عنوان: **ویدیوی رایگان LINQ Programming with C# 3.0**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۷/۱۲/۱۳ ۱۳:۲۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ

مواردی را که در این ویدیو مشاهده خواهید کرد:

Introduction to LINQ
C# 3.0 Language Features
LINQ to Objects
Lambda Expressions
LINQ to DataSets
Getting Started with LINQ to SQL
Additional LINQ to SQL Features
LINQ to XML
LINQ to Entities and the Entity Framework

دریافت فایل

ماخذ

نظرات خوانندگان

نویسنده: Alex's Blog
تاریخ: ۱۳۸۷/۱۲/۱۸ ۰۹:۳۴:۰۰

طبق معمول مفید بود.

نویسنده: Anonymous
تاریخ: ۱۳۸۸/۰۱/۱۷ ۱۵:۰۱:۰۰

لینک دانلود کار نمیکنه

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۱/۱۷ ۱۷:۳۷:۰۰

http://thewahlingroup.com/CourseDetails.aspx?LINQ200-3

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۱/۱۹ ۲۲:۰۳:۰۰

اگر لینک دانلود برای شما فیلتر است:

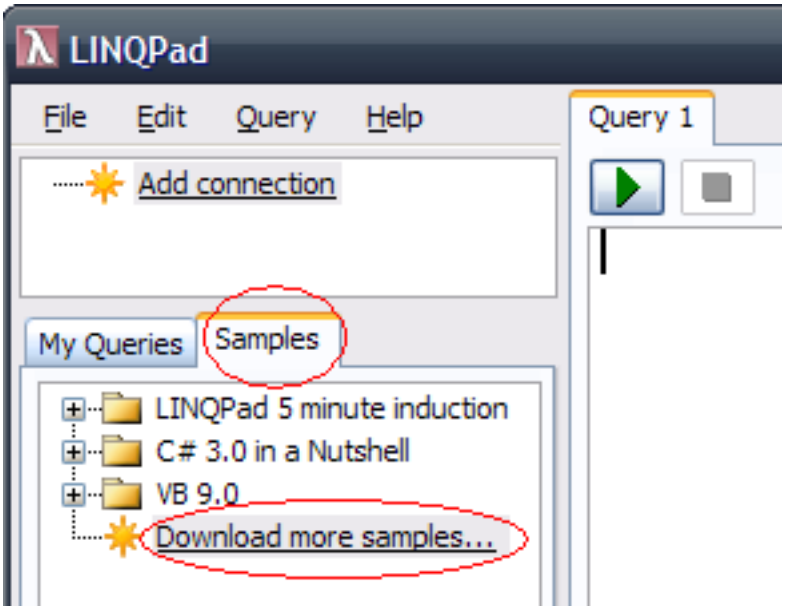
http://rapidshare.com/files/218967618/Dwahlin-GettingStartedWithLINQInNET35290.mp4

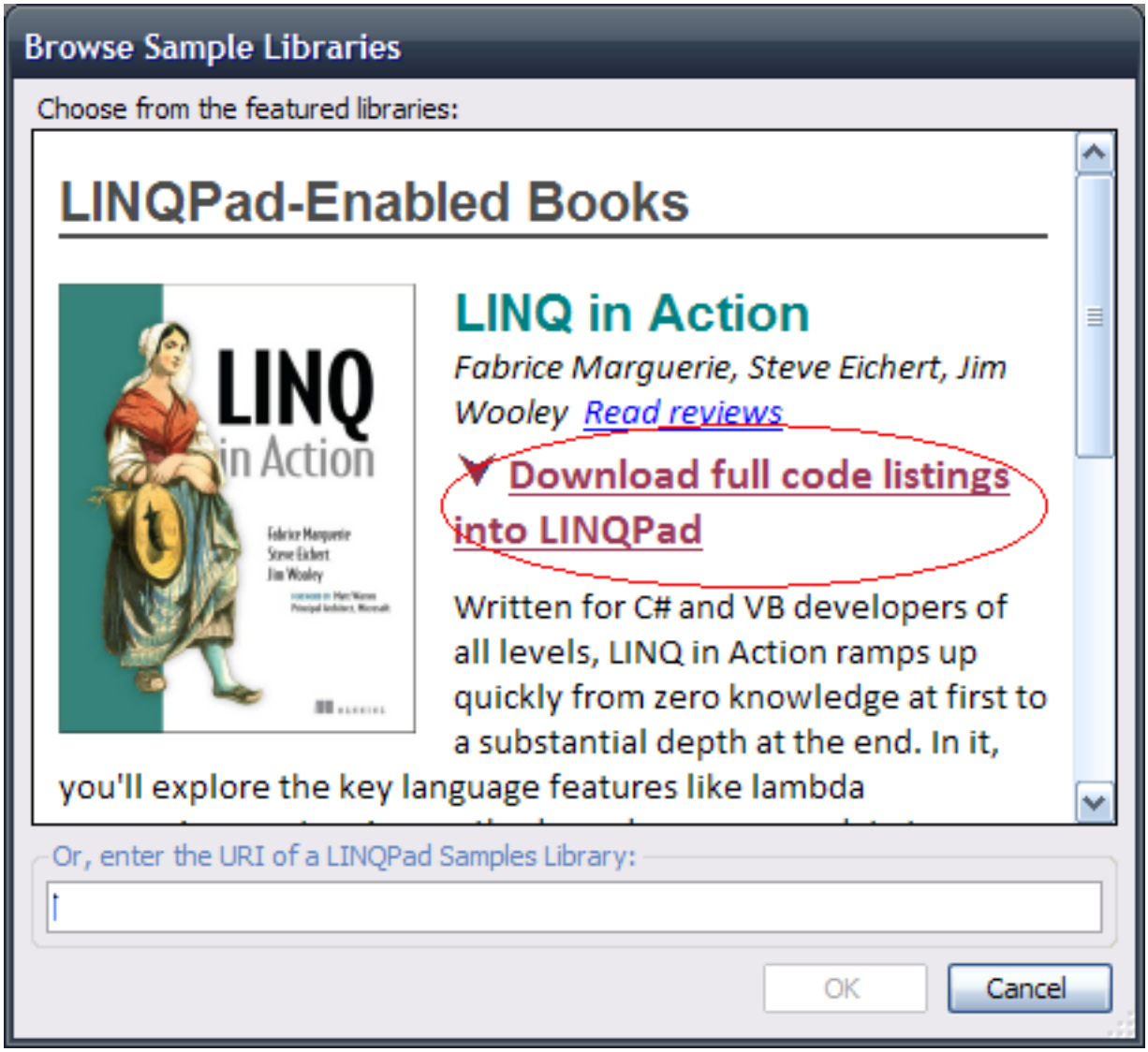
نویسنده: reza
تاریخ: ۱۳۸۸/۰۱/۱۹ ۲۳:۲۲:۰۰

دست گلتون درد نکنه خیلی ممنون

عنوان: **برنامه LINQPad و مثال‌های جدید آن**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۳/۱۷ ۲۱:۴۳:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ

برنامه معروف LINQPad تا کنون به همراه مثال‌های کتاب C# 3.0 in a Nutshell به صورت یکپارچه ارائه می‌شد.
اکنون مثال‌های کتاب LINQ in Action نیز قابلیت یکپارچگی با این برنامه را یافته‌اند. به این صورت بسیار ساده و در همان محیط LINQPad می‌توان این مثال‌ها را مرور و اجرا کرد که در یادگیری LINQ کمک شایانی می‌نمایند.
برای نصب این مثال‌های یکپارچه جدید، بر روی لینک Download more samples آن کلیک کرده و در صفحه‌ی باز شده، بر روی لینکی به نام Download full code listings into LINQPad کلیک کنید.





