عنوان: **کوئری هایی با قابلیت استفاده ی مجدد**
نویسنده: شاهین کیاست
تاریخ: ۱۳۹۱/۰۸/۱۸ ۲۰:۳۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, LINQ, ORM

با توجه به اصل Dry تا می‌توان باید از نوشتن کدهای تکراری خودداری کرد و کدها را تا جایی که ممکن است به قسمت هایی با قابلیت استفاده‌ی مجدد تبدیل کرد. حین کار کردن با ORMهای معروف مثل NHibernate و EntityFramework زمان زیادی نوشتن کوئری‌ها جهت واکشی داده‌ها از دیتابیس صرف می‌شود. اگر بتوان کوئری هایی با قابلیت استفاده‌ی مجدد نوشت علاوه بر کاهش زمان توسعه قابلیت هایی قدرتمندی مانند زنجیر کردن کوئری‌ها به دنبال هم به دست می‌آید.
با یک مثال نحوه‌ی نوشتن و مزایای کوئری با قابلیت استفاده‌ی مجدد را بررسی می‌کنیم :
برای مثال دو جدول شهرها و دانش آموزان را درنظر بگیرید:

```
namespace ReUsableQueries.Model
{
    public class Student
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string LastName { get; set; }
        public int Age { get; set; }
    [ForeignKey("BornInCityId")]
        public virtual City BornInCity { get; set; }
        public int BornInCityId { get; set; }
    }

    public class City
    {
        public int Id { get; set; }
        public string Name { get; set; }

        public virtual ICollection<Student> Students { get; set; }
    }
}
```

در ادامه این کلاس‌ها را در معرض دید EF Code first قرار داده:

```
using System.Data.Entity;
using ReUsableQueries.Model;

namespace ReUsableQueries.DAL
{
    public class MyContext : DbContext
    {
        public DbSet<City> Cities { get; set; }
        public DbSet<Student> Students { get; set; }
    }
}
```

و همچنین تعدادی رکورد آغازین را نیز به جداول مرتبط اضافه می‌کنیم:

```
public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }
        protected override void Seed(MyContext context)
        {
            var city1 = new City { Name = "city-1" };
            var city2 = new City { Name = "city-2" };
            context.Cities.Add(city1);
            context.Cities.Add(city2);
            var student1 = new Student() {Name = "Shaahin",LastName = "Kiassat",Age=22,BornInCity =
city1};
            var student2 = new Student() { Name = "Mehdi", LastName = "Farzad", Age = 31, BornInCity =
```

```
city1 };
            var student3 = new Student() { Name = "James", LastName = "Hetfield", Age = 49, BornInCity
= city2 };
            context.Students.Add(student1);
            context.Students.Add(student2);
            context.Students.Add(student3);
            base.Seed(context);
        }
    }
```

فرض کنید قرار است یک کوئری نوشته شود که در جدول دانش آموزان بر اساس نام ، نام خانوادگی و سن جستجو کند :

```
var context = new MyContext();
         var query= context.Students.Where(x => x.Name.Contains(name)).Where(x =>
x.LastName.Contains(lastName)).Where(
              x => x.Age == age);
```

احتمالا هنوز کسانی هستند که فکر می‌کنند کوئری‌های LINQ همان لحظه که تعریف می‌شوند اجرا می‌شوند اما اینگونه نیست . در واقع این کوئری فقط یک Expression از رکوردهای جستجو شده است و تا زمانی که متد ToList یا ToArray روی آن اجرا نشود هیچ داده ای برگردانده نمی‌شود.

در یک برنامه‌ی واقعی داده‌های باید به صورت صفحه بندی شده و مرتب شده برگردانده شود پس کوئری به این صورت خواهد بود :

```
var query= context.Students.Where(x => x.Name.Contains(name)).Where(x =>
x.LastName.Contains(lastName)).Where(
              x => x.Age == age).OrderBy(x=>x.LastName).Skip(skip).Take(take);
```

ممکن است بخواهیم در متد دیگری در لیست دانش آموزان بر اساس نام ، نام خانوادگی ، سن و شهر جستجو کنیم و سپس خروجی را اینبار بر اساس سن مرتب کرده و صفحه بندی نکنیم:

```
var query = context.Students.Where(x => x.Name.Contains(name)).Where(x =>
x.LastName.Contains(lastName)).Where
              (
                  x => x.Age == age).Where(x => x.BornInCityId == 1).OrderBy(x => x.Age);
```

همانطور که می‌بینید قسمت هایی از این کوئری با کوئری هایی که قبلا نوشتیم یکی است ، همچنین حتی ممکن است در قسمت دیگری از برنامه نتیجه‌ی همین کوئری را به صورت صفحه بندی شده لازم داشته باشیم.

اکنون نوشتن این کوئری‌ها میان کد های Business Logic باعث شده هیچ استفاده‌ی مجددی نتوانیم از این کوئری‌ها داشته باشیم. حال بررسی می‌کنیم که چگونه می‌توان کوئری هایی با قابلیت استفاده‌ی مجدد نوشت :

```
namespace ReUsableQueries.Quries
{
    public static class StudentQueryExtension
    {
        public static IQueryable<Student> FindStudentsByName(this IQueryable<Student> students,string
name)
        {
            return students.Where(x => x.Name.Contains(name));
        }
        public static IQueryable<Student> FindStudentsByLastName(this IQueryable<Student> students,
string lastName)
        {
            return students.Where(x => x.LastName.Contains(lastName));
        }
        public static IQueryable<Student> SkipAndTake(this IQueryable<Student> students, int skip , int
take)
        {
            return students.Skip(skip).Take(take);
        }
        public static IQueryable<Student> OrderByAge(this IQueryable<Student> students)
```

```
        {
            return students.OrderBy(x=>x.Age);
        }
    }
}
```

همان طور که مشاهده می‌کنید به کمک متدهای الحاقی برای شیء IQueryable<Student> چند کوئری نوشته ایم . اکنون در محل استفاده از کوئری‌ها می‌توان این کوئری‌ها را به راحتی به هم زنجیر کرد. همچنین اگر روزی قرار شد منطق یکی از کوئری‌ها عوض شود با عوض کردن آن در یک قسمت برنامه همه جا اعمال می‌شود. نحوه‌ی استفاده از این متدهای الحاقی به این صورت خواهد بود :

```
var query =
context.Students.FindStudentsByName(name).FindStudentsByLastName(lastName).SkipAndTake(skip,take);
```

فرض کنید قرار است یک سیستم جستجوی پیشرفته به برنامه اضافه شود که بر اساس شرط‌های مختلف باید یک شرط در کوئری اعمال شود یا نشود ، به کمک این طراحی جدید به راحتی می‌توان بر اساس شرط‌های مختلف یک کوئری را اعمال کرد یا نکرد :

```
var query = context.Students.AsQueryable();
          if (searchByName)
          {
              query= query.FindStudentsByName(name);
          }
          if (orderByAge)
          {
              query = query.OrderByAge();
          }
          if (paging)
          {
             query =  query.SkipAndTake(skip, take);
          }
          return query.ToList();
```

همچنین این کوئری‌ها وابسته به ORM خاصی نیستند البته این نکته هم مد نظر است که LINQ Provider بعضی ORMها ممکن است بعضی کوئری‌ها را پشتیبانی نکند.

نظرات خوانندگان

نویسنده: محمد باقر سیف اللهی
تاریخ: ۱۳۹۱/۰۸/۱۸ ۲۲:۲۳

ممنون از مطلب خوبتون... می‌خواستم بدونم اگه بخوام این متدها رو (در کلاس StudentQueryExtension) جوری بنویسم که با Anonymous Type هم قابل استفاده باشه چه راه حلی وجود داره؟ (یعنی تمام ستون‌ها رو برنگردونم و فقط اونهایی رو که نیاز دارم نمایش بدم و این اعلام نیاز بتونه داینامیک باشه و از طریق پارامتر به تابع پاس داده بشه یا چیزی شبیه این !) . نوع خروجی متدها بهتره چجوری نوشته بشن؟

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۱/۰۸/۱۸ ۲۲:۴۱

خواهش می‌کنم.
با توجه به این که متدهای الحاقی برای

```
IQueryable<Entity>
```

نوشته شده اند پس نوع خروجی هم باید از همین نوع باشد ، راه حلی که به نظرم می‌آید اینه که برای برگداندن چند ستون نوع برگشتی را از نوع یک CustomObject بگذارید مثلا StudentDTO
در مورد داینامیک بودن نمی‌دانم چه کار باید کرد اما برای خودم هم جالب هست که آیا میشه این کار رو کرد یا خیر .

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۱۹ ۱:۲۵

- هیچ تغییری را در متدهای الحاقی همه منظوره ایجاد نکنید. این متدها رکوردی رو بر نمی‌گردونند (در متن لینک داده شده). فقط یک سری عبارت هستند. Select نهایی ویژه را پیش از ToList آخر کار انجام بدید.
- برای پویا کردن LINQ امکان استفاده از رشته‌ها وجود داره: (^)
- نوع خروجی متد در این حالت خاص می‌تونه object یا IEnumerable خالی باشد.

نویسنده: محسن.د
تاریخ: ۱۳۹۱/۰۸/۱۹ ۱:۴۷

اول تشکر می‌کنم بابت مطلب خوبتون ..
اگر سوال جناب سیف اللهی رو درست متوجه شده باشم ، ایشون می‌خوان که فیلدهایی رو که از یک تابع برگشت داده میشه خودشون انتخاب کنن و محدود به مقدار بازگشتی از نوع Student برای مثال نباشن .
ایده ای که به ذهن من رسید (بر اساس برداشتی که از سوال داشتم) استفاده از قابلیت بسیار کاربردی Func هستش . یک Func با ورودی از نوع Entity و مقدار بازگشتی از نوع anonymous Type . در هنگام فراخوانی هم میشه از نوع dynamic برای دریافت نتیجه استفاده کرد . یک نمونه از پیاده سازی همچین چیزی رو اینجا قرار دادم .

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۱/۰۸/۱۹ ۲:۱۸

ممنونم.
نمونه کد خیلی خوبی بود تشکر.

نویسنده: ابراهیم
تاریخ: ۱۳۹۱/۰۸/۱۹ ۱۱:۴۶

سلام. ممنون از مطلب خوبتان. می‌خواستم نظرتان را در رابطهٔ با الگوی Repository بدانم، به نظر من این الگو با اینکه محبوبیت زیادی هم پیدا کرده ولی به پیچیدگی نالازمی نسبت به روش شما دارد. سوالی نیز داشتم، امکان نداشت به شیوه‌ای از IEnumerable به جای IQueryable استفاده شود؟ به نظر من مزیت آن در این است که بتوان خارج از چارچوب ORM از این کوئری‌ها استفاده شود و برای آن‌ها تست ایجاد نمود.
باز هم ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۱۹ ۱۲:۱

- مطلب جاری نفی کننده وجود لایه سرویس در برنامه نیست و مکمل آن است.
- پیاده سازی الگوی مخزنی را که لینک دادید اشتباه است. دلایل اشتباه بودن آن‌را در این مطلب مطالعه کنید: (^)

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۱/۰۸/۱۹ ۱۲:۲

سلام ؛ استفاده از الگوی Repository اضافی در EF Code first؛ آری یا خیر؟!

لطفا مطلب تفاوت‌های IQueryable و IEnumerable را مطالعه بفرمایید.
اگر از IEnumerable استفاده شود دیگر نمی‌توان کوئری‌ها را به هم زنجیر کرد .

نویسنده: محمد باقر سیف اللهی
تاریخ: ۱۳۹۱/۰۸/۲۰ ۹:۴

بسیار ممنون از تمام دوستان...

نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۱/۰۸/۲۰ ۱۷:۹

در قسمت زنجیر کردن کوئری‌ها نباید

```
var query =
context.Students.FindStudentsByName(name).FindStudentsByLastName(lastName).SkipAndTake(skip,take);
```

به

```
var query =
context.Students.AsQueryable().FindStudentsByName(name).FindStudentsByLastName(lastName).SkipAndTake(sk
ip,take);
```

تغییر کند؟! اگر جواب منفی است چرا؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۲۰ ۱۸:۴۷

نیازی نیست چون DbSet از یک سری کلاس منجمله IQueryable مشتق می‌شود.

نویسنده: کیا
تاریخ: ۱۳۹۱/۰۸/۲۱ ۹:۱۲

برای حالتی که بخواین بصورت داینامیک و Anonoymous ستونها رو پاس بدین می‌تونین بصورت زیر عمل کنین. در سمت سرویس :

```
public IEnumerable<dynamic> GetCustomColumnsDynamic(Func<Student, dynamic> pColumns)
        {
            return _entities.Select(pColumns).ToList();
        }
```

و برای استفاده :

```
var resultDynamic = _serviceStudent.GetCustomColumnsDynamic
                (
                    x=> new { x.Id, x.LastName, x.Age  }
                );

            MessageBox.Show(resultDynamic.LastName);
```

و البته همونطور که می‌دونین چون نتیجه بصورت dynamic در اختیار شما قرار می‌گیره از امکانات کامپایلر بی نصیب هستید

نویسنده: محمد جواد تواضعی
تاریخ: ۱۳۹۱/۰۸/۲۹ ۱۷:۳۰

سلام
شاهین جان بابت مطلب بسیار عالی بود.
می خواستم نظرت در مورد اینکه برای گرفتن کوئری با قابلیت مجدد از این روش استفاده بشود چیست ؟ Expression tree

و برای کوئری با قابلیت مجدد کدام روش بهینه‌تر می‌باشد ؟

نویسنده: کوروش شاهی
تاریخ: ۱۳۹۳/۰۲/۲۸ ۱۳:۱۷

با توجه به مطلبی که در مبحث « تفاوت بین IQueryable و IEnumerable در حین کار با ORMs » بیان شده, خروجی متد یا باید List و یا باید IEnumerable باشد ؟
اگه مثالی هم بیان بشه این مهم بیشتر قابل درک است و یا لینکی که با مثال این رو توضیح داده باشه.
متشکر.

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۲/۲۸ ۱۳:۳۴

بستگی داره در چه لایه‌ای کار می‌کنید و این خروجی قراره در چه لایه‌ای استفاده بشه. خروجی لایه سرویس قراره در لایه UI نمایش داده بشه؟ خروجی لایه سرویس نباید IQueryable باشه. داخل لایه سرویس می‌خواهید کوئری‌ها را با هم ترکیب کنید؟ باید IQueryable باشه.

نویسنده: کوروش شاهی
تاریخ: ۱۳۹۳/۰۲/۲۸ ۱۵:۱۳

با توجه به موارد و بستگی هایی که بیان کردین, فقط در لایه سرویس(بیزینس) باید IQueryable بودن یا نبودن خروجی متد رو مشخص کنیم و یا همچنین در لایه Repository یا همون DAL هم باید این موارد رو در نظر بگیریم ؟
با تشکر.

نویسنده: کوروش شاهی
تاریخ: ۱۳۹۳/۰۲/۲۹ ۱۶:۵۷

اگر منبع معتبری هم باشه که این موارد رو در قبال مثال توضیح داده باشه, میتونه خیلی بیشتر مثمر ثمر واقع بشه.
من خیلی گوگل کردم ولی روش‌ها بسیار متنوع بود و آدم سردرگم میشه بیشتر.
متشکرم.

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۳/۰۲/۲۹ ۱۷:۳۰

من یک دور بازخوردهای شما را خواندم اما متوجه موردی که برای شما ابهام ایجاد کرده نشدم.
آیا شما از Entity Framework استفاده می‌کنید؟ اگر پاسخ مثبت است، خود EF لایه‌ی Repository را پیاده سازی کرده است، و این پیاده سازی یک IQueryable جهت انجام Queryهای متفاوت در اختیار شما قرار می‌دهد. شما می‌توانید مستقیما از DbContext سمت لایه‌ی سرویس استفاده کنید و داده‌ها را جهت استفاده برای استفاده کننده‌ی لایه‌ی سرویس فراهم کنید.
لایه‌ی سرویس باید داده‌ها را درون حافظه برگرداند، نه اینکه یک IQueryable برگرداند که استفاده کننده آن را اجرا کند.
از Repository در لایه‌ی سرویس استفاده کنید.