عنوان: **چند نکته کاربردی درباره Entity Framework**
نویسنده: ایمان محمدی
تاریخ: ۱۳۹۲/۰۳/۰۹ ۱۴:۵۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, EF Code First

1) رفتار متصل و غیر متصل در EF چیست؟

اولین نکته ای که به ذهنم می‌رسه اینه که برای استفاده از EF حتما باید درک صحیحی از رفتارها و قابلیت‌های اون داشته باشیم. نحوه استفاده ازEF رو به دو رفتار متصل و غیر متصل تقسیم می‌کنیم.

حالت پیش فرضEF بر مبنای رفتار متصل می‌باشد. در این حالت شما یک موجودیت رو از دیتابیس فرا می‌خونید EF این موجودیت رو ردگیری می‌کنه اگه تغییری در اون مشاهده کنه بر روی اون برچسب "تغییر داده شد" می‌زنه و حتی اونقدر هوشمند هست که وقتی تغییرات رو ذخیره می‌کنید کوئری آپدیت رو فقط براساس فیلدهای تغییر یافته اجرا کنه. یا مثلا در صورتی که شما بخواهید به یک خاصیت رابطه ای دسترسی پیدا کنید اگر قبلا لود نشده باشه در همون لحظه از دیتابیس فراخوانی میشه، البته این رفتارها هزینه بر خواهد بود و در تعداد زیاد موجودیت‌ها میتونه کارایی رو به شدت پایین بیاره.

رفتار متصل شاید در ویندوز اپلیکیشن کاربرد داشته باشه ولی در حالت وب اپلیکیشن کاربردی نداره چون با هر در خواستی به سرور همه چیز از نو ساخته میشه و پس از پاسخ به درخواست همه چی از بین میره. پس DbContext همیشه از بین می‌ره و ما برحسب نیاز، در درخواست‌های جدید به سرور ، دوباره DbContext رو می‌سازیم. پس از ساخته شدن DbContext باید موجودیت مورد استفاده رو به اون معرفی کنیم و وضعیت اون موجودیت رو هم مشخص کنیم.(جدید ، تغییر یافته، حذف ، بدون تغییر) در این حالت سیستم ردگیری تغییرات بی استفاده است و ما فقط در حال هدر دادن منابع سیستم هستیم.

در حالت متصل ما باید همیشه از یک DbContext استفاده کنیم و همه موجودیت‌ها در آخر باید تحت نظر این DbContext باشند در یک برنامه واقعی کار خیلی سخت و پیچیده ای است. مثلا بعضی وقت‌ها مجبور هستیم از موجودیت هایی که قبلا در حافظه برنامه بوده اند استفاده کنیم اگر این موجودیت در حافظه DbContext جاری وجود نداشته باشه با معرفی کردن اون از طریق متد attach کار ادامه پیدا می‌کنه ولی اگر قبلا موجودیتی در سیستم ردگیری DbContext با همین شناسه وجود داشته باشد با خطای زیر مواجه می‌شویم.

```
An object with the same key already exists in the ObjectStateManager. The ObjectStateManager cannot
track multiple objects with the same key
```

این خطا مفهوم ساده و مشخصی داره ، دو شی با یک شناسه نمی‌توانند در یک DbContext وجود داشته باشند. معمولا در این حالت ما بااین اشیا تکراری کاری نداریم و فقط به شناسه اون شی برای نشان دادن روابط نیاز داریم و از دیگر خاصیت‌های اون جهت نمایش به کاربر استفاده می‌کنیم ولی متاسفانه DbContext نمی‌دونه چی تو سر ما می‌گذره و فقط حرف خودشو می‌زنه! البته اگه خواستید با DbContext بر سر این موضوع گفتگو کنید از کدهای زیر استفاده کنید:

```
T attachedEntity = set.Find(entity.Id);
var attachedEntry = dbContext.Entry(attachedEntity);
attachedEntry.CurrentValues.SetValues(entity);
```

خوب با توجه به صحبت‌های بالا اگر بخواهیم از رفتار غیر متصل استفاده کنیم باید تنظیمات زیر رو به متد سازنده DbContext اضافه کنیم. از اینجا به بعد همه چیز رو خودمون در اختیار می‌گیریم و ما مشخص می‌کنیم که کدوم موجودیت باید چه وضعیتی داشته باشه (افزودن ، بروز رسانی، حذف)و اینکه چه موقع روابط خودش را با دیگر موجودیتها فراخوانی کنه.

```
public DbContext()
        {
            this.Configuration.ProxyCreationEnabled = false;
            this.Configuration.LazyLoadingEnabled = false;
            this.Configuration.AutoDetectChangesEnabled = false;
         }
```