اکنون مثال‌های سی شارپ و VB.Net آن به صورت یکپارچه در اختیار شما خواهند بود.

عنوان: **خلاصه‌ای از LINQ to XML**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۵/۲۱ ۱۲:۵۷:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ

در این مقاله مروری سریع و کاربردی خواهیم داشت بر توانایی‌های مقدماتی LINQ to XML .

فایل Employee.XML را با محتویات زیر در نظر بگیرید:

```
<Employees>
     <Employee>
         <Name>Vahid</Name>
         <Phone>11111111</Phone>
         <Department>IT</Department>
  <Age>52</Age>
     </Employee>
     <Employee>
         <Name>Farid</Name>
         <Phone>124578963</Phone>
         <Department>Civil</Department>
        <Age>35</Age>
     </Employee>
     <Employee>
         <Name>Mehdi</Name>
         <Phone>1245788754</Phone>
         <Department>HR</Department>
        <Age>30</Age>
     </Employee>
 </Employees>
```

1 - چگونه یک فایل XML را جهت استفاده توسط LINQ بارگذاری کنیم؟

قبل از شروع، اسمبلی System.Xml.Linq باید به ارجاعات برنامه اضافه شود. سپس:

```
using System.Xml.Linq;

XDocument xDoc = XDocument.Load("Employee.xml");
```

2 - اگر محتویات XML دریافتی به صورت رشته بود (مثلا از یک دیتابیس دریافت شد)، اکنون چگونه باید آن‌را بارگذاری کرد؟

این‌کار را با استفاده از یک StringReader به صورت زیر می‌توان انجام داد:

```
// loading XML from string
StringReader sr = new StringReader(stringXML);
XDocument xDoc = XDocument.Load(sr);
```

3- چگونه یک کوئری ساده شامل تمامی رکوردهای Employee مجموعه Employees را تهیه کنیم؟

```
using System.Collections;

IEnumerable<XElement> empList = from e in xDoc.Root.Elements("Employee") select e;
```

توسط کوئری فوق، تمامی رکوردهای کارکنان در یک Collection در اختیار ما خواهند بود. نکته‌ی مهم عبارت LINQ فوق،

xDoc.Root.Elements("Employee") می‌باشد. به این صورت از xDoc بارگذاری شده، ابتدا Root و یا همان محتوای فایل XML را جهت بررسی انتخاب کرده و سپس گره‌های مرتبط با کارکنان را انتخاب می‌کنیم.
اکنون که مجموعه کارکنان توسط متغیر empList در اختیار ما است، دسترسی به محتویات آن به سادگی زیر خواهد بود:

```
foreach (XElement employee in empList)
{
    foreach (XElement e in employee.Elements())
    {
         Console.WriteLine(e.Name + " = " + e.Value);
    }
}
```

در این‌جا حلقه خارجی اطلاعات کلی تمامی کارکنان را باز می‌گرداند و حلقه داخلی اطلاعات یک گره دریافت شده را نمایش می‌دهد.

4 - کوئری بنویسید که اطلاعات تمامی کارکنان بخش HR را باز گرداند.

```
IEnumerable<XElement> hrList = from e in xDoc.Root.Elements("Employee")
                                                  where e.Element("Department").Value == "HR"
                                                  select e;
```

همانطور که ملاحظه می‌کنید همانند عبارات SQL ، در تمامی عناصر متعلق به کارکنان، عناصری که دپارتمان آن‌ها مساوی HR است بازگشت داده می‌شود.

-5 کوئری بنویسید که لیست تمامی کارکنان بالای 30 سال را ارائه دهد.

```
IEnumerable<XElement> tList = from e in xDoc.Root.Elements("Employee")
                                                   where int.Parse(e.Element("Age").Value) > 30
                                                   select e;
```

چون حاصل e.Element("Age").Value یک رشته است، برای اعمال فیلترهای عددی باید این رشته‌ها تبدیل به عدد شوند. به همین جهت از int.Parse استفاده شده است.

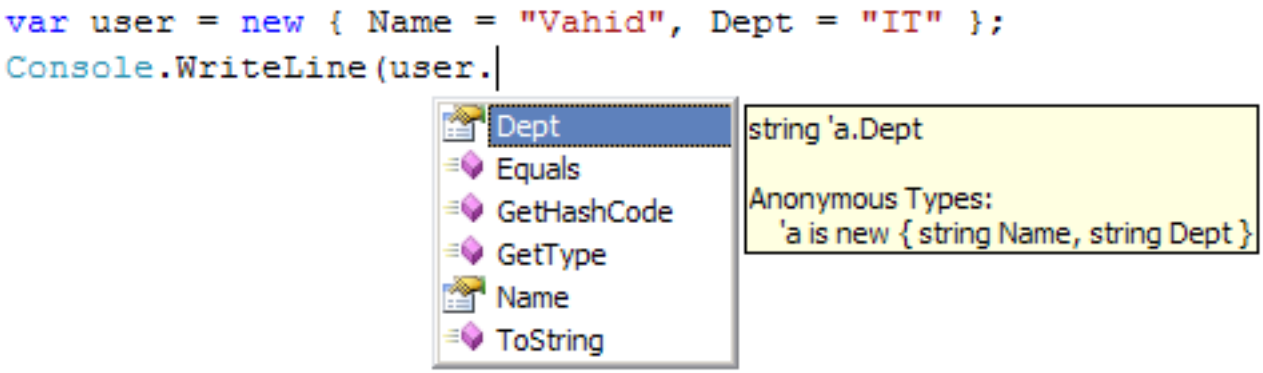
6 - کوئری بنویسید که لیست تمامی کارکنان بالای 30 سال را مرتب شده بر اساس نام باز گرداند.

```
IEnumerable<XElement> tList = from e in xDoc.Root.Elements("Employee")
                                                  where int.Parse(e.Element("Age").Value) > 30
                                                  orderby e.Element("Name").Value
                                                  select e;
```

در اینجا همانند عبارات SQL از orderby جهت مرتب سازی بر اساس عناصر نام استفاده شده است.

