عنوان: **ایجاد سرویس چندلایه‌ی WCF با Entity Framework در قالب پروژه - 10**
نویسنده: حامد قنادی
تاریخ: ۱۳۹۳/۰۴/۲۷ ۲۲:۲۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, WCF

بهره‌گیری از یک تابع پویا برای افزودن، ویرایش

در مثال‌های گذشته دیدید که برای هر کدام از عمل‌های درج، ویرایش و حذف، تابع‌های مختلفی نوشته بودیم که این‌کار هنگامی‌که یک پروژه‌ی بزرگ در دست داریم زمان‌بر خواهد بود. چه بسا یک جدول بزرگ داشته باشیم و بخواهیم در هر فرمی، ستون یا ستون‌های خاص به‌روزرسانی شوند. برای رفع این نگرانی افزودن تابع زیر به سرویس‌مان گره‌گشا خواهد بود.

```
public bool AddOrUpdateOrDelete<TEntity>(TEntity newItem, bool updateIsNull) where TEntity : class
        {
            try
            {
                var dbMyNews = new dbMyNewsEntities();
                if (updateIsNull)
                    dbMyNews.Set<TEntity>().AddOrUpdate(newItem);
                else
                {
                    dbMyNews.Set<TEntity>().Attach(newItem);
                    var entry = dbMyNews.Entry(newItem);
                    foreach (
                        var pri in newItem.GetType().GetProperties()
                            .Where(pri =>
(pri.GetGetMethod(false).ReturnParameter.ParameterType.IsSerializable &&
                                                pri.GetValue(newItem, null) != null)))
                    {
                        entry.Property(pri.Name).IsModified = true;
                    }
                }
                dbMyNews.SaveChanges();
                return true;
            }
            catch (Exception)
            {
                return false;
            }
        }
```

این تابع دو پارامتر ورودی newItem و updateIsNull دارد که نخستین، همان نمونه‌ای از Entity است که قصد افزودن، ویرایش یا حذف آن‌را داریم و با دومی مشخص می‌کنیم که آیا ستون‌هایی که دارای مقدار null هستند نیز در موجودیت اصلی به‌هنگام شوند یا خیر. این پارامتر جهت رفع این مشکل گذاشته شده است که هنگامی‌که قصد به‌هنگام‌کردن یک یا چند ستون خاص را داشتیم و تابع update را به گونه‌ی زیر صدا می‌زدیم، بقیه‌ی ستون‌ها مقدار null می‌گرفت.

```
var news = new tblNews();
news.tblCategoryId = 2;
news.tblNewsId = 1;
MyNews.EditNews(news);
```

توسط تکه کد بالا، ستون tblCategoryId از جدول tblNews با شرط این‌که شناسه‌ی جدول آن برابر با 1 باشد، مقدار 2 خواهد گرفت. ولی بقیه‌ی ستون‌های آن به علت این‌که مقداری برای آن مشخص نکرده ایم، مقدار null خواهد گرفت. راهی که برای حل آن استفاده می‌کردیم، به این صورت بود:

```
var news = MyNews.GetNews(1);
 news.tblCategoryId = 2;
MyNews.EditNews(news)
```

در این روش یک رفت و برگشت بی‌هوده به WCF انجام خواهد شد در حالتی که ما اصلاً نیازی به مقدار ستون‌های دیگر نداریم و اساساً کاری روی آن نمی‌خواهیم انجام دهیم.
در تابع AddOrUpdateOrDelete نخست بررسی می‌کنیم که آیا این‌که ستون‌هایی که مقدار ندارند، در جدول اصلی هم مقدار null بگیرند برای ما مهم است یا نه. برای نمونه هنگامی‌که می‌خواهیم سطری به جدول بیفزاییم یا این‌که واقعاً بخواهیم مقدار دیگر

ستون‌ها برابر با null شود. در این صورت همان متد AddOrUpdate از Entity Framework اجرا خواهد شد. حالت دیگر که در حذف و ویرایش از آن بهره می‌بریم با یک دستور foreach همه‌ی پروپرتی‌هایی که Serializable باشد (که در این صورت پروپرتی‌های virtual حذف خواهد شد) و مقدار آن نامساوی با null باشد، در حالت ویرایش خواهند گرفت و در نتیجه دیگر ستون‌ها ویرایش نخواهد شد. این دستور دیدگاه جزءنگر دستور زیر است که کل موجودیت را در وضعیت ویرایش قرار می‌داد:

```
dbMyNews.Entry(news).State = EntityState.Modified;
```

با آن‌چه گفته شد، می‌توانید به جای سه تابع زیر:

```
public int AddNews(tblNews News)
        {
            dbMyNews.tblNews.Add(News);
            dbMyNews.SaveChanges();
            return News.tblNewsId;
        }

        public bool EditNews(tblNews News)
        {
            try
            {
                dbMyNews.Entry(News).State = EntityState.Modified;
                dbMyNews.SaveChanges();
                return true;
            }
            catch (Exception exp)
            {
                return false;
            }
        }

        public bool DeleteNews(int tblNewsId)
        {
            try
            {
                tblNews News = dbMyNews.tblNews.FirstOrDefault(p => p.tblNewsId == tblNewsId);
                News.IsDeleted = true;
                dbMyNews.SaveChanges();
            return true;
            }
            catch (Exception exp)
            {
                return false;
            }
        }
```

تابع زیر را بنویسید:

```
public bool AddOrEditNews(tblNews News)
        {
            return AddOrUpdateOrDelete(News, News.tblNewsId == 0);
        }
```

به همین سادگی. من در این‌جا شرط کردم فقط در حالت درج، از قسمت نخست تابع بهره گرفته شود.
در سمت برنامه از این تابع برای عمل درج، ویرایش و حذف به سادگی و بدون نگرانی استفاده می‌کنید. برای نمونه جهت حذف در یک خط به این صورت می‌نویسید:

```
MyNews.AddOrEditNews (new tblNews { tblNewsId = 1, IsDeleted =true });
```

در بخش پسین آموزش، پیرامون ایجاد امنیت در WCF خواهم نوشت.

نظرات خوانندگان

نویسنده: محمد آزاد
تاریخ: ۱۳۹۳/۰۴/۲۸ ۳:۲۵

به نظرتون این جوری اصل SRP رو نقض نکردیم؟

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۴/۲۸ ۱۱:۴۱

خود EF متدی به نام AddOrUpdate داره: http://msdn.microsoft.com/en-us/library/hh846520%28v=vs.103%29.aspx

در اصل تک مسئولیتی، مسئولیت به دلیل تغییر یک کلاس ترجمه میشه. بنابراین در این اصل می‌گن که یک کلاس باید فقط یک دلیل برای تغییر داشته باشه. برای مثال کلاسی که هم اطلاعات گزارشی رو تهیه می‌کنه و هم اون رو پرینت می‌کنه، دو مسئولیت رو به عهده گرفته که میشه از هم جداشون کرد. اما در اینجا یک مسئولیت به روز رسانی اطلاعات یک موجودیت خاص بیشتر در کار نیست. دلیل دومی برای تغییر کلاس نداریم. وابستگی خارجی دومی نداره.