عنوان: **لغو Lazy Loading در حین کار با AutoMapper و Entity Framework**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۲/۰۶ ۱۹:۱۳
آدرس: www.dotnettips.info
گروه‌ها: MVC, Entity framework, AutoMapper

پیشنیازها

- مطالعه‌ی مطالب گروه AutoMapper در سایت، دید خوبی را برای شروع به کار با آن فراهم می‌کنند و در اینجا قصد تکرار این مباحث پایه‌ای را نخواهیم داشت. هدف بیشتر بررسی یک سری نکات پیشرفته‌تر و عمیق‌تر است از کار با AutoMapper.
- آشنایی با Lazy loading و Eager loading در حین کار با EF

ساختار و پیشنیازهای برنامه‌ی مطلب جاری

جهت سهولت پیگیری مطلب و تمرکز بیشتر بر روی مفاهیم اصلی مورد بحث، یک برنامه‌ی کنسول را آغاز کرده و سپس بسته‌های نیوگت ذیل را به آن اضافه کنید:

```
PM> install-package AutoMapper
PM> install-package EntityFramework
```

به این ترتیب بسته‌های AutoMapper و EF به پروژه‌ی جاری اضافه خواهند شد.

آشنایی با ساختار مدل‌های برنامه

در اینجا ساختار جداول مطالب یک بلاگ را به همراه نویسندگان آن‌ها، مشاهده می‌کنید:

```
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; }
}

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

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

هر کاربر می‌تواند تعدادی مطلب تهیه کند و هر مطلب توسط یک کاربر نوشته شده‌است.

هدف از این مثال

فرض کنید اطلاعاتی که قرار است به کاربر نمایش داده شوند، توسط ViewModel ذیل تهیه می‌شود:

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

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

در اینجا می‌خواهیم اولین کاربر ثبت شده را یافته و سپس لیست مطالب آن‌را نمایش دهیم. همچنین می‌خواهیم این کوئری تهیه شده به صورت خودکار اطلاعاتش را بر اساس ساختار ViewModel ایی که مشخص کردیم (و این ViewModel الزاما تمام عناصر آن با عناصر مدل اصلی یکی نیست)، بازگشت دهیم.

تهیه نگاشت‌های AutoMapper

برای مدیریت بهتر نگاشت‌های AutoMapper توصیه شده‌است که کلاس‌های Profile ایی را به شکل ذیل تهیه کنیم:

```
public class TestProfile : Profile
{
    protected override void Configure()
    {
        this.CreateMap<User, UserViewModel>();
    }

    public override string ProfileName
    {
        get { return this.GetType().Name; }
    }
}
```

کار با ارث بری از کلاس پایه Profile کتابخانه‌ی AutoMapper شروع می‌شود. سپس باید متد Configure آن‌را بازنویسی کنیم. در اینجا می‌توان با استفاده از متدی مانند Create مشخص کنیم که قرار است اطلاعاتی با ساختار شیء User، به اطلاعاتی با ساختار از نوع شیء UserViewModel به صورت خودکار نگاشت شوند.

ثبت و معرفی پروفایل‌های AutoMapper

پس از تهیه‌ی پروفایل مورد نیاز، در ابتدای برنامه با استفاده از متد Mapper.Initialize، کار ثبت این تنظیمات صورت خواهد گرفت:

```
Mapper.Initialize(cfg => // In Application_Start()
{
    cfg.AddProfile<TestProfile>();
});
```

روش متداول کار با AutoMapper جهت نگاشت اطلاعات User به ViewModel آن

در ادامه به نحو متداولی، ابتدا اولین کاربر ثبت شده را یافته و سپس با استفاده از متد Mapper.Map اطلاعات این شیء user به ViewModel آن نگاشت می‌شود:

```
using (var context = new MyContext())
{
    var user1 = context.Users.FirstOrDefault();
    if (user1 != null)
    {
        var uiUser = new UserViewModel();
        Mapper.Map(source: user1, destination: uiUser);

        Console.WriteLine(uiUser.Name);
        foreach (var post in uiUser.BlogPosts)
        {
            Console.WriteLine(post.Title);
        }
    }
}
```

تا اینجا اگر برنامه را اجرا کنید، مشکلی را مشاهده نخواهید کرد، اما این کدها سبب اجرای حداقل دو کوئری خواهند شد:
الف) یافتن اولین کاربر

ب) واکشی لیست مطالب او در یک کوئری دیگر

کاهش تعداد رفت و برگشت‌ها به سرور با استفاده از متدهای ویژه‌ی AutoMapper

در حالت متداول کار با EF، با استفاده از متد Include می‌توان این Lazy loading را لغو کرد و در همان اولین کوئری، مطالب کاربر یافت شده را نیز دریافت نمود:

```
var user1 = context.Users.Include(user => user.BlogPosts).FirstOrDefault();
```

و سپس این اطلاعات را توسط AutoMapper نگاشت کرد.
در این حالت، AutoMapper برای ساده سازی این مراحل، متدهای Project To را معرفی کرده‌است:

```
var uiUser = context.Users.Project().To<UserViewModel>().FirstOrDefault();
```

در اینجا نیز Lazy loading لغو شده و به صورت خودکار جوینی به جدول مطالب کاربران ایجاد خواهد شد.
بنابراین با استفاده از متدهای Project To می‌توان از ذکر Includeهای EF صرفنظر کرد و همچنین دیگر نیازی به نوشتن متد Select جهت نگاشت دستی خواص مورد نظر به خواص ViewModel نیست.

کدهای کامل این قسمت را از اینجا می‌توانید دریافت کنید:
AM_Sample01.zip

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۲/۰۹ ۱۱:۴۶

یک نکته‌ی تکمیلی

فراخوانی متدهای متداول EF مانند ToList و FirstOrDefault و امثال آن، *اگر به همراه Select نباشند* ، سبب واکشی تمام فیلدها و خواص جدول مورد نظر می‌شوند. اما اگر از متد Project To مانند مطلب فوق استفاده کنید، واکشی انجام شده به صورت خودکار تنها بر اساس خواص موجود در ViewModel صورت می‌گیرد و به این ترتیب حجم کمتری از اطلاعات رد و بدل خواهد شد (چون AutoMapper کار نوشتن Select را بر اساس خواص ViewModel، در پشت صحنه انجام داده‌است و این Select حاوی تمام خواص کلاس جدول مورد استفاده نیست).

نویسنده: غلامرضا ربال
تاریخ: ۱۳۹۴/۰۶/۱۳ ۱۲:۲۲

با تشکر.
هنگام استفاده از ValueResolver یا ValueConverter برای تبدیل Datetime به رشته که در مقاله " تبدیلگر تاریخ شمسی برای AutoMapper " مطرح کردید ، امکان استفاده از متدها متدها Project و To وجود ندارد و خطای

```
LINQ to Entities does not recognize the method 'System.String
```

آیا راه حلی نیست تا بتوان از این دو امکان کنار هم استفاده کرد و مجبور نشویم که به روش قبل با select این کار را انجام دهیم؟
این مسئله به قدرت Linq ربط دارد و آیا امکانی در این کتابخانه موجود نیست برای حل این مشکل؟
ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۶/۱۳ ۱۳:۴

تبدیلگرها هم در نهایت باید تبدیل به SQL شوند وگرنه قابلیت استفاده در EF را نخواهند داشت. برای این حالت‌های خاص، متدهای ProjectUsing, ConstructProjectionUsing پیش بینی شده‌اند (^ و ^).

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۷/۳۰ ۱۵:۵۴

جهت اطلاع

کلیه مثال‌های این دوره برای استفاده از آخرین نگارش AutoMapper در مخزن کد آن به روز شدند.