7 - تبدیل نتیجه‌ی یک کوئری LINQ به لیستی از اشیاء

مفهومی به سی شارپ 3 اضافه شده است به نام anonymous types . برای مثال:



توسط این قابلیت می‌توان یک شیء را بدون نیاز به تعریف ابتدایی آن ایجاد کرد و حتی از intelliSense موجود در IDE نیز بهره مند شد. این نوع‌های ناشناس توسط واژه‌های کلیدی new و var تولید می‌شوند. کامپایلر به صورت خودکار برای هر anonymous type یک کلاس ایجاد می‌کند.

دقیقا از همین توانایی در LINQ نیز می‌توان استفاده نمود:

```
var empList = from e in xDoc.Root.Elements("Employee")
                          orderby e.Element("Name").Value
                          select new
                          {
                              Name = e.Element("Name").Value,
                              Phone = e.Element("Phone").Value,
                              Department = e.Element("Department").Value,
                              Age = int.Parse(e.Element("Age").Value)
                          };
```

در این‌جا حاصل کوئری، تبدیل به لیستی از اشیاءanonymous می‌شود. اکنون برای نمایش آن‌ها نیز می‌توان از واژه کلیدی var استفاده نمود که از هر لحاظ نسبت به روش اعمال foreach بر روی Xelement ها که در مثال 3 مشاهده کردیم خواناتر است:

```
foreach (var employee in empList)
            {
                Console.WriteLine("Name = " + employee.Name);
                Console.WriteLine("Dep = " + employee.Department);
                Console.WriteLine("Phone = " + employee.Phone);
                Console.WriteLine("Age = " + employee.Age);
            }
```

و البته بدیهی است که می‌توان از anonymous types استفاده نکرد و دقیقا تعریف شیء را پیش از انتخاب آن نیز مشخص نمود. برای مثال:

```
public class Employee
   {
        public string Name { get; set; }
        public string Phone { get; set; }
        public string Department { get; set; }
        public int Age { get; set; }
  }
```

در این حالت، قسمت select new عبارت LINQ ما به select new Employee تغییر خواهد کرد.

برای مثال اگر بخواهیم لیست دریافتی را به صورت یک لیست جنریک بازگشت دهیم خواهیم داشت:

```
public class Employee
        {
            public string Name { get; set; }
            public string Phone { get; set; }
```

```
        public string Department { get; set; }
        public int Age { get; set; }
    }

    List<Employee> Get()
    {
        XDocument xDoc = XDocument.Load("Employee.xml");
        var items =
            from e in xDoc.Root.Elements("Employee")
            orderby e.Element("Name").Value
            select new Employee
            {
                Name = e.Element("Name").Value,
                Phone = e.Element("Phone").Value,
                Department = e.Element("Department").Value,
                Age = int.Parse(e.Element("Age").Value)
            };
        return items.ToList();
    }
```

نظرات خوانندگان

نویسنده: Anonymous
تاریخ: ۱۳۸۸/۰۵/۲۱ ۱۳:۰۸:۰۷

برای تکمیل بحث با اجازه آقای نصیری این لینک هم من اضافه می کنم که قابلیت های خارق العاده LINQ رو برای کار با XML نشون می ده :

http://windowsclient.net/learn/video.aspx?v=6895

موفق باشید.

نویسنده: علی اقدم
تاریخ: ۱۳۸۹/۰۷/۰۴ ۱۲:۲۷:۳۵

بسیار مفید بود.ممنون.

عنوان: **استفاده از LINQ to XML جهت خواندن فیدهای RSS**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۵/۲۳ ۲۰:۰۵:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ

مثال زیر را به عنوانی نمونه‌ای از کاربرد LINQ to XML برای خواندن فیدهای RSS که اساسا به فرمت XML هستند می‌توان ارائه داد.
ابتدا کد کامل مثال را در نظر بگیرید:

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

namespace LinqToRSS
{
 public static class LanguageExtender
 {
     public static string SafeValue(this XElement input)
     {
         return (input == null) ? string.Empty : input.Value;
     }

     public static DateTime SafeDateValue(this XElement input)
     {
         return (input == null) ? DateTime.MinValue : DateTime.Parse(input.Value);
     }
 }

 public class RssEntry
 {
     public string Title { set; get; }
     public string Description { set; get; }
     public string Link { set; get; }
     public DateTime PublicationDate { set; get; }
     public string Author { set; get; }
     public string BlogName { set; get; }
     public string BlogAddress { set; get; }
 }

 public class Rss
 {
     static XElement selectDate(XElement date1, XElement date2)
     {
         return date1 ?? date2;
     }

     public static List<RssEntry> GetEntries(string feedUrl)
     {
         //applying namespace in an XElement
         XName xn = XName.Get("{http://purl.org/dc/elements/1.1/}creator");//{namespace}root
         XName xn2 = XName.Get("{http://purl.org/dc/elements/1.1/}date");

         var feed = XDocument.Load(feedUrl);
         if (feed.Root == null) return null;

         var items = feed.Root.Element("channel").Elements("item");
         var feedQuery =
             from item in items
             select new RssEntry
                        {
                            Title = item.Element("title").SafeValue(),
                            Description = item.Element("description").SafeValue(),
                            Link = item.Element("link").SafeValue(),
                            PublicationDate =
                                selectDate(item.Element(xn2), item.Element("pubDate")).SafeDateValue(),
                            Author = item.Element(xn).SafeValue(),
                            BlogName = item.Parent.Element("title").SafeValue(),
                            BlogAddress = item.Parent.Element("link").SafeValue()
                        };

         return feedQuery.ToList();
     }
 }
```

```
 class Program
 {
     static void Main(string[] args)
     {
         List<RssEntry> entries = Rss.GetEntries("http://weblogs.asp.net/aspnet-team/rss.aspx");
         if (entries != null)
             foreach (var item in entries)
                 Console.WriteLine(item.Title);

         Console.WriteLine("Press a key...");
         Console.ReadKey();
     }
 }
}
```

توضیحات:

-1 در این مثال فقط جهت سهولت بیان آن در یک صفحه، تمامی کلاس‌های تعریف شده در یک فایل آورده شدند. این روش صحیح نیست و باید به ازای هر کلاس یک فایل جدا در نظر گرفته شود.

-2 کلاس LanguageExtender از قابلیت extension methods سی شارپ 3 استفاده می‌کند. به این صورت کلاس XElement دات نت بسط یافته و دو متد به آن اضافه می‌شود که به سادگی در کدهای خود می‌توان از آن‌ها استفاده کرد. هدف آن هم بررسی نال بودن یک آیتم دریافتی و ارائه‌ی حاصلی امن برای این مورد است.

-3 کلاس RssEntry به جهت استفاده در خروجی کوئری LINQ تعریف شد. می‌خواهیم خروجی نهایی، یک لیست جنریک از نوع RssEntry باشد.

-4 متد اصلی برنامه، GetEntries است. این متد آدرس اینترنتی یک فید را دریافت کرده و پس از آنالیز، آن‌را به صورت یک لیست بر می‌گرداند.

