومین context با اینکه عملیات خواندن صورت میگیره متد savechaneges رو فراخونی کردید اگه امکان داره بشتر توضیح بدید ممنون میشم .

ضمنا قسمت حذف یک key از cache خیلی جالب بود .

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۰۱ ۱۸:۲۳

بیشتر هدف تست کردن با چند تراکنش مختلف بوده. هر Context جدید یا هر SaveChanges یعنی خاتمه تراکنش قبلی و شروع تراکنش بعدی.

نویسنده: رضا گرمارودی
تاریخ: ۱۳۹۲/۰۹/۰۲ ۱۶:۲۰

ببخشید؛ من بخش آخر را متوجه نشدم. هنگامی که تغییری در یک جدول ایجاد می‌کنیم با دستور this.InvalidateSecondLevelCache(); کل کش را غیره معتبر می‌کند یا فقط جدولی که تغییرات داشته است ؟

نویسنده: رضا گرمارودی
تاریخ: ۱۳۹۲/۰۹/۰۲ ۱۷:۵۷

سعی کردم کدهام و با SecondLevelCash به صورت Reactor اصلاح کنم اما یکی از موارد پر کاربرد گرفتند Count از Iqueryble است .

موقع Count گرفتن Linq به دستورات Sql مواردی اضافه می‌کند و نمی‌توان Count را کش کرد.

برای این دست موارد باید دستی Query کانت جنریت بشه و یا راه حل دیگه ای دارد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۰۲ ۱۸:۲۵

```
context.Products.Cacheable(x => x.First())
context.Products.Cacheable(x => x.Count())
...
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۰۲ ۱۸:۲۹

سورس کامل آن در مطلب فوق در دسترس است . فقط جداولی را که (نه یک جدول؛ چون در یک Context می‌شود با چند جدول کار کرد) تغییرات داشتند بررسی می‌کند.

نویسنده: رضا گرمارودی
تاریخ: ۱۳۹۲/۰۹/۱۸ ۱۴:۵۶

سلام؛ نظرتون در رابطه با ترکیب سطح دوم کش و از کارانداختن سطح اول کش در زمان گزارش گیری چیه؟

به نظرتون کد زیر مناسب‌تر هست و یا چون دستور اسکیوالی اجرا نمیشه لزومی به اجرای آن ندارد؟

