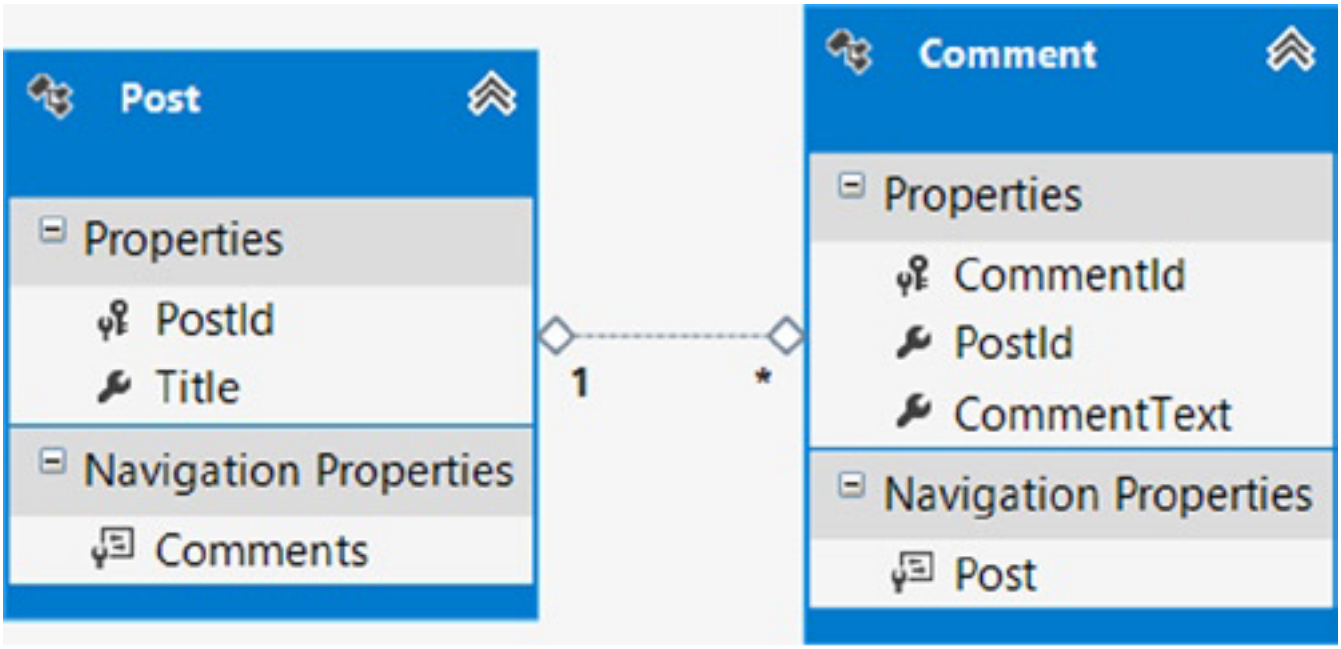
عنوان: **استفاده از EF در اپلیکیشن های N-Tier : قسمت دوم**
نویسنده: آرمین ضیاء
تاریخ: ۱۳۹۲/۱۱/۰۷ ۷:۴۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, Tips, WCF, N-Layer Architecture, Entity Framework 6

در قسمت قبل معماری اپلیکیشن‌های N-Tier و بروز رسانی موجودیت‌های منفصل توسط Web API را بررسی کردیم. در این قسمت بروز رسانی موجودیت‌های منفصل توسط WCF را بررسی می‌کنیم.

بروز رسانی موجودیت‌های منفصل توسط WCF

سناریویی را در نظر بگیرید که در آن عملیات CRUD توسط WCF پیاده سازی شده اند و دسترسی داده‌ها با مدل Code-First انجام می‌شود. فرض کنید مدل اپلیکیشن مانند تصویر زیر است.



