

عنوان: ایجاد یک Repository در پروژه برای دستورات EF

نویسنده: میثم ثوامری

تاریخ: ۱۱:۴۷ ۱۳۹۱/۰۵/۱۹

آدرس: www.dotnettips.info

برچسب‌ها: ASP.Net, Design patterns, Entity framework, VB.NET

ایجاد یک Pattern در پروژتون میتونه نظم، سرعت و زیبایی خاصی به کدتون بده. با وجود framework‌های و Pattern‌هایی مسه MVC و MVVM برنامه نویسان را وادار کنه که همه Action‌های یک پروژه رو به سمت کلاینت ببرن. تو یک فرصت دیگه در مورد فریمورک Knockout حتما تایپیک میزارم. امروز میخوام یک Pattern با استفاده از یک Interface و codefirst model براتون بزارم.

گام اول: ایجاد که class property

```
Public Class Employee
    Public Property ID As Integer
    Public Property Fname As String
    Public Property Bdate As DateTime
End Class
```

گام دوم: ایجاد بانک با استفاده از CodeFirst

```
Imports System.Data.Entity
Public Class EmployeeDbContext : Inherits DbContext
    Public Property Employees As DbSet(Of Employee)
End Class
```

گام سوم: ایجاد repository با استفاده از interface

```
Interface EmployeeRepository
    ReadOnly Property All As List(Of Employee)
    Function Find(id As Integer) As Employee
    Sub InsertOrUpdate(p As Employee)
    Sub Delete(id As Integer)
    Sub Save()
End Interface
```

گام چهارم: تعریف کلاس برای implement کردن از iInterface

```
Public Class EmployeeClass : Implements EmployeeRepository
    Private DB As New EmployeeDbContext
    Public ReadOnly Property All As List(Of Employee) Implements EmployeeRepository.All
        Get
            Return DB.Employees.ToList()
        End Get
    End Property

    Public Sub Delete(id As Integer) Implements EmployeeRepository.Delete
        Dim query = DB.Employees.Single(Function(q) q.ID = id)
        DB.Employees.Remove(query)
    End Sub

    Public Function Find(id As Integer) As Employee Implements EmployeeRepository.Find
        Return DB.Employees.Where(Function(q) q.ID = id)
    End Function

    Public Sub InsertOrUpdate(p As Employee) Implements EmployeeRepository.InsertOrUpdate
        If p.ID = Nothing Then
            DB.Employees.Add(p)
        Else
            DB.Entry(p).State = Data.EntityState.Modified
        End If
    End Sub

    Public Sub Save() Implements EmployeeRepository.Save
        DB.SaveChanges()
    End Sub
```

```
End Class
```

برای استفاده تو پروژه براحتی میتونید یک instance از classتون ایجاد کنید و ..

```
Dim cls As New EmployeeClass
```

```
Public Sub BindGrid()  
    GridView1.DataSource = cls.All  
    GridView1.DataBind()  
End Sub
```

موفق باشید

نظرات خوانندگان

نویسنده: علیرضا صالحی
تاریخ: ۱۳۹۱/۰۵/۱۹ ۱۲:۱۳

برای مواردی که خروجی یک لیست (تعدادی آیتم) باشد از Property استفاده نمی‌شود. مثلاً برای All باید از Method استفاده کنید. [Properties vs. Methods](#)
بهتر است برای خروجی متدهایی مانند All نیز به جای لیست از IEnumerable یا IQueryable استفاده کنید.
متدهای Update و Insert نیز به طور جداگانه تعریف شوند. (قرار است هر متد تنها یک وظیفه داشته باشد)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۱۹ ۱۲:۲۱

- در مورد آرایه بحث شده در MSDN. ضمن اینکه استفاده از متد عموماً برای حالتیکه عملیات قابل توجهی در بدنه آن قرار است صورت گیرد، [توصیه می‌شود](#). البته در اینجا چون عملیات دریافت اطلاعات از بانک اطلاعاتی می‌تواند سنگین در نظر گرفته شود، استفاده از متد ارجحیت دارد. خواص نمایانگر اطلاعاتی سبک و با دسترسی سریع هستند.
- خروجی لیست بهتر است. (^) + اگر ReSharper جدید را نصب کنید استفاده از IEnumerable را [نیز توصیه نمی‌کند](#)؛ چون ممکن است چندین بار رفت و برگشت به بانک اطلاعاتی در این بین صورت گیرد.
- مشکلی ندارد. خود EF Code first چنین متدی را دارد. (^) بحث کلاس تک وظیفه‌ای متفاوت است با متدی که نهایتاً قرار است اطلاعات یک رکورد را در بانک اطلاعاتی تغییر دهد (اگر نبود ثبتش کند؛ اگر بود فقط همان رکورد مشخص را به روز رسانی کند).