```
context.Products.AsNoTracking().ToCacheableList()
```

نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۰۹/۲۴ ۱۵:۴۹

سلام.
من به ازای هر viewmodel یک context ایجاد میکنم.خصوصیات یک entity را در یک context تغییر می‌دهم.اما تغییرات را در context بعدی ندارم. -آیا همچین چیزی طبیعیه؟
-بنابراین مجبورم از context.Entry(entity1).Reload استفاده کنم. اما مشکل اینجاست که این دستور relationها را فراخوانی مجدد نمی‌کند.مشکل من اینه که اگر نتونیم ازاین navigation propertyها استفاده کنیم و بخواهیم از dbsetهای خودشون مستقیم دیتا رو لود کنیم که استفاده این خصوصیات چیه؟! -و البته سوال دیگر من اینه که این اطلاعات چه موقع cash می‌شوند؟چون اینطور به نظر میرسه که در اولین فراخوانی از دیتابیس کش شده اند (بدون توجه به شی context) و حالا در همه contextهای بعدی از همانجا لود میشوند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۴ ۱۷:۴۴

خیر. طبیعی نیست. اگر هم کش می‌شود یا این احساس را دارید، یعنی Context هنوز Dispose نشده. یک نمونه توضیحات بیشتر در اینجا:
« نکته‌ای در مورد مدیریت طول عمر اشیاء در حالت HybridHttpOrThreadLocalScoped در برنامه‌های دسکتاپ »

+ بحث سطح دوم کش (بحث جاری) کاری به Context ندارد. مستقل عمل می‌کند. در اینجا فقط از Context سؤال می‌پرسد چه کوئری قرار هست صادر شود. بعد نتیجه‌اش را از کش سیستم (و نه Context جاری) دریافت می‌کند.

نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۰۹/۲۶ ۱۳:۵۶

در مورد پاسختون من Context رو Dispose می کردم و مشکلم ربطی به اون نداشت.بعد از گشتن‌های زیاد متوجه شدم که مشکل اینجاست که Context دوم که داده‌ی به روز شده رو لود نمیکنه قبل از context ی که داده‌ها رو تغییر داده ایجاد شده(مثلا دو فرم رو تصور کنید که همزمان بازند و دومین فرمی که باز شده تغییرات رو انجام داده،وقتی دوباره به فرم اول بر میگردیم تغییرات وجود ندارند).من مشکل رو متوجه شدم اما دلیلشو نفهمیدم .اینطور به نظر میرسه که دفعه اولی که یک کوئری اجرا می‌شه اون رو cash میکنه و دفعه‌های بعدی دیگه سمت دیتابیس نمیره.

نویسنده: حسین
تاریخ: ۱۳۹۲/۱۱/۲۱ ۱۸:۷

با سلام وتشکر
شما با روابط many to many مشکلی ندارید با این روش کش کردن؟

نویسنده: علی
تاریخ: ۱۳۹۳/۰۸/۰۵ ۱۲:۱۶

سلام.
آیا از این روش می‌توان در Database first استفاده کرد؟

نویسنده: وحید نصیری

تاریخ: ۱۳۹۳/۰۸/۰۵ ۱۲:۲۱

- نیازی به اندکی تغییر دارد. DbContext آن باید بشود ObjectContext و امثال آن.
- ضمنا پروژه‌ی « Second Level Cache for Entity Framework 6.1 » را هم مدنظر داشته باشید.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۳/۱۰/۰۷ ۹:۳۶

من با استفاده از دستور زیر یک مقاله توی لایه سرویس بازیابی میکنم
اما هر بار بعد از اینکه یک مقاله در کش ذخیره شد به ازای مقالات دیگر همان مقاله ابتدایی که در کش ذخیره شده بازیابی میشود. این نمونه دستور من برای بازیابی هست.

```
public Article GetById(int id)
        {
            return articles.Where(e => e.Id == id).Include(x => x.Category).Include(x =>
x.Tags).Cacheable(x => x.FirstOrDefault());
        }
```

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۳/۱۰/۰۸ ۱۳:۲۷

اشکال کار را پیدا کردم
چون سیستم کش لایه دوم بر اساس پارامتر آخر عمل میکنه باعث میشه که دیتا به ازای پارامترهای مختلف یک داده خاص فقط از کش بازیابی کنه چون باعث میشه کلید یکسان برای همه ثبت بشه مثلا در دستور زیر

```
public Article GetById(int id)
{
   return articles
          .Where(e => e.Id == id)
          .Include(x => x.Category)
          .Include(x => x.Tags) //پارامتر در نظر گرفته شده برای تولید کلید کش
          .Cacheable(x => x.FirstOrDefault());
}
```

قسمت Tags به عنوان پارامتر در نظر گرفته میشه که برای همه مقالات یکسان از آب در میاد
در صورتی که دستور به شکل زیر اصلاح بشه این مشکل رفع میشه چون در اون صورت Id مقاله به عنوان پارامتر در نظر گرفته میشه

```
public Article GetById(int id)
{
   return articles
          .Include(x => x.Category)
          .Include(x => x.Tags)
          .Where(e => e.Id == id) //پارامتر در نظر گرفته شده برای تولید کلید کش
          .Cacheable(x => x.FirstOrDefault());
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۱۰ ۱۵:۳۱

پروژه‌های تکمیلی

- https://github.com/loresoft/EntityFramework.Extended دقیقا شبیه به پیاده سازی مثال جاری هست. فقط cache provider اختصاصی ایجاد کرده (بجای استفاده از HttpRuntime.Cache).
- https://efcache.codeplex.com داخل سیستم Data Reader می‌شود برای کش کردن. (جهت ایده دادن به تیم EF خوب است)
- https://github.com/osjoberg/LinqCache نمونه‌ی توسعه یافته مثال جاری است.