2) تعیین وضعیت یک موجودیت و راوبط آن در EF چگونه است؟

با کد زیر می‌تونیم وضعیت یک موجدیت رو مشخص کنیم ، با اجرای هر یک از دستورات زیر موجودیت تحت نظر DbContext

قرار می‌گیره یعنی عمل attach نیز صورت گرفته است :

```
dbContext.Entry(entity).State = EntityState.Unchanged ;
dbContext.Entry(entity).State = EntityState.Added ; //or  Dbset.Add(entity)
dbContext.Entry(entity).State = EntityState.Modified ;
dbContext.Entry(entity).State = EntityState.Deleted ; // or  Dbset.Remove(entity)
```

با اجرای این کد موجودیت از سیستم ردگیری DbContext خارج می‌شه.

```
dbContext.Entry(entity).State = EntityState.Detached;
```

در موجودیت‌های ساده با دستورات بالا نحوه ذخیره سازی را مشخص می‌کنیم در وضعیتی که با موجودیت‌های رابطه ای سروکار داریم باید به نکات زیر توجه کنیم.

در نظر بگیرید یک گروه از قبل وجود دارد و ما مشتری جدیدی می‌سازیم در این حالت انتظار داریم که فقط یک مشتری جدید ذخیره شده باشد:

```
// group id=19  Name="General"
var customer = new Customer();
customer.Group = group;
customer.Name = "mohammadi";
dbContext.Entry(customer).State = EntityState.Added;
var customerstate = dbContext.Entry(customer).State;// customerstate=EntityState.Added
var groupstate = dbContext.Entry(group);// groupstate=EntityState.Added
```

اگه از روش بالا استفاده کنید می‌بینید گروه General جدیدی به همراه مشتری در دیتابیس ساخته می‌شود.نکته مهمی که اینجا وجود داره اینه که DbContext به id موجودیت گروه توجهی نداره ، برای جلو گیری از این مشکل باید قبل از معرفی موجودیت‌های جدید رابطه هایی که از قبل وجود دارند را به صورت بدون تغییر attach کنیم و بعد وضعیت جدید موجودیت رو اعمال کنیم.

```
// group id=19  Name="General"
var customer = new Customer();
customer.Group = group;
 customer.Name = "mohammadi";
dbContext.Entry(group).State = EntityState.Unchanged;
dbContext.Entry(customer).State = EntityState.Added;
var customerstate = dbContext.Entry(customer).State;// customerstate=EntityState.Added
 var groupstate = dbContext.Entry(group);// groupstate=EntityState.Unchanged
```

در مجموع بهتره که موجودیت ریشه رو attach کنیم و بعد با توجه به نیاز تغییرات رو اعمال کنیم.

```
// group id=19  Name="General"
var customer = new Customer();
 customer.Group = group;
customer.Name = "mohammadi";
dbContext.Entry(customer).State = EntityState.Unchanged;
dbContext.Entry(customer).State = EntityState.Added;
var customerstate = dbContext.Entry(customer).State;// customerstate=EntityState.Added
var groupstate = dbContext.Entry(group);//// groupstate=EntityState.Unchanged
```

3) AsNoTracking و Include دو ابزار مهم در رفتار غیر متصل:

درصورتیکه ما تغییراتی روی داده‌ها نداشته باشیم و یا از روش‌های غیر متصل از موجودیت‌ها استفاده کنیم با استفاده از متد AsNoTracking() در زمان و حافظه سیستم صرف جویی می‌کنیم در این حالت موجودیت‌های فراخوانی شده از دیتابیس در سیستم

ردگیری DbContext قرار نمی‌گیرند و اگر وضعیت آنها را بررسی کنیم در وضعیت Detached قرار دارند.

```
var customer  = dbContext.Customers.FirstOrDefault();
 var customerAsNoTracking  = dbContext.Customers.AsNoTracking().FirstOrDefault();
var customerstate = dbContext.Entry(customer).State;// customerstate=EntityState.Unchanged
var customerstateAsNoTracking  = dbContext.Entry(customerAsNoTracking).State;//
customerstate=EntityState.Detached
```