همانطور که می‌بینید مدل ما متشکل از پست‌ها و نظرات کاربران است. برای ساده نگاه داشتن مثال جاری، اکثر فیلدها حذف شده اند. مثلا متن پست ها، نویسنده، تاریخ و زمان انتشار و غیره. می‌خواهیم تمام کد دسترسی داده‌ها را در یک سرویس WCF پیاده سازی کنیم تا کلاینت‌ها بتوانند عملیات CRUD را توسط آن انجام دهند. برای ساختن این سرویس مراحل زیر را دنبال کنید.
در ویژوال استودیو پروژه جدیدی از نوع Class Library بسازید و نام آن را به Recipe2 تغییر دهید.
با استفاده از NuGet Package Manager کتابخانه Entity Framework 6 را به پروژه اضافه کنید.
سه کلاس با نام‌های Post, Comment و Recipe2Context به پروژه اضافه کنید. کلاس‌های Post و Comment موجودیت‌های مدل ما هستند که به جداول متناظرشان نگاشت می‌شوند. کلاس Recipe2Context آبجکت DbContext ما خواهد بود که بعنوان درگاه عملیاتی EF عمل می‌کند. دقت کنید که خاصیت‌های لازم WCF یعنی DataContract و DataMember در کلاس‌های موجودیت‌ها بدرستی استفاده می‌شوند. لیست زیر کد این کلاس‌ها را نشان می‌دهد.

```
[DataContract(IsReference = true)]
public class Post
{
    public Post()
    {
        comments = new HashSet<Comments>();
    }

    [DataMember]
    public int PostId { get; set; }
```

```
    [DataMember]
    public string Title { get; set; }
    [DataMember]
    public virtual ICollection<Comment> Comments { get; set; }
}

[DataContract(IsReference=true)]
public class Comment
{
    [DataMember]
    public int CommentId { get; set; }
    [DataMember]
    public int PostId { get; set; }
    [DataMember]
    public string CommentText { get; set; }
    [DataMember]
    public virtual Post Post { get; set; }
}

public class EFRecipesEntities : DbContext
{
    public EFRecipesEntities() : base("name=EFRecipesEntities") {}

    public DbSet<Post> posts;
    public DbSet<Comment> comments;
}
```

یک فایل App.config به پروژه اضافه کنید و رشته اتصال زیر را به آن اضافه نمایید.

```
<connectionStrings>
  <add name="Recipe2ConnectionString"
    connectionString="Data Source=.;
    Initial Catalog=EFRecipes;
    Integrated Security=True;
    MultipleActiveResultSets=True"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

حال یک پروژه WCF به Solution جاری اضافه کنید. برای ساده نگاه داشتن مثال جاری، نام پیش فرض Service1 را بپذیرید. فایل IService1.cs را باز کنید و کد زیر را با محتوای آن جایگزین نمایید.

```
[ServiceContract]
public interface IService1
{
    [OperationContract]
    void Cleanup();
    [OperationContract]
    Post GetPostByTitle(string title);
    [OperationContract]
    Post SubmitPost(Post post);
    [OperationContract]
    Comment SubmitComment(Comment comment);
    [OperationContract]
    void DeleteComment(Comment comment);
}
```

فایل Service1.svc.cs را باز کنید و کد زیر را با محتوای آن جایگزین نمایید. بیاد داشته باشید که پروژه Recipe2 را ارجاع کنید و فضای نام آن را وارد نمایید. همچنین کتابخانه EF 6 را باید به پروژه اضافه کنید.

```
public class Service1 : IService
{
    public void Cleanup()
    {
        using (var context = new EFRecipesEntities())
        {
            context.Database.ExecuteSqlCommand("delete from [comments]");
            context. Database.ExecuteSqlCommand ("delete from [posts]");
        }
    }

    public Post GetPostByTitle(string title)
```

```
    {
        using (var context = new EFRecipesEntities())
        {
            context.Configuration.ProxyCreationEnabled = false;
            var post = context.Posts.Include(p => p.Comments).Single(p => p.Title == title);
            return post;
        }
    }

    public Post SubmitPost(Post post)
    {
        context.Entry(post).State =
            // if Id equal to 0, must be insert; otherwise, it's an update
            post.PostId == 0 ? EntityState.Added : EntityState.Modified;
        context.SaveChanges();
        return post;
    }

    public Comment SubmitComment(Comment comment)
    {
        using (var context = new EFRecipesEntities())
        {
            context.Comments.Attach(comment);
            if (comment.CommentId == 0)
            {
                // this is an insert
                context.Entry(comment).State = EntityState.Added);
            }
            else
            {
                // set single property to modified, which sets state of entity to modified, but
                // only updates the single property – not the entire entity
                context.entry(comment).Property(x => x.CommentText).IsModified = true;
            }
            context.SaveChanges();
            return comment;
        }
    }

    public void DeleteComment(Comment comment)
    {
        using (var context = new EFRecipesEntities())
        {
            context.Entry(comment).State = EntityState.Deleted;
            context.SaveChanges();
        }
    }
}
```

در آخر پروژه جدیدی از نوع Windows Console Application به Solution جاری اضافه کنید. از این اپلیکیشن بعنوان کلاینتی برای تست سرویس WCF استفاده خواهیم کرد. فایل program.cs را باز کنید و کد زیر را با محتوای آن جایگزین نمایید. روی نام پروژه کلیک راست کرده و گزینه Add Service Reference را انتخاب کنید، سپس ارجاعی به سرویس Service1 اضافه کنید. رفرنسی هم به کتابخانه کلاس‌ها که در ابتدای مراحل ساختید باید اضافه کنید.

```
class Program
{
    static void Main(string[] args)
    {
        using (var client = new ServiceReference2.Service1Client())
        {
            // cleanup previous data
            client.Cleanup();
            // insert a post
            var post = new Post { Title = "POCO Proxies" };
            post = client.SubmitPost(post);
            // update the post
            post.Title = "Change Tracking Proxies";
            client.SubmitPost(post);
            // add a comment
            var comment1 = new Comment { CommentText = "Virtual Properties are cool!", PostId =
post.PostId };
            var comment2 = new Comment { CommentText = "I use ICollection<T> all the time", PostId =
post.PostId };
            comment1 = client.SubmitComment(comment1);
            comment2 = client.SubmitComment(comment2);
            // update a comment
```

```
            comment1.CommentText = "How do I use ICollection<T>?";
            client.SubmitComment(comment1);
            // delete comment 1
            client.DeleteComment(comment1);
            // get posts with comments
            var p = client.GetPostByTitle("Change Tracking Proxies");
            Console.WriteLine("Comments for post: {0}", p.Title);
            foreach (var comment in p.Comments)
            {
                Console.WriteLine("\tComment: {0}", comment.CommentText);
            }
        }
    }
}
```

اگر اپلیکیشن کلاینت (برنامه کنسول) را اجرا کنید با خروجی زیر مواجه می‌شوید.

```
Comments for post: Change Tracking Proxies
Comment: I use ICollection<T> all the time
```

شرح مثال جاری

ابتدا با اپلیکیشن کنسول شروع می‌کنیم، که کلاینت سرویس ما است. نخست در یک بلاک {} using وهله ای از کلاینت سرویس مان ایجاد می‌کنیم. درست همانطور که وهله ای از یک EF Context می‌سازیم. استفاده از بلوک‌های using توصیه می‌شود چرا که متد Dispose بصورت خودکار فراخوانی خواهد شد، چه بصورت عادی چه هنگام بروز خطا. پس از آنکه وهله ای از کلاینت سرویس را در اختیار داشتیم، متد Cleanup را صدا می‌زنیم. با فراخوانی این متد تمام داده‌های تست پیشین را حذف می‌کنیم. در چند خط بعدی، متد SubmitPost را روی سرویس فراخوانی می‌کنیم. در پیاده سازی فعلی شناسه پست را بررسی می‌کنیم. اگر مقدار شناسه صفر باشد، خاصیت State موجودیت را به Added تغییر می‌دهید تا رکورد جدیدی ثبت کنیم. در غیر اینصورت فرض بر این است که چنین موجودیتی وجود دارد و قصد ویرایش آن را داریم، بنابراین خاصیت State را به Modified تغییر می‌دهیم. از آنجا که مقدار متغیرهای int بصورت پیش فرض صفر است، با این روش می‌توانیم وضعیت پست‌ها را مشخص کنیم. یعنی تعیین کنیم رکورد جدیدی باید ثبت شود یا رکوردی موجود بروز رسانی گردد. رویکردی بهتر آن است که پارامتری اضافی به متد پاس دهیم، یا متدی مجزا برای ثبت رکوردهای جدید تعریف کنیم. مثلا رکوردی با نام InsertPost. در هر حال، بهترین روش بستگی به ساختار اپلیکیشن شما دارد.

اگر پست جدیدی ثبت شود، خاصیت PostId با مقدار مناسب جدید بروز رسانی می‌شود و وهله پست را باز می‌گردانیم. ایجاد و بروز رسانی نظرات کاربران مشابه ایجاد و بروز رسانی پست‌ها است، اما با یک تفاوت اساسی: بعنوان یک قانون، هنگام بروز رسانی نظرات کاربران تنها فیلد متن نظر باید بروز رسانی شود. بنابراین با فیلدهای دیگری مانند تاریخ انتشار و غیره اصلا کاری نخواهیم داشت. بدین منظور تنها خاصیت CommentText را بعنوان Modified علامت گذاری می‌کنیم. این امر منجر می‌شود که Entity Framework عبارتی برای بروز رسانی تولید کند که تنها این فیلد را در بر می‌گیرد. توجه داشته باشید که این روش تنها در صورتی کار می‌کند که بخواهید یک فیلد واحد را بروز رسانی کنید. اگر می‌خواستیم فیلدهای بیشتری را در موجودیت Comment بروز رسانی کنیم، باید مکانیزمی برای ردیابی تغییرات در سمت کلاینت در نظر می‌گرفتیم. در مواقعی که خاصیت‌های متعددی می‌توانند تغییر کنند، معمولا بهتر است کل موجودیت بروز رسانی شود تا اینکه مکانیزمی پیچیده برای ردیابی تغییرات در سمت کلاینت پیاده گردد. بروز رسانی کل موجودیت بهینه‌تر خواهد بود.

برای حذف یک دیدگاه، متد Entry را روی آبجکت DbContext فراخوانی می‌کنیم و موجودیت مورد نظر را بعنوان آرگومان پاس می‌دهیم. این امر سبب می‌شود که موجودیت مورد نظر بعنوان Deleted علامت گذاری شود، که هنگام فراخوانی متد SaveChanges اسکریپت لازم برای حذف رکورد را تولید خواهد کرد.

در آخر متد GetPostByTitle یک پست را بر اساس عنوان پیدا کرده و تمام نظرات کاربران مربوط به آن را هم بارگذاری می‌کند. از آنجا که ما کلاس‌های POCO را پیاده سازی کرده ایم، Entity Framework آبجکتی را بر می‌گرداند که Dynamic Proxy نامیده می‌شود. این آبجکت پست و نظرات مربوط به آن را در بر خواهد گرفت. متاسفانه WCF نمی‌تواند آبجکت‌های پروکسی را مرتب سازی (serialize) کند. اما با غیرفعال کردن قابلیت ایجاد پروکسی‌ها (ProxyCreationEnabled=false) ما به Entity Framework

می‌گوییم که خود آبجکت‌های اصلی را بازگرداند. اگر سعی کنید آبجکت پروکسی را سریال کنید با پیغام خطای زیر مواجه خواهید شد:

```
The underlying connection was closed: The connection was closed unexpectedly
```

می توانیم غیرفعال کردن تولید پروکسی را به متد سازنده کلاس سرویس منتقل کنیم تا روی تمام متدهای سرویس اعمال شود.

در این قسمت دیدیم چگونه می‌توانیم از آبجکت‌های POCO برای مدیریت عملیات CRUD توسط WCF استفاده کنیم. از آنجا که هیچ اطلاعاتی درباره وضعیت موجودیت‌ها روی کلاینت ذخیره نمی‌شود، متدهایی مجزا برای عملیات CRUD ساختیم. در قسمت‌های بعدی خواهیم دید چگونه می‌توان تعداد متدهایی که سرویس مان باید پیاده سازی کند را کاهش داد و چگونه ارتباطات بین کلاینت و سرور را ساده‌تر کنیم.

نظرات خوانندگان

نویسنده: جلال
تاریخ: ۱۳۹۲/۱۲/۱۰ ۲۰:۲۰

در این سناریو، فرض را بر این گذاشته اید که موجودیتها یا جدید هستند و یا ویرایش شده اند و بنابراین حتی اگر یک پست کامنتهایی داشته باشد که ویرایش نشده اند، برای انها دستور update صادر میشود و این، در مواردیکه تعداد کامنتها(که البته همیشه این موجودیتهای اینگونه به سادگی کامنت نیستند) زیاد باشد، روی کارایی تأثیر منفی خواهد داشت، چه راهی برای تشخیص موجودیتهایی که سمت کلاینت تغییری نکرده اند، پیشنهاد میدهید؟