عنوان: **ساخت یک Mini ORM با AutoMapper**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۲/۱۵ ۱۶:۴۸
آدرس: www.dotnettips.info
گروه‌ها: MVC, Entity framework, AutoMapper

Mini ORMها برخلاف ORMهای کاملی مانند Entity framework یا NHibernate، کوئری‌های LINQ را تبدیل به SQL نمی‌کنند. در اینجا کار با SQL نویسی مستقیم شروع می‌شود و مهم‌ترین کار این کتابخانه‌ها، نگاشت نتیجه‌ی دریافتی از بانک اطلاعاتی به اشیاء دات نتی هستند. خوب ... AutoMapper هم دقیقا همین کار را انجام می‌دهد! بنابراین در ادامه قصد داریم یک Mini ORM را به کمک AutoMapper طراحی کنیم.

کلاس پایه AdoMapper

```
public abstract class AdoMapper<T> where T : class
{
    private readonly SqlConnection _connection;

    protected AdoMapper(string connectionString)
    {
        _connection = new SqlConnection(connectionString);
    }

    protected virtual IEnumerable<T> ExecuteCommand(SqlCommand command)
    {
        command.Connection = _connection;
        command.CommandType = CommandType.StoredProcedure;
        _connection.Open();

        try
        {
            var reader = command.ExecuteReader();
            try
            {
                return Mapper.Map<IDataReader, IEnumerable<T>>(reader);
            }
            finally
            {
                reader.Close();
            }
        }
        finally
        {
            _connection.Close();
        }
    }

    protected virtual T GetRecord(SqlCommand command)
    {
        command.Connection = _connection;
        _connection.Open();
        try
        {
            var reader = command.ExecuteReader();
            try
            {
                reader.Read();
                return Mapper.Map<IDataReader, T>(reader);
            }
            finally
            {
                reader.Close();
            }
        }
        finally
        {
            _connection.Close();
        }
    }

    protected virtual IEnumerable<T> GetRecords(SqlCommand command)
    {
        command.Connection = _connection;
        _connection.Open();
```

```
        try
        {
            var reader = command.ExecuteReader();
            try
            {
                return Mapper.Map<IDataReader, IEnumerable<T>>(reader);
            }
            finally
            {
                reader.Close();
            }
        }
        finally
        {
            _connection.Close();
        }
    }
}
```

در اینجا کلاس پایه Mini ORM طراحی شده را ملاحظه می‌کنید. برای نمونه قسمت GetRecords آن مانند مباحث استاندارد ADO.NET است. فقط کار خواندن و همچنین نگاشت رکوردهای دریافت شده از بانک اطلاعاتی به شیءایی از نوع T توسط AutoMapper انجام خواهد شد.

نحوه‌ی استفاده از کلاس پایه AdoMapper

در کدهای ذیل نحوه‌ی ارث بری از کلاس پایه AdoMapper و سپس استفاده از متدهای آن‌را ملاحظه می‌کنید:

```
public class UsersService : AdoMapper<User>, IUsersService
{
    public UsersService(string connectionString)
        : base(connectionString)
    {
    }

    public IEnumerable<User> GetAll()
    {
        using (var command = new SqlCommand("SELECT * FROM Users"))
        {
            return GetRecords(command);
        }
    }

    public User GetById(int id)
    {
        using (var command = new SqlCommand("SELECT * FROM Users WHERE Id = @id"))
        {
            command.Parameters.Add(new SqlParameter("id", id));
            return GetRecord(command);
        }
    }
}
```

در این مثال نحوه‌ی تعریف کوئری‌های پارامتری نیز در متد GetById به نحو متداولی مشخص شده‌است. کار نگاشت حاصل این کوئری‌ها به اشیاء دات نتی را AutoMapper انجام خواهد داد. نحوه‌ی کار نیز، نگاشت فیلد f1 به خاصیت f1 است (هم نام‌ها به هم نگاشت می‌شوند).

تعریف پروفایل مخصوص AutoMapper

ORMهای تمام عیار، کار نگاشت فیلدهای بانک اطلاعاتی را به خواص اشیاء دات نتی، به صورت خودکار انجام می‌دهند. در اینجا همانند روش‌های متداول کار با AutoMapper نیاز است این نگاشت را به صورت دستی یکبار تعریف کرد:

```
public class UsersProfile : Profile
{
    protected override void Configure()
    {
```

```
        this.CreateMap<IDataRecord, User>();
    }

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

و سپس در ابتدای برنامه آن‌را به AutoMapper معرفی نمود:

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

سفارشی سازی نگاشت‌های AutoMapper

فرض کنید کلاس Advertisement زیر، معادل است با جدول Advertisements بانک اطلاعاتی؛ با این تفاوت که در کلاس تعریف شده، خاصیت TitleWithOtherName تطابقی با هیچکدام از فیلدهای بانک اطلاعاتی ندارد. بنابراین اطلاعاتی نیز به آن نگاشت نخواهد شد.

```
public class Advertisement
{
    public int Id { set; get; }
    public string Title { get; set; }
    public string Description { get; set; }
    public int UserId { get; set; }

    public string TitleWithOtherName { get; set; }
}
```

برای رفع این مشکل می‌توان حین تعریف پروفایل مخصوص Advertisement، آن‌را سفارشی سازی نیز نمود:

```
public class AdvertisementsProfile : Profile
{
    protected override void Configure()
    {
        this.CreateMap<IDataRecord, Advertisement>()
            .ForMember(dest => dest.TitleWithOtherName,
                       options => options.MapFrom(src =>
                            src.GetString(src.GetOrdinal("Title"))));
    }

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

در اینجا پس از تعریف نگاشت مخصوص کار با IDataRecordها، عنوان شده‌است که هر زمانیکه به خاصیت TitleWithOtherName رسیدی، مقدارش را از فیلد Title دریافت و جایگزین کن.

کدهای کامل این مطلب را از اینجا می‌توانید دریافت کنید.

نظرات خوانندگان

نویسنده: امین کاشانی
تاریخ: ۱۳۹۴/۰۲/۱۸ ۲۳:۵۰

باسلام
به نظرتون auto mapper در مقایسه با dapper کدام یک بهتر و کامل‌تر هست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۲/۱۸ ۲۳:۵۳

این قیاس صحیح نیست چون AutoMapper یک Mini ORM نیست؛ اما می‌توان بر اساس آن یک Mini ORM ساخت که نمونه‌ای از آن در اینجا مطرح شده‌است.