نحوه بررسی کردن موجودیت‌های موجود در سیستم ردگیری DbContext :

```
var Entries = dbContext.ChangeTracker.Entries();
var AddedEntries = dbContext.ChangeTracker.Entries().Where(entityEntry =>
entityEntry.State==EntityState.Added);
var ModifiedEntries = dbContext.ChangeTracker.Entries().Where(entityEntry =>
entityEntry.State==EntityState.Modified);
var UnchangedEntries = dbContext.ChangeTracker.Entries().Where(entityEntry =>
entityEntry.State==EntityState.Unchanged);
var DeletedEntries = dbContext.ChangeTracker.Entries().Where(entityEntry =>
entityEntry.State==EntityState.Deleted);
var DetachedEntries = dbContext.ChangeTracker.Entries().Where(entityEntry =>
entityEntry.State==EntityState.Detached);//* not working !
```

* در نظر داشته باشید وضعیت Detached وجود خارجی ندارد و به حالتی گفته می‌شود که DbContext در سیستم رد گیری خود اطلاعی از موجودیت مورد نظر نداشته باشد.

وقتی که سیستم فراخوانی خودکار رابطه‌ها خاموش باشد باید موقع فراخوانی موجودیت‌ها روابط مورد نیاز را هم با دستور Include در سیستم فراخوانی کنیم.

```
var CustomersWithGroup = dbContext.Customers.AsNoTracking().Include("Group").ToList();
 var CustomerFull =
dbContext.Customers.AsNoTracking().Include("Group").Include("Bills").Include("Bills.BillDetails").ToLis
t();
```

4) از متد AddOrUpdate در در فضای نام System.Data.Entity.Migrations استفاده نکنیم، چرا؟

در صورتی که از فیلد RowVersion و کنترل مسایل همزمانی استفاده کرده باشیم هر وقتی متد AddOrUpdate رو فراخوانی کنیم، تغییر اطلاعات توسط دیگر کاربران نادیده گرفته می‌شود. با توجه به این که متد AddOrUpdate برای عملیات Migrations در نظر گرفته شده است، این رفتار کاملا طبیعی است. برای حل این مشکل می‌تونیم این متد رو با بررسی شناسه به سادگی پیاده سازی کنیم:

```
public virtual void AddOrUpdate(T entity)
        {
            if (entity.Id == 0)
                Add(entity);
            else
                Update(entity);

        }
```

5) اگر بخواهیم موجودیت‌های رابطه ای در دیتا گرید ویو (ویندوز فرم) نشون بدیم باید چه کار کنیم؟

گرید ویو در ویندوز فرم قادر به نشون دادن فیلدهای رابطه ای نیست برای حل این مشکل می‌تونیم یک نوع ستون جدید برای گرید ویو تعریف کنیم و برای نشون دادن فیلدهای رابطه ای از این نوع ستون استفاده کنیم:

```
public class DataGridViewChildRelationTextBoxCell : DataGridViewTextBoxCell
    {
        protected override object GetValue(int rowIndex)
        {
            try
            {
```

```
                var bs = (BindingSource)DataGridView.DataSource;
                var cl = (DataGridViewChildRelationTextBoxColumn)DataGridView.Columns[ColumnIndex];
                return getChildValue(bs.List[rowIndex], cl.DataPropertyName).ToString();
            }
            catch (Exception)
            {
                return "";
            }
        }

        private object getChildValue(object dataSource, string childMember)
        {
            int nextPoint = childMember.IndexOf('.');
            if (nextPoint == -1) return
dataSource.GetType().GetProperty(childMember).GetValue(dataSource, null);
            string proName = childMember.Substring(0, nextPoint);
            object newDs = dataSource.GetType().GetProperty(proName).GetValue(dataSource, null);
            return getChildValue(newDs, childMember.Substring(nextPoint + 1));
        }
    }

    public class DataGridViewChildRelationTextBoxColumn : DataGridViewTextBoxColumn
    {
        public string DataMember { get; set; }

        public DataGridViewChildRelationTextBoxColumn()
        {
            CellTemplate = new DataGridViewChildRelationTextBoxCell();
        }
    }
```

نحوه استفاده را در ادامه می‌بینید. این روش توسط ویزارد گریدویو هم قابل استفاده است. موقع Add کردن Column نوع اون رو روی DataGridViewChildRelationTextBoxColumn تنظیم کنید.

```
GroupNameColumn= new DataGridViewChildRelationTextBoxColumn(); //from your class
GroupNameColumn.HeaderText = "گروه مشتری";
GroupNameColumn.DataPropertyName = "Group.Name"; //EF  Property: Customer.Group.Name
GroupNameColumn.Visible = true;
GroupNameColumn.Width = 300;
DataGridView.Columns.Add(GroupNameColumn);
```

نظرات خوانندگان

نویسنده: امیرحسین جلوداری
تاریخ: ۱۳۹۲/۰۳/۰۹ ۱۶:۹

سلام ... مطلب بسیار کاربردی بود ...
در برنامه‌های وب با استفاده از ابزارهایی مثه Stucturemap میتوان DbContext رو به httpContext محدود کرد که فک میکنم باعث رفتار متصل میشه !