نویسنده: ناشناس
تاریخ: ۱۳۹۱/۰۵/۱۹ ۱۲:۳۷

با سلام
دلیل استفاده از Interface EmployeeRepository چیه؟
دقیقاً دلیل استفاده Interface اینجا چیه؟
با تشکر از مطلب خوبتون.

نویسنده: ناشناس
تاریخ: ۱۳۹۱/۰۵/۱۹ ۱۲:۴۵

اینجا شاید استفاده از IQueryable بهتر باشه.
شاید کاربر بخواهد قبل از نمایش اطلاعات اونو فیلتر کنه یا اینکه بهتره دو متد Find داشته باشی یکی با خروجی یک آیتم و دیگری با خروجی چندین آیتم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۱۹ ۱۲:۴۸

خیر (^). طراحی یک لایه سرویس که خروجی IQueryable دارد نشی دار در نظر گرفته شده و توصیه نمی‌شود. اصطلاحاً [leaky abstraction](#) هم به آن گفته می‌شود؛ چون طراح نتوانسته حد و مرز سیستم خودش را مشخص کند و همچنین نتوانسته سازوکار درونی آنرا به خوبی کپسوله سازی و مخفی نماید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۱۹ ۱۲:۵۰

به دو دلیل:
- استفاده از امکان تزریق وابستگی‌ها

- امکان نوشتن ساده‌تر آزمون‌های واحد با فراهم شدن زیر ساخت mocking اشیاء

نویسنده: ناشناس
تاریخ: ۱۳۹۱/۰۵/۱۹ ۱۳:۲

در [همین پست](#) خود شما تعداد زیاد رکوردها رو مثال زدید و این پیاده سازی از این موضوع رنج میبره. و در مورد IQueryable قبول دارم و گفتم که بهتر است از دو یا چند متد find استفاده کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۱۹ ۱۳:۲۵

منظور از آن مطلب این بود که از ابزاری که در اختیار دارید درست استفاده کنید. اگر قرار است دو یا چند جستجو را انجام دهید، اینکارها بله باید با IQueryable داخل یک متد انجام شود، اما خروجی متد فقط باید لیست حاصل باشد؛ نه IQueryable ایی که انتهای آن باز است و سبب نشی لایه سرویس شما در لایه‌های دیگر خواهد شد.

نویسنده: میثم ثوامری
تاریخ: ۱۳۹۱/۰۵/۱۹ ۱۹:۵۴

برای برنامه نویسا پیدا کردن یک property راحت‌تر در ضمن از property برای تزریق یا بازیابی اطلاعات از یک object استفاده میکنند.

IQueryable در واقع توسعه یافته IEnumerable. تفاوت عمدشون در LINQ operators که در IQueryable استفاده میشه. اگر هم بخوایم دلیل پیشنهادی داده باشیم اونم اینه که مدیریت حافظه در IQueryable رعایت شده در حالی که Listها کامپایلرو مجاب به اجنام دستور تا انتها میکنند.

نویسنده: میثم ثوامری
تاریخ: ۱۳۹۱/۰۵/۱۹ ۱۹:۵۹

مهندس با نظر دوستمون موافقم

IQueryable بهترین انتخاب برای remote data source که میشه به database یا webservice اشاره کرد. بطور کل اگر شما از ORM مسه linqtosql استفاده میکنید
IQueryable : کوئری شمارو به دستورات sql در database server تبدیل میکنه
IEnumerable: همه رکوردهای شما قبل از اینکه بسمت دیتابیس برن بصورت object در memory نگهداری میشن.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۱۹ ۲۰:۱۱

IQueryable فقط یک expression است. هنوز اجرا نشده. (expose آن از طریق وب سرویس اشتباه است و به مشکلات serialization برخواهید خورد).

زمانیکه ToList, First و امثال آن روی این عبارت فراخوانی شود تبدیل به SQL شده و سپس بر روی بانک اطلاعاتی اجرا می‌شود. به این deferred execution یا اجرای به تعویق افتاده گفته می‌شود.

اگر این عبارت را در اختیار لایه‌های دیگر قرار دهید، یعنی انتهای کار را باز گذاشته‌اید و حد و حدود سیستم شما مشخص نیست. شما اگر IQueryable بازگشت دهید، در لایه‌ای دیگر می‌شود یک join روی آن نوشت و اطلاعات چندین جدول دیگر را استخراج کرد؛ درحالیکه نام متد شما GetUsers بوده. بنابراین بهتر است به صورت صریح اطلاعات را به شکل List بازگشت دهید، تا انتهای کار باز نمانده و طراحی شما نشی نداشته باشد.

نویسنده: محمد عامریان
تاریخ: ۱۳۹۱/۰۸/۱۸ ۱۷:۶

با سلام من یک معماری طراحی کردم به شکل زیر
ابتدا یک اینترفیس به شکل زیر دارم

```
using System;
using System.Collections;
using System.Linq;

namespace Framework.Model
{
    public interface IContext
    {
        T Get<T>(Func<T, bool> prediction) where T : class;
        IEnumerable List<T>(Func<T, bool> prediction) where T : class;
        void Insert<T>(T entity) where T : class;
        int Save();
    }
}
```

بعد یک کلاس ارزش مشتق شده

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Text;

namespace Framework.Model
{
    public class Context : IContext
    {
        private readonly DbContext _dbContext;

        public Context(DbContext context)
        {
            _dbContext = context;
        }

        public T Get<T>(Func<T, bool> prediction) where T : class
        {
            var dbSet = _dbContext.Set<T>();
            if (dbSet != null)
                return dbSet.Single(prediction);

            throw new Exception();
        }

        public void Insert<T>(T entity) where T : class
        {
            var dbSet = _dbContext.Set<T>();
            if (dbSet != null)
            {
                _dbContext.Entry(entity).State = EntityState.Added;
            }
        }

        public int Save()
        {
            return _dbContext.SaveChanges();
        }

        IEnumerable IContext.List<T>(Func<T, bool> prediction)
        {
            var dbSet = _dbContext.Set<T>();
            if (dbSet != null)
                return dbSet.Where(prediction).ToList();

            throw new Exception();
        }
    }
}
```

سپس یک کلاس context دارم که مستقیماً از dbContext مشتق شده

```
using System.Data.Entity;
using DataModel;

namespace Model
{
    public class EFContext : DbContext
    {
        public EFContext(string db): base(db)
        {
        }

        public DbSet<Product> Products { get; set; }
    }
}
```

و سپس کلاس دارم که اوامده پیاده سازی کرده context که خودم ساختمو

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Text;

namespace Model
{
    public class Context : Framework.Model.Context
    {
        public Context(string db): base(new EFContext(db))
        {
        }
    }
}
```

در پروژه دیگری اوامدم یک کلاس context جدید ساختم

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Biz
{
    public class Context : Model.Context
    {
        public Context(string db) : base(db)
        {
        }
    }
}
```

و در کنترلر هم به این شکل ارزش استفاده کردم

```
using System.Web.Mvc;
using Framework.Model;

namespace ProductionRepository.Controllers
{
    public class BaseController : Controller
    {
        public IContext DataContext { get; set; }

        public BaseController()
        {
            DataContext = new
Biz.Context(System.Configuration.ConfigurationManager.ConnectionStrings["Database"].ConnectionString);
        }
    }
}
```

```
using System.Web.Mvc;
using DataModel;
using System.Collections.Generic;

namespace ProductionRepository.Controllers
{
    public class ProductController : BaseController
    {
        public ActionResult Index()
        {
            var x = DataContext.List<Product>(s => s.Name != null);
            return View(x);
        }
    }
}
```

و این هم تست

```
using NUnit.Framework;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Web.Mvc;

namespace TestUnit
{
    [TestFixture]
    public class Test
    {
        [Test]
        public void IndexShouldListProduct()
        {
            var repo = new Moq.Mock<Framework.Model.IContext>();
            var products = new List<DataModel.Product>();
            products.Add(new DataModel.Product { Id = 1, Name = "asdasdasd" });
            products.Add(new DataModel.Product { Id = 2, Name = "adaqwe" });
            products.Add(new DataModel.Product { Id = 4, Name = "qewqw" });
            products.Add(new DataModel.Product { Id = 5, Name = "qwe" });
            repo.Setup(x => x.List<DataModel.Product>(p => p.Name !=
null)).Returns(products.AsEnumerable());
            var controller = new ProductionRepository.Controllers.ProductController();
            controller.DataContext = repo.Object;
            var result = controller.Index() as ViewResult;
            var model = result.Model as List<DataModel.Product>;
            Assert.AreEqual(4, model.Count);
            Assert.AreEqual("", result.ViewName);
        }
    }
}
```

نظرتون چیه آقای نصیری

نویسنده: وحید نصیری
تاریخ: ۱۷:۱۳ ۱۳۹۱/۰۸/۱۸

موارد 1 و 2 عنوان شده در این مطلب رو تکرار کرده: ([^](#))