```
<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet type="text/xsl" href="http://weblogs.asp.net/utility/FeedStylesheets/rss.xsl"
media="screen"?>
<rss version="2.0" xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:slash="http://purl.org/rss/1.0/modules/slash/" xmlns:wfw="http://wellformedweb.org/CommentAPI/">
<channel>
 <title>Latest Microsoft Blogs</title>
 <link>http://weblogs.asp.net/aspnet-team/default.aspx</link>
 <description />
 <dc:language>en</dc:language>
 <generator>CommunityServer 2007 SP1 (Build: 20510.895)</generator>
 <item>
   <title>Comments on my recent benchmarks.</title>
   <link>http://misfitgeek.com/blog/aspnet/comments-on-my-recent-benchmarks/</link>
   <pubDate>Mon, 10 Aug 2009 23:33:59 GMT</pubDate>
   <guid isPermaLink="false">c06e2b9d-981a-45b4-a55f-ab0d8bbfdc1c:7166225</guid>
   <dc:creator>Misfit Geek: msft</dc:creator>
   <slash:comments>0</slash:comments>
   <wfw:commentRss xmlns:wfw="http://wellformedweb.org/CommentAPI/">http://weblogs.asp.net/aspnet-
team/rsscomments.aspx?PostID=7166225</wfw:commentRss>
   <comments>http://misfitgeek.com/blog/aspnet/comments-on-my-recent-benchmarks/#comments</comments>
   <description>Overall I’ve been pretty impressed ...</description>
   <category domain="http://weblogs.asp.net/aspnet-
team/archive/tags/ASP.NET/default.aspx">ASP.NET</category>
 </item>
</channel>
</rss>
```

برای نمونه خروجی یک فید می‌تواند به صورت فوق باشد. آیتم‌های آن به صورت قابل بیان است:

```
var items = feed.Root.Element("channel").Elements("item");
```

و نکته مهمی که اینجا وجود دارد، اعمال فضاهای نام بکار رفته در این فایل xml پیشرفته می‌باشد. برای اعمال فضاهای نام به یکی از دو روش زیر می‌توان عمل کرد:

```
XName.Get("{mynamespace}root");
```

```
//or
XName.Get("root", "mynamespace");
```

عنوان: **استفاده از LINQ جهت جستجوی فایل‌ها**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۵/۳۰ ۲۱:۲۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ

یکی دیگر از کاربردهای anonymous types ، امکان استفاده از قابلیت‌های LINQ برای جستجوی فایل‌ها و پوشه‌ها است.
مثال:

```
using System;
using System.Linq;
using System.IO;

namespace LINQtoDir
{
   class Program
   {
       static void Main(string[] args)
       {
           var query = from f in new DirectoryInfo(@"C:\Documents and Settings\vahid\My Documents\My
Pictures")
                                .GetFiles("*.*", SearchOption.AllDirectories)
                       where f.Extension.ToLower() == ".png" || f.Extension.ToLower() == ".jpg"
                       orderby f.LastAccessTime
                       select new
                       {
                           DateLastModified = f.LastWriteTime,
                           Extension = f.Extension,
                           Size = f.Length,
                           FileName = f.Name
                       };

           foreach (var file in query)
               Console.WriteLine(file.FileName);

           Console.WriteLine("Press a key...");
           Console.ReadKey();
       }
   }
}
```

در این مثال توسط کوئری نوشته شده، تمامی تصاویر jpg و یا png موجود در پوشه my pictures یافت شده و سپس بر اساس LastAccessTime مرتب می‌شوند. در آخر با استفاده از anonymous types ، یک شیء IEnumerable از خواص مورد نظر فایل‌های یافت شده، بازگشت داده می‌شود. اکنون هر استفاده‌ی دلخواهی را می‌توان از این شیء انجام داد.

عنوان: **استثنای Sequence contains no elements در حین استفاده از LINQ**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۶/۰۸ ۲۱:۲۵:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ

در ابتدا مثال‌های زیر را در نظر بگیرید:

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

namespace testWinForms87
{
  public class Data
  {
      public int id { get; set; }
      public string name { get; set; }
  }

  class CLinqTests
  {
      public static int TestGetListMin1()
      {
          var lst = new List<Data>
          {
                  new Data{ id=1, name="id1"},
                  new Data{ id=2, name="id2"},
                  new Data{ id=3, name="name3"}
          };

          return (from c in lst
                  where c.name.Contains("id")
                  select c.id).Min();
      }

      public static int TestGetListMin2()
      {
          var lst = new List<Data>();

          return (from c in lst
                  where c.name.Contains("id")
                  select c.id).Min();
      }
  }
}
```

در متد TestGetListMin1 قصد داریم کوچکترین آی دی رکوردهایی را که نام آن‌ها حاوی id است، از لیست تشکیل شده از کلاس Data بدست آوریم (همانطور که مشخص است سه رکورد از نوع Data در لیست lst ما قرار گرفته‌اند).
محاسبات آن کار می‌کند و مشکلی هم ندارد. اما همیشه در دنیای واقعی همه چیز قرار نیست به این خوبی پیش برود. ممکن است همانند متد TestGetListMin2 ، لیست ما خالی باشد (برای مثال از دیتابیس، رکوردی مطابق شرایط کوئری‌های قبلی بازگشت داده نشده باشد). در این حالت هنگام فراخوانی متد Min ، استثنای Sequence contains no elements رخ خواهد داد و همانطور که در مباحث defensive programming عنوان شد، وظیفه‌ی ما این نیست که خودرو را به دیوار کوبیده (یا منتظر شویم تا کوبیده شود) و سپس به فکر چاره بیفتیم که خوب، عجب! مشکلی رخ داده است!
اکنون چه باید کرد؟ حداقل یک مرحله بررسی اینکه آیا کوئری ما حاوی رکوردی می‌باشد یا خیر باید به این متد اضافه شود (به صورت زیر):

```
public static int TestGetListMin3()
      {
          var lst = new List<Data>();
          var query = from c in lst
                      where c.name.Contains("id")
                      select c.id;

          if (query.Any())
              return query.Min();
```

```
            else
                return -1;
        }
```

البته می‌شد اگر هیچ رکوردی بازگشت داده نمی‌شد، یک استثنای سفارشی را ایجاد کرد، اما به شخصه ترجیح می‌دهم عدد منهای یک را بر گردانم (چون می‌دانم رکوردهای من عدد مثبت هستند و اگر حاصل منفی شد نیازی به ادامه‌ی پروسه نیست).