با توجه به این مقاله (قسمت **تعیین طول عمر اشیاء در StructureMap**)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۹ ۱۷:۳

در حالت Detached (مثل ایجاد یک شیء CLR ساده)
در متد Updateایی که نوشتید، قسمت Find حتما اتفاق می‌افته. چون Tracking خاموش هست (مطابق تنظیماتی که عنوان کردید)، بنابراین Find چیزی رو از کشی که وجود نداره نمی‌تونه دریافت کنه و میره سراغ دیتابیس. ماخذ :

```
The Find method on DbSet uses the primary key value to attempt to find an entity tracked by the
context.
If the entity is not found in the context then a query will be sent to the database to find the entity
there.
Null is returned if the entity is not found in the context or in the database.
```

حالا تصور کنید که در یک حلقه می‌خواهید 100 آیتم رو ویرایش کنید. یعنی 100 بار رفت و برگشت خواهید داشت با این متد Update سفارشی که ارائه دادید. البته منهای کوئری‌های آپدیت متناظر. این 100 تا کوئری فقط Find است.
قسمت Find متد Update شما در حالت detached اضافی است. یعنی اگر می‌دونید که این Id در دیتابیس وجود داره نیازی به Findاش نیست. فقط State اون رو تغییر بدید کار می‌کنه .

در حالت نه آنچنان Detached ! (دریافت یک لیست از Context ایی که ردیابی نداره)
با خاموش کردن Tracking حتما نیاز خواهید داشت تا متد context.ChangeTracker.DetectChanges رو هم پیش از ذخیره سازی یک لیست دریافت شده از بانک اطلاعاتی فراخوانی کنید. وگرنه چون این اطلاعات ردیابی نمی‌شوند، هر تغییری در آن‌ها، وضعیت Unchanged رو خواهد داشت و نه Detached. بنابراین SaveChanges عمل نمی‌کنه؛ مگر اینکه DetectChanges فراخوانی بشه.

سؤال: این سربار که می‌گن چقدر هست؟ ارزشش رو داره که راسا خاموشش کنیم؟ یا بهتره فقط برای گزارشگیری این کار رو انجام بدیم؟
یک آزمایش:

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

namespace EF_General.Models.Ex21
{
    public abstract class BaseEntity
    {
        public int Id { set; get; }
    }

    public class Factor : BaseEntity
    {
        public int TotalPrice { set; get; }
    }
```

```
    public class MyContext : DbContext
    {
        public DbSet<Factor> Factors { get; set; }

        public MyContext() { }
        public MyContext(bool withTracking)
        {
            if (withTracking)
                return;

            this.Configuration.ProxyCreationEnabled = false;
            this.Configuration.LazyLoadingEnabled = false;
            this.Configuration.AutoDetectChangesEnabled = false;
        }

        public void CustomUpdate<T>(T entity) where T : BaseEntity
        {
            if (entity == null)
                throw new ArgumentException("Cannot add a null entity.");


            var entry = this.Entry<T>(entity);
            if (entry.State != EntityState.Detached)
                return;

            /*var set = this.Set<T>(); // این‌ها اضافی است
            //متد فایند اگر اینجا باشه حتما به بانک اطلاعاتی رجوع می‌کنه در حالت منقطع از زمینه و در یک حلقه
به روز رسانی کارآیی مطلوبی نخواهد داشت
            T attachedEntity = set.Find(entity.Id);
            if (attachedEntity != null)
            {
                var attachedEntry = this.Entry(attachedEntity);
                attachedEntry.CurrentValues.SetValues(entity);
            }
            else
            {*/
            entry.State = EntityState.Modified;
            //}
        }
    }

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

        protected override void Seed(MyContext context)
        {
            if (!context.Factors.Any())
            {
                for (int i = 0; i < 20; i++)
                {
                    context.Factors.Add(new Factor { TotalPrice = i });
                }
            }
            base.Seed(context);
        }
    }

    public class Performance
    {
        public TimeSpan ListDisabledTracking { set; get; }
        public TimeSpan ListNormal { set; get; }
        public TimeSpan DetachedEntityDisabledTracking { set; get; }
        public TimeSpan DetachedEntityNormal { set; get; }
    }

    public static class Test
    {
        public static void RunTests()
        {
            startDb();

            var results = new List<Performance>();
            var runs = 20;
            for (int i = 0; i < runs; i++)
            {
```

```
                Console.WriteLine("\nRun {0}", i + 1);

                var tsListDisabledTracking = PerformanceHelper.RunActionMeasurePerformance(() =>
updateListTotalPriceDisabledTracking());
                var tsListNormal = PerformanceHelper.RunActionMeasurePerformance(() =>
updateListTotalPriceNormal());
                var tsDetachedEntityDisabledTracking = PerformanceHelper.RunActionMeasurePerformance(()
=> updateDetachedEntityTotalPriceDisabledTracking());
                var tsDetachedEntityNormal = PerformanceHelper.RunActionMeasurePerformance(() =>
updateDetachedEntityTotalPriceNormal());
                results.Add(new Performance
                {
                    ListDisabledTracking = tsListDisabledTracking,
                    ListNormal = tsListNormal,
                    DetachedEntityDisabledTracking = tsDetachedEntityDisabledTracking,
                    DetachedEntityNormal = tsDetachedEntityNormal
                });
            }

            var detachedEntityDisabledTrackingAvg = results.Average(x =>
x.DetachedEntityDisabledTracking.TotalMilliseconds);
            Console.WriteLine("detachedEntityDisabledTrackingAvg: {0} ms.",
detachedEntityDisabledTrackingAvg);

            var detachedEntityNormalAvg = results.Average(x =>
x.DetachedEntityNormal.TotalMilliseconds);
            Console.WriteLine("detachedEntityNormalAvg: {0} ms.", detachedEntityNormalAvg);

            var listDisabledTrackingAvg = results.Average(x =>
x.ListDisabledTracking.TotalMilliseconds);
            Console.WriteLine("listDisabledTrackingAvg: {0} ms.", listDisabledTrackingAvg);

            var listNormalAvg = results.Average(x => x.ListNormal.TotalMilliseconds);
            Console.WriteLine("listNormalAvg: {0} ms.", listNormalAvg);
        }

        private static void updateDetachedEntityTotalPriceNormal()
        {
            using (var context = new MyContext(withTracking: true))
            {
                var detachedEntity = new Factor { Id = 1, TotalPrice = 10 };

                var attachedEntity = context.Factors.Find(detachedEntity.Id);
                if (attachedEntity != null)
                {
                    attachedEntity.TotalPrice = 100;

                    context.SaveChanges();
                }
            }
        }

        private static void updateDetachedEntityTotalPriceDisabledTracking()
        {
            using (var context = new MyContext(withTracking: false))
            {
                var detachedEntity = new Factor { Id = 2, TotalPrice = 10 };
                detachedEntity.TotalPrice = 200;

                context.CustomUpdate(detachedEntity); // custom update with change tracking disabled.
                context.SaveChanges();
            }
        }

        private static void updateListTotalPriceNormal()
        {
            using (var context = new MyContext(withTracking: true))
            {
                foreach (var item in context.Factors)
                {
                    item.TotalPrice += 10; // normal update with change tracking enabled.
                }
                context.SaveChanges();
            }
        }

        private static void updateListTotalPriceDisabledTracking()
        {
            using (var context = new MyContext(withTracking: false))
            {
                foreach (var item in context.Factors)
```

```
                {
                    item.TotalPrice += 10;
                    //نیازی به این دو سطر نیست
                    //context.ChangeTracker.DetectChanges();  // هربار باید محاسبه صورت گیرد در غیراینصورت وضعیت تغییر نیافته گزارش می‌شود
                    //context.CustomUpdate(item); // custom update with change tracking disabled.
                }
                context.ChangeTracker.DetectChanges();  // در غیراینصورت وضعیت تغییر نیافته گزارش می‌شود
                context.SaveChanges();
            }
        }

        private static void startDb()
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyContext, Configuration>());
            // Forces initialization of database on model changes.
            using (var context = new MyContext())
            {
                context.Database.Initialize(force: true);
            }
        }
    }

    public class PerformanceHelper
    {
        public static TimeSpan RunActionMeasurePerformance(Action action)
        {
            var stopwatch = new Stopwatch();
            stopwatch.Start();

            action();

            stopwatch.Stop();
            return stopwatch.Elapsed;
        }
    }
}
```

نتیجه این آزمایش بعد از 20 بار اجرا و اندازه گیری:

```
detachedEntityDisabledTrackingAvg: 22.32089 ms.
detachedEntityNormalAvg: 54.546815 ms.
listDisabledTrackingAvg: 413.615445 ms.
listNormalAvg: 393.194625 ms.
```

در حالت کار با یک شیء ساده، به روز رسانی حالت منقطع بسیار سریعتر است (چون یکبار رفت و برگشت کمتری داره به دیتابیس).
در حالت کار با لیستی از اشیاء دریافت شده از بانک اطلاعاتی، به روز رسانی حالت متصل به Context سریعتر است.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۹ ۱۷:۱۱

طول عمر یک شیء، کاری به خاموش یا روشن بودن سیستم ردیابی ندارد.
مقصود از متصل و غیرمتصلی که در اینجا عنوان شده، فعال و غیرفعال سازی مباحث Tracking در Context است و وضعیت یک شیء نسبت به Context (به علاوه خاموش کردن lazy loading و غیره). مثلا اگر خاصیت Name رو تغییر دادید، Context می‌دونه اتفاقی رخ داده یا اینکه وضعیت رو unchanged یا detached گزارش می‌ده؟

نویسنده: ایمان محمدی
تاریخ: ۱۳۹۲/۰۳/۰۹ ۱۷:۳۱

در کد آپدیت بالا هدف نشان دادن نحوه بروز رسانی یک شی اتچ شده بود که به اشتباه متد آپدیت رو قراردادم. (اصلاح شد)

```
T attachedEntity = set.Find(entity.Id);
var attachedEntry = dbContext.Entry(attachedEntity);
attachedEntry.CurrentValues.SetValues(entity);
```

نویسنده: ایمان محمدی
تاریخ: ۱۳۹۲/۰۳/۰۹ ۱۷:۴۸

در حالت نه آنچنان Detached ! (دریافت یک لیست از Context ایی که ردیابی نداره)

....
در متن هم گفته شد وقتی همه چیز رو خاموش کردیم ما باید وضعیت موجودیت رو مشخص کنیم. مثلا لیستی از اشیا رو می‌سازیم کاربر یکی رو انتخاب می‌کنه تغییر می‌ده و ما در لحظه ذخیره سازی وضعیت اونو به "تغییر داده شده" تغییر می‌دیم.

```
dbContext.Entry(entity).State = EntityState.Modified;
```

در حقیقت همه اشیا CLR ساده هستند و در موقع درخواست ثبت تغییرات از ef کمک می‌گیریم.

نویسنده: arezoo
تاریخ: ۱۳۹۲/۰۳/۳۰ ۱۶:۲۸

سلام . من یه راهنمایی میخوام ممنون میشم کمکم کنین برا آپدیت مشکل دارم،کدش رو به این صورت نوشتم اما چنین اروری میده(An object with the same key already exists in the ObjectStateManager. The ObjectStateManager cannot track multiple objects with the same key.)

```
public void InsertOrUpdate(Core.Models.PersonelAction personelAction {
            if (personelAction.Id == default(int))
            {
                _unitOfWork.Entry(personelAction).State = EntityState.Added;
            }
            else
            {
                 _unitOfWork.Entry(personelAction).State=EntityState.Modified;

            }
        }
```

نویسنده: ایمان محمدی
تاریخ: ۱۳۹۲/۰۳/۳۰ ۱۷:۴۹

احتمالا رابطه‌های PersonelAction با دیگر موجودیت باعث این ارور شده، کدی که اینجا نوشتید برای پاسخ دادن خیلی ناقصه.

نویسنده: arezoo
تاریخ: ۱۳۹۲/۰۳/۳۰ ۲۲:۲۴

ممنون که جواب دادین ،موجودیت‌های من به این صورت هستن

```
(...
```

,

```
customerAction(customerId,Money,PersonelAction
```

```
(PersonelAction(Id,User,Pass,ReceivedMoeny
```

در واقع PersonelAction کارمندی هست که بعد از هر واریز مبلغ به موسسه باید مبلغ دریافتی کارمند مربوطه آپدیت شه و در

جدول customerAction هم مشخص میشه که کدوم کارمند دریافت وجه رو انجام داده

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۳/۳۰ ۲۳:۲۱

- ممکنه نتونسته باشید unit of work رو درست مدیریت کنید و در پاس دادن اون به لایه‌های مختلف، چند وهله ازش ساخته شده. در این حالت خطای فوق رو می‌گیرد.
- ممکنه شیء در حال استفاده قبلا توسط Context بارگذاری شده و هنوز هست، مثلا در یک متد GetAll الان دوباره می‌خواهید اضافه‌اش کنید که نمی‌شود. یا مدیریت ناصحیح Context و باز نگه داشتن بیش از حد آن به ازای کل برنامه یا چندین فرم مختلف با هم که باز هم سبب این مساله می‌شود.
- یا حتی ممکنه وضعیت موجودیت EntityState.Detached باشه که باید اول Attach شود. (وضعیت اتصال موجودیت‌ها رو ابتدا چک کنید)
- اگر قراره موجودیت جدیدی اضافه بشه چرا از متد Add استفاده نکردید؟

نویسنده: arezoo
تاریخ: ۱۳۹۲/۰۳/۳۱ ۳:۷

بله حق با شما بود این موجودیت توی یکی از فرم‌ها به context اضافه شده بود
در مورد add کردن مشکلی نداشتم
میشه در مورد مدیریت unit of work یه توضیحی بدین؟ما توی هر فرم برای ذخیره‌ی تغییرات آیا یک instance از unit of work میسازیم؟

نویسنده: محسن جمشیدی
تاریخ: ۱۳۹۲/۰۳/۳۱ ۸:۴۵

این خطا زمانی پیش می‌آید که personelAction ای قبلا با همین Id در DbContext شما موجود باشد و شما بخواهید وهله ای (نمونه ای) دیگر از personelAction را به DbContext جاری وارد کرده و State آن را Modifiedقرار بدید. بنابراین در else نیاز هست که چک کنید personelAction.Id قبلا در DbContext موجود است یا خیر

نویسنده: محسن جمشیدی
تاریخ: ۱۳۹۲/۰۳/۳۱ ۸:۵۲

بستگی به فرم داره. اگر شما دو فرم داشته باشید که یکی Masterهست و دیگری Detail مثلا فرم سفارش و اقلام سفارش در این حالت می‌بایست از یک UnitOfWork برای دو فرم استفاده کنید چراکه دو فرم به هم وابسته هستند و نیاز هست که یکجا ذخیره شوند. نمی‌شود که اقلام سفارش ذخیره شود ولی خود سفارش ذخیره نشود

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۳/۳۱ ۹:۹

یعنی الان یک Context به ازای کل برنامه دارید که دچار این تداخل شدید؟ معمولا در WPF و همچنین WinForms این Context به ازای هر فرم تعریف می‌شود و با بسته شدن آن تخریب.

حالا یک سؤال مهم! به نظر شما در اولین سؤالی که پرسیدید، یک شخص چطور می‌بایستی ساختار کار شما رو که بر مبنای یک Context در کل برنامه است، حدس می‌زد و عیب یابی می‌کرد؟!

نویسنده: علیرضا
تاریخ: ۱۳۹۲/۰۴/۰۱ ۱۸:۵۶

مطرح کردید:

در حالت کار با لیستی از اشیاء دریافت شده از بانک اطلاعاتی، به روز رسانی حالت متصل به Context سریعتر است.
چرا؟ چه توجیهی برای این هست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۰۱ ۱۸:۵۸

چون عناصر آن متصل هستند. یعنی Context نیازی به اتصال مجدد و بررسی وضعیت تک تک عناصر آن برای تولید SQL صحیح ندارد و همه چیز از پیش محاسبه شده است (این دو مورد اتصال و محاسبه وضعیت، زمانبر است؛ برای 20 عنصر در محاسبات فوق نزدیک به 20 میلی ثانیه تفاوتش است).

نویسنده: علیرضا
تاریخ: ۱۳۹۲/۰۴/۰۱ ۲۳:۲۲

به نظر من عنوان صحیح نیست. اصولا EF نامتصل هست. اینی که اینجا بحث شده در حقیقت دو حالت Dispose شده یا نشده Context هست نه چیزی به عنوان Connected یا Disconnected

نویسنده: علیرضا
تاریخ: ۱۳۹۲/۰۴/۰۱ ۲۳:۳۳

اگر این توضیح صحیح باشه باید برای هر تعداد از عناصر هم صحیح باشه. یعنی با هر تعداد شیی در لیست. خوب حالا فرض کنید لیست ما 1 عنصر داره. این یعنی همون حالت "کار با یک شیی ساده". چرا در این حالت توضیحی که شما گفتید صادق نیست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۰۱ ۲۳:۵۱

- سورس آزمایش به عمد ارسال شد، تا بتونید خودتون اجراش کنید و اندازه گیری کنید. این‌ها چشم بندی نبوده یا نظر شخصی نیست. یک سری اندازه گیری است.
- توضیح دادم در انتهای همان آزمایش. برای تکرار مجدد: چون یکبار رفت و برگشت کمتری داره به دیتابیس. چون تغییر State یک شیء و ورود آن به سیستم ردیابی، خیلی سریعتر است از واکشی اطلاعات از بانک اطلاعاتی. اما در مورد لیستی از اشیاء، توسط context.Factors سیستم EF دسترسی به IDها پیدا می‌کنه (در هر دو حالت متصل و منقطع). اگر سیستم ردیابی خاموش شود، برای اتصال مجدد این‌ها زمان خواهد برد (چون IDهای دریافت شده از بانک اطلاعاتی ردیابی نمی‌شوند)، اما در حالت متصل، همان بار اولی که کوئری گرفته شده، همانجا اتصال هم برقرار شده و در حین به روز رسانی اطلاعات می‌داند چه تغییراتی رخ داده و چگونه سریعا باید محاسبات رو انجام بده. اما در حالت منقطع توسط متد DetectChanges تازه شروع به اتصال و محاسبه می‌کند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۰۲ ۰:۰

واژه‌های Attached و Detached مانند EntityState.Detached جزو فرهنگ لغات EF هستند. این معانی هم مرتبط هستند با این کلمات و نه هیچ برداشت دیگری.

نویسنده: ایمان محمدی
تاریخ: ۱۳۹۲/۰۴/۰۲ ۱:۱۷

احتمالا برداشت شما متصل بودن به دیتابیس است ، اینجا منظور متصل بودن یک شی به DbContext است. که این متصل بودن مزایایی همچون ردگیری تغییرات توسط DbContext را دارد.

نویسنده: علیرضا
تاریخ: ۱۳۹۲/۰۴/۰۲ ۱۱:۴۸

درسته شاید پیدا کردن 2 واژه فارسی متفاوت برای Attached و Connected کمی سخت باشه. زبان فارسی در رشته ما کمی ناکارآمده.

نویسنده: علیرضا
تاریخ: ۱۳۹۲/۰۴/۰۲ ۱۱:۵۷

درسته. من کد رو با دقت نخونده بودم مخصوصا متد CustomUpdate . ممنون از توضیح دوباره.

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۰۴/۲۱ ۱۴:۴۵

در حالت غیر متصل؛ روش پیگیری و مدیریت تغییرات Entityهای سمت کلاینت و اعمال اونها سمت سرور به چه صورته؟
منظورم اینه که فرض کنید من یک موجودیت سفارش با چندین آیتم سفارش دارم(در واقع دو موجودیت) که کاربر ممکنه وقتی یک سفارش رو ویرایش میکنه، یک آیتم رو اضافه کنه، یک آیتم رو حذف و یکی رو ویرایش کنه، در این حالت چطوری همه این تغییرات در یک UOW و توسط یک .SaveChanges() در سمت سرور اعمال میشن؟

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۰۴/۲۱ ۱۸:۳۴

قسمت Find متد Update شما در حالت detached اضافی است. یعنی اگر می‌دونید که این Id در دیتابیس وجود داره نیازی به Findاش نیست. فقط State اون رو تغییر بدید کار می‌کنه .
آیا این قسمت از کد برای تشخیص objectهای تکراری در گراف objectهای ما نیست؟ ونبایستی uncomment شود؟ طبق این لینک .

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۲۱ ۱۸:۵۳

صورت مساله در *اینجا* بر اساس خاموش کردن سیستم ردیابی بود (به سازنده کلاس توجه کنید). مابقی جملات هم بر این اساس نوشته شد. زمانیکه سیستم ردیابی خاموش شود، دیگر گراف objectایی از ابتدای کار وجود ندارد؛ چیزی ردیابی نمی‌شود.
بنابراین Find در حلقه‌ایی بر اساس آپدیت کردن مثلا 100 شیء منقطع جدید، مجبور میشه 100 بار به دیتابیس مراجعه کنه؛ چون EF هنوز از وجود آن‌ها مطلع نشده و کشی رو برای نگهداری اطلاعات آن‌ها تشکیل نداده.

نویسنده: محمد واحدی
تاریخ: ۱۳۹۲/۰۷/۰۹ ۱۳:۴۹

منم همین مشکل را دارم. چون می‌خوام از WCF استفاده کنم مجبورم از حالت غیر متصل استفاده کنم. فرم Header-Detail هست چند تا از آیتمها تغییر کردند یا حذف شدند. می‌خوام آبجکت هدر را وقتی میدم به سرور با بچه هاش بره چه کار باید کنم لطفا راهنمایی کنید؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۷/۱۱ ۸:۳۸

Using WCF Data Services 5.6.0 with Entity Framework 6