شبیه به این مورد در هنگام استفاده از تابع Single مربوط به LINQ نیز ممکن است رخ دهد (تولید استثنای ذکر شده) اما در اینجا مایکروسافت تابع SingleOrDefault را نیز پیش بینی کرده است. در این حالت اگر کوئری ما رکوردی را برنگرداند، SingleOrDefault مقدار نال را برگشت داده و استثنایی رخ نخواهد داد (نمونه‌ی دیگر آن متدهای First و FirstOrDefault هستند).
در مورد متدهای Min و Max ، متدهای MinOrDefault یا MaxOrDefault در دات نت فریم ورک وجود ندارند. می‌توان این نقیصه را با استفاده از extension methods برطرف کرد.

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

  public static class LinqExtensions
  {
      public static T MinOrDefault<T>(this IEnumerable<T> source, T defaultValue)
      {
          if (source.Any<T>())
              return source.Min<T>();

          return defaultValue;
      }

      public static T MaxOrDefault<T>(this IEnumerable<T> source, T defaultValue)
      {
          if (source.Any<T>())
              return source.Max<T>();

          return defaultValue;
      }
  }
```

اکنون با استفاده از extension methods فوق، کد ما به صورت زیر تغییر خواهد کرد:

```
public static int TestGetListMin4()
      {
          var lst = new List<Data>();
          return (from c in lst
                  where c.name.Contains("id")
                  select c.id).MinOrDefault(-1);
      }
```

عنوان: **استفاده از LINQ جهت تهیه کدهایی کوتاه‌تر و خواناتر**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۸/۰۴ ۲۰:۱۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ

با کمک امکانات ارائه شده توسط LINQ ، می‌توان بسیاری از اعمال برنامه نویسی را در حجمی کمتر، خواناتر و در نتیجه با قابلیت نگهداری بهتر، انجام داد که تعدادی از آن‌ها را در ادامه مرور خواهیم کرد.

الف) تهیه یک یک رشته، حاوی عناصر یک آرایه، جدا شده با کاما.

```
using System.Linq;

public class CLinq
{
 public static string GetCommaSeparatedListNormal(string[] data)
 {
     string items = string.Empty;

     foreach (var item in data)
     {
         items += item + ", ";
     }

     return items.Remove(items.Length - 2, 1).Trim();
 }

 public static string GetCommaSeparatedList(string[] data)
 {
     return data.Aggregate((s1, s2) => s1 + ", " + s2);
 }
}
```

همانطور که ملاحظه می‌کنید در روش دوم با استفاده از LINQ Aggregate extension method ، کد جمع و جورتر و خواناتری نسبت به روش اول حاصل شده است.

ب) پیدا کردن تعداد عناصر یک آرایه حاوی مقداری مشخص
برای مثال آرایه زیر را در نظر بگیرید:

```
var names = new[] { "name1", "name2", "name3", "name4", "name5", "name6", "name7" };
```

قصد داریم تعداد عناصر حاوی name را مشخص سازیم.
در تابع GetCountNormal زیر، این کار به شکلی متداول انجام شده و در GetCount از LINQ Count extension method کمک گرفته شده است.

```
using System.Linq;

public class CLinq
{
 public static int GetCountNormal()
 {
     var names = new[] { "name1", "name2", "name3", "name4", "name5", "name6", "name7" };
     var count = 0;
     foreach (var name in names)
     {
         if (name.Contains("name"))
             count += 1;
     }
     return count;
 }

 public static int GetCount()
```

```
 {
     var names = new[] { "name1", "name2", "name3", "name4", "name5", "name6", "name7" };
     return names.Count(name => name.Contains("name"));
 }
}
```

به نظر شما کدام روش خواناتر بوده و نگهداری و یا تغییر آن در آینده ساده‌تر می‌باشد؟

ج) دریافت لیستی از عناصر شروع شده با یک عبارت
در اینجا نیز دو روش متداول و استفاده از LINQ بررسی شده است.

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

public class CLinq
{
 public static List<string> GetListNormal()
 {
     List<string> sampleList = new List<string>() { "A1", "A2", "P1", "P10", "B1", "B@", "J30", "P12"
};
     List<string> result = new List<string>();
     foreach (var item in sampleList)
     {
         if (item.StartsWith("P"))
             result.Add(item);
     }
     return result;
 }

 public static List<string> GetList()
 {
     List<string> sampleList = new List<string>() { "A1", "A2", "P1", "P10", "B1", "B@", "J30", "P12"
};
     return sampleList.Where(x => x.StartsWith("P")).ToList();
 }
}
```

و در حالت کلی، اکثر حلقه‌های foreach متداول را می‌توان با نمونه‌های خواناتر کوئری‌های LINQ معادل، جایگزین کرد.

نظرات خوانندگان

نویسنده: افشار محبی
تاریخ: ۱۳۸۸/۰۸/۰۵ ۰۹:۱۰:۳۵

آیا LINQ در زبان‌های دیگر مثل جاوا معادلی دارد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۸/۰۵ ۱۱:۵۳:۲۳

سلام
تلاش‌هایی به صورت مستقل برای جاوا هم شده (یکپارچه با زبان نیست)
http://xircles.codehaus.org/projects/quaere
اما باز هم پیاده سازی آن در بسیاری از موارد type safety دات نت را ندارد و از رشته‌ها کمک گرفته.
در کل جاوا به دلیل نداشتن معادلی برای lambda expressions که پایه و اساس LINQ را تشکیل می‌دهد، هنوز در این زمینه کار پایه‌ای را انجام نداده است و در کل قسمت LI مربوط به LINQ را ندارد (language integrated)

نویسنده: Majid
تاریخ: ۱۳۸۸/۱۱/۲۲ ۲۱:۳۸:۰۹

ضمن تشکر از مطالب خوبتان.
بد نیست نگاهی به این add-in بیاندازید، برای من که جالب بود
http://code.msdn.microsoft.com/vlinq

عنوان: **پیدا کردن آیتم‌های تکراری در یک لیست به کمک LINQ**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۸/۰۶ ۲۱:۴۴:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ

گاهی از اوقات نیاز می‌شود تا در یک لیست، آیتم‌های تکراری موجود را مشخص کرد. به صورت پیش فرض متد Distinct برای حذف مقادیر تکراری در یک لیست با استفاده از LINQ موجود است که البته آن‌هم اما و اگرهایی دارد که در ادامه به آن پرداخته خواهد شد، اما باز هم این مورد پاسخ سؤال اصلی نیست (نمی‌خواهیم موارد تکراری را حذف کنیم).

برای حذف آیتم‌های تکراری از یک لیست جنریک می‌توان متد زیر را نوشت:

```
public static List<T> RemoveDuplicates<T>(List<T> items)
{
   return (from s in items  select s).Distinct().ToList();
}
```

برای مثال:

```
public static void TestRemoveDuplicates()
{
   List<string> sampleList =
       new List<string>() { "A1", "A2", "A3", "A1", "A2", "A3" };
   sampleList = RemoveDuplicates(sampleList);
   foreach (var item in sampleList)
       Console.WriteLine(item);
}
```

این متد بر روی لیست‌هایی با نوع‌های اولیه مانند string و int و امثال آن درست کار می‌کند. اما اکنون مثال زیر را در نظر بگیرید:

```
public class Employee
{
   public int ID { get; set; }
   public string FName { get; set; }
   public int Age { get; set; }
}

public static void TestRemoveDuplicates()
{
   List<Employee> lstEmp = new List<Employee>()
   {
      new  Employee(){ ID=1, Age=20, FName="F1"},
      new  Employee(){ ID=2, Age=21, FName="F2"},
      new  Employee(){ ID=1, Age=20, FName="F1"},
   };

   lstEmp = RemoveDuplicates<Employee>(lstEmp);

   foreach (var item in lstEmp)
       Console.WriteLine(item.FName);
}
```

اگر متد TestRemoveDuplicates را اجرا نمائید، رکورد تکراری این لیست جنریک حذف نخواهد شد؛ زیرا متد distinct بکارگرفته شده نمی‌داند اشیایی از نوع کلاس سفارشی Employee را چگونه باید با هم مقایسه نماید تا بتواند موارد تکراری آن‌ها را حذف کند.
برای رفع این مشکل باید از آرگومان دوم متد distinct جهت معرفی وهله‌ای از کلاسی که اینترفیس IEqualityComparer را پیاده سازی می‌کند، کمک گرفت.

```
public static IEnumerable<TSource> Distinct<TSource>(this IEnumerable<TSource> source,
IEqualityComparer<TSource> comparer);
```

که نمونه‌ای از پیاده سازی آن به شرح زیر می‌تواند باشد:

```
public class EmployeeComparer : IEqualityComparer<Employee>
{
 public bool Equals(Employee x, Employee y)
 {
   //آیا دقیقا یک وهله هستند؟
   if (Object.ReferenceEquals(x, y)) return true;

   //آیا یکی از وهله‌ها نال است؟
   if (Object.ReferenceEquals(x, null) ||
       Object.ReferenceEquals(y, null))
       return false;

   return x.Age == y.Age && x.FName == y.FName && x.ID == y.ID;
 }

 public int GetHashCode(Employee obj)
 {
   if (Object.ReferenceEquals(obj, null)) return 0;
   int hashTextual = obj.FName == null ? 0 : obj.FName.GetHashCode();
   int hashDigital = obj.Age.GetHashCode();
   return hashTextual ^ hashDigital;
 }
}
```

اکنون اگر یک overload برای متد RemoveDuplicates با درنظر گرفتن IEqualityComparerتهیه کنیم، به شکل زیر خواهد بود:

```
public static List<T> RemoveDuplicates<T>(List<T> items, IEqualityComparer<T> comparer)
{
   return (from s in items select s).Distinct(comparer).ToList();
}
```

به این صورت متد آزمایشی ما به شکل زیر (که وهله‌ای از کلاس EmployeeComparer به آن ارسال شده) تغییر خواهد کرد:

```
public static void TestRemoveDuplicates()
{
   List<Employee> lstEmp = new List<Employee>()
   {
      new  Employee(){ ID=1, Age=20, FName="F1"},
      new  Employee(){ ID=2, Age=21, FName="F2"},
      new  Employee(){ ID=1, Age=20, FName="F1"},
   };

   lstEmp = RemoveDuplicates(lstEmp, new EmployeeComparer());

   foreach (var item in lstEmp)
       Console.WriteLine(item.FName);
}
```

پس از این تغییر، حاصل این متد تنها دو رکورد غیرتکراری می‌باشد.

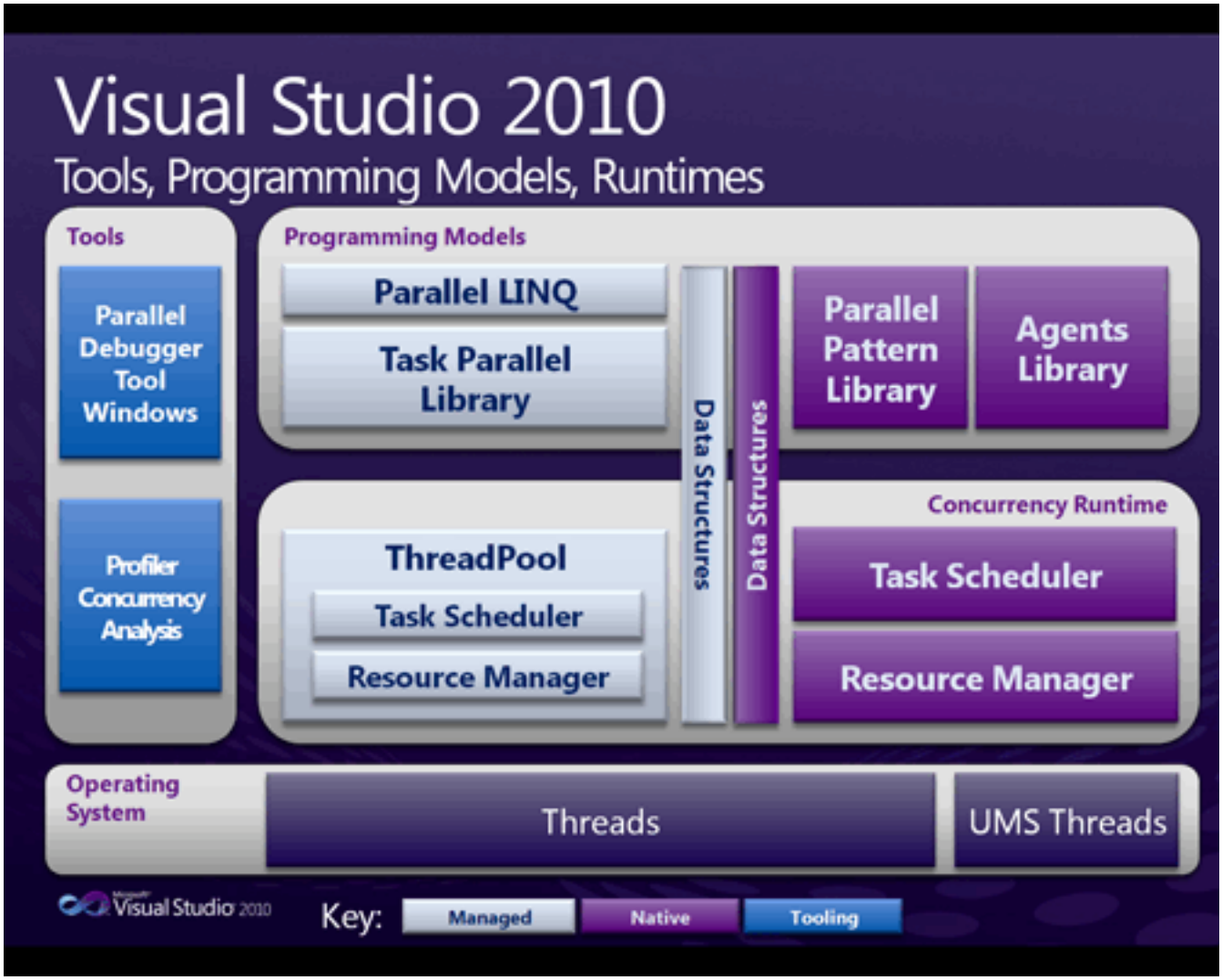
سؤال: برای یافتن آیتم‌های تکراری یک لیست چه باید کرد؟
احتمالا مقاله " روش‌هایی برای حذف رکوردهای تکراری " را به خاطر دارید. اینجا هم می‌توان کوئری LINQ ایی را نوشت که رکوردها را بر اساس سن، گروه بندی کرده و سپس گروه‌هایی را که بیش از یک رکورد دارند، انتخاب نماید.

```
public static void FindDuplicates()
{
   List<Employee> lstEmp = new List<Employee>()
   {
      new  Employee(){ ID=1, Age=20, FName="F1"},
      new  Employee(){ ID=2, Age=21, FName="F2"},
      new  Employee(){ ID=1, Age=20, FName="F1"},
```

```
    };

    var query = from c in lstEmp
                group c by c.Age into g
                where g.Count() > 1
                select new { Age = g.Key, Count = g.Count() };

    foreach (var item in query)
    {
        Console.WriteLine("Age {0} has {1} records", item.Age, item.Count);
    }
}
```

عنوان: **پردازش موازی در vs2010**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۸/۱۸ ۱۰:۴۴:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ

دموی نسبتا مفصلی از توانایی‌های دات نت فریم ورک 4 و vs2010 را که توسط تیم مربوطه در مورد پردازش موازی تهیه شده است، از آدرس زیر می‌توانید دریافت نمائید.



Toub_ParallelismTour_Oct2009.pptx

نظرات خوانندگان

نویسنده: Majid325
تاریخ: ۱۳۸۹/۰۳/۱۲ ۰۹:۲۱:۴۸

سلام
لینک خطای file or directory not found رو میده

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۳/۱۲ ۱۱:۲۰:۲۸

سلام،
اخیرا وبلاگ‌های msdn به روز رسانی شده‌اند و تغییرات زیادی در نرم افزار آن اعمال شده. به همین جهت اکثر لینک‌های قدیمی شاید دیگر کار نکنند.
لینک وبلاگ اصلی اینجا است:
pfxteam

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۳/۱۲ ۱۱:۲۵:۰۷

این هم لینک اصلی به همان مطلب فوق
Slides from Parallelism Tour

عنوان: **Fluent Linq to Sql**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۱۰/۰۵ ۱۲:۳۷:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ

نگارش بعدی یا چهارم entity framework چیزی است شبیه به Fluent NHibernate . یعنی اگر مقاله‌ای را در این زمینه مطالعه کنید و عنوان آن حذف شود، نمی‌توان تشخیص داد که این مقاله مربوط به entity framework است یا Fluent NHibernate. هر چند entity framework حداقل دو نگارش دیگر لازم دارد تا NHibernate را کاملا پشت سر بگذارد.
از آن طرف محبوبیت Linq to SQL هم هنوز پابرجا است و برای مثال سایت پر ترافیکی مثل stack overflow از آن استفاده می‌کند و بسیار هم موفق بوده و کارش را به خوبی انجام می‌دهد.
پروژه مکملی به نام Fluent Linq to Sql با الهام گیری از Fluent NHibernate در سایت codeplex موجود است که این نوع نگاشت‌ها را برای Linq to Sql نیز میسر می‌سازد. به این صورت دیگر نیازی به استفاده از attributes و یا فایل‌های xml نگاشت‌های Linq to Sql نخواهد بود. همچنین مدل کاری اول کد بعد دیتابیس نیز به این صورت محقق می‌شود.

صفحه خانگی
مثالی در مورد استفاده از آن

عنوان: **Any یا Count**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۸/۰۵ ۱۸:۲۴:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ

با وجود امکانات مهیای توسط LINQ ، یک سری از عادات متداول حین کار با گروهی از اشیاء باید کنار گذاشته شوند؛ برای مثال چگونگی بررسی این مطلب که آیا شیء IEnumerable ما حاوی عنصری هست یا خیر.
روش متداول انجام اینکار استفاده از متد Count است. چون این متد پیش از تدارک امکانات LINQ نیز وجود داشته، بنابراین اولین موردی که جهت بررسی آن به ذهن خطور می‌کند، استفاده از متد Count می‌باشد؛ برای مثال:

```
void Method(IEnumerable<Status> statuses)
{
 if (statuses != null && statuses.Count() > 0)
 // do something...
}
```

این روش بهینه نیست زیرا کار متد Count بررسی تک تک عناصر شیء IEnumerable و سپس بازگرداندن تعداد آن‌ها است. این مورد خصوصا در حالت‌های کار با بانک اطلاعاتی و تنظیمات lazy-loading آن و یا تعداد بالای عناصر یک لیست، بسیار هزینه‌بر خواهد شد.
ولی در اینجا هدف ما این است که آیا شیء IEnumerable دارای حداقل یک عنصر است یا خیر؟ بنابراین بجای استفاده از متد Count بهتر است از یکی از extension methods فراهم شده توسط LINQ به نام Any استفاده شود.
کار متد Any ، پس از بررسی اولین عنصر یک مجموعه، خاتمه خواهد یافت و بدیهی است که نسبت به متد Count بسیار سریعتر و کم هزینه‌تر خواهد بود. علاوه بر آن حین کار با بانک‌های اطلاعاتی برای مثال توسط LINQ to Entities ، در SQL نهایی تولیدی به EXISTS ترجمه خواهد شد.

```
void Method(IEnumerable<Status> statuses)
{
 if (statuses != null && statuses.Any())
 // do something...
}
```

خلاصه‌ی بحث:
از این پس حین استفاده از انواع و اقسام لیست‌ها، آرایه‌ها، IEnumerable ها و امثال آن‌ها، جهت بررسی خالی بودن یا نبودن آن‌ها تنها از متد Any فراهم شده توسط LINQ استفاده نمائید.

```
if (myArray != null && myArray.Any())
// do something...
```

نظرات خوانندگان

نویسنده: Meysam
تاریخ: ۱۳۸۹/۰۸/۰۵ ۲۰:۲۷:۰۸

نکته ای که گفتین، زمانی که از dotTrace استفاده بکنید، به وضوح میبینید.

نویسنده: علی اقدم
تاریخ: ۱۳۸۹/۰۸/۰۶ ۰۰:۲۲:۵۳

نکته قابل توجهی بود،ممنون

نویسنده: Hosein Khoshraftar
تاریخ: ۱۳۸۹/۰۸/۱۰ ۰۷:۴۲:۱۳

خیلی نکته جالبی بود
ممنونم . من هم خودم همیشه از کانت استفاده می کردم

عنوان: **تفاوت بین IQueryable و IEnumerable در حین کار با ORMs**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۸/۰۷ ۱۷:۴۴:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ

متد زیر را که یکی از اشتباهات رایج حین استفاده از LINQ خصوصا جهت Binding اطلاعات است، در نظر بگیرید:

```
IQueryable<Customer> GetCustomers()
```

این متد در حقیقت هیچ چیزی را Get نمی‌کند! نام اصلی آن GetQueryableCustomers و یا GetQueryObjectForCustomersاست.
IQueryable قلب LINQ است و تنها بیانگر یک عبارت (expression) از رکوردهایی می‌باشد که مد نظر شما است و نه بیشتر.

```
IQueryable<Customer> youngCustomers = repo.GetCustomers().Where(m => m.Age < 15);
```

برای مثال زمانیکه یک IQueryable را همانند مثال فوق فیلتر می‌کنید نیز هنوز چیزی از بانک اطلاعاتی یا منبع داده‌ای دریافت نشده است. هنوز هیچ اتفاقی رخ نداده است و هنوز رفت و برگشتی به منبع داده‌ای صورت نگرفته است.
به آن باید به شکل یک expression builder نگاه کرد و نه لیستی از اشیاء فیلتر شده‌ی ما. به این مفهوم، deferred execution (اجرای به تاخیر افتاده) نیز گفته می‌شود (باید دقت داشت که IQueryable هم یک نوع IEnumerable است به علاوه expression trees که مهم‌ترین وجه تمایز آن نیز می‌باشد).
برای مثال در عبارت زیر تنها در زمانیکه متد ToList فراخوانی می‌شود، کل عبارت LINQ ساخته شده، به عبارت SQL متناظر با آن ترجمه شده، اطلاعات از دیتابیس اخذ گردیده و حاصل به صورت یک لیست بازگشت داده می‌شود:

```
IList<Competitor> competitorRecords =  competitorRepository
  .Competitors
  .Where(m => !m.Deleted)
  .OrderBy(m => m.countryId)
  .ToList(); //فقط اینجا است که اس کیوال نهایی تولید می‌شود
```

در مورد IEnumerable ها چطور؟

```
IEnumerable<Product> products = repository.GetProducts();
var productsOver25 = products.Where(p => p.Cost >= 25.00);
```

دو سطر فوق به این معنا است:
لطفا ابتدا به بانک اطلاعاتی رجوع کن و تمام رکوردهای محصولات موجود را بازگشت بده. سپس بر روی این حجم بالای اطلاعات، محصولاتی را که قیمت بالای 25 دارند، فیلتر کن.

اگر همین دو سطر را با IQueryable بازنویسی کنیم چطور؟

```
IQueryable<Product> products = repository.GetQueryableProducts();
var productsOver25 = products.Where(p => p.Cost >= 25.00);
```

در سطر اول تنها یک عبارت LINQ ساخته شده است و بس. در سطر دوم نیز به همین صورت. در طی این دو سطر حتی یک رفت و برگشت به بانک اطلاعاتی صورت نخواهد گرفت. در ادامه اگر این اطلاعات به نحوی Select شوند (یا ToList فراخوانی شود، یا در طی یک حلقه برای مثال Iteration ایی روی این حاصل صورت گیرد یا موارد مشابه دیگر)، آنگاه کوئری SQL متناظر با عبارت LINQ فوق ساخته شده و بر روی بانک اطلاعاتی اجرا خواهد شد.

بدیهی است این روش منابع کمتری را نسبت به حالتی که تمام اطلاعات ابتدا دریافت شده و سپس فیلتر می‌شوند، مصرف می‌کند (حالت بازگشت تمام اطلاعات ممکن است شامل 20000 رکورد باشد، اما حالت دوم شاید فقط 5 رکورد را بازگشت دهد).

سؤال: پس IQueryable بسیار عالی است و از این پس کلا از IEnumerable ها دیگر نباید استفاده کرد؟
خیر! توصیه اکید طراحان این است که لطفا تا حد امکان متدهایی که IQueryable بازگشت می‌دهند ایجاد نکنید! IQueryable یعنی اینکه این نقطه‌ی آغازین کوئری در اختیار شما، بعد برو هر کاری که دوست داشتی با آن در طی لایه‌های مختلف انجام بده و هر زمانیکه دوست داشتی از آن یک خروجی تهیه کن. خروجی IQueryable به معنای مشخص نبودن زمان اجرای نهایی کوئری و همچنین مبهم بودن نحوه‌ی استفاده از آن است. به همین جهت متدهایی را طراحی کنید که IEnumerable بازگشت می‌دهند اما در بدنه‌ی آن‌ها به نحو صحیح و مطلوبی از IQueryable استفاده شده است. به این صورت حد و مرز یک متد کاملا مشخص می‌شود. متدی که واقعا همان فیلتر کردن محصولات را انجام می‌دهد، همان 5 رکورد را بازگشت خواهد داد؛ اما با استفاده از یک لیست یا یک IEnumerable و نه یک IQueryable که پس از فراخوانی متد نیز به هر نحو دلخواهی قابل تغییر است.

نظرات خوانندگان

نویسنده: Afshar Mohebbi
تاریخ: ۱۳۸۹/۰۸/۰۷ ۲۲:۰۱:۴۷

جالب بود. من هم چند وقت پیش به این موضوع برخورد کرده بودم: http://stackoverflow.com/questions/3949823/why-skip-and-take-does-not-work-when-passing-through-a-method

حتی یک مطلب کوچولو هم برای آن آماده کرده و در سیستم اتوماتیک وبلاگم برای انتشار گذاشته‌ام.

نویسنده: سامان نام نیک
تاریخ: ۱۳۸۹/۰۸/۰۸ ۱۱:۱۴:۳۸

مدت ها بود که سوال فوق در ذهنم بود
از توضیح مختصر و مفیدتون ممنون

نویسنده: علی اقدم
تاریخ: ۱۳۸۹/۰۸/۰۹ ۱۲:۱۳:۵۱

کاملا درسته، به خاطر Tricky بودن IQueryable شدیدا توصیه می کنم که اگر معماری چند لایه کار می کنید اصلا لایه Bussiness داده ها رو به صورت IQueryable به UI پاس نکنه و در عوض می تونید از IList استفاده کنید.

آقای نصیری بسیار جالب و آموزنده بود،ممنون

نویسنده: کیان
تاریخ: ۱۳۹۱/۰۶/۲۸ ۱۶:۲۰

آیا می‌شه به نوع **IList** بسنده کرد یا کاملا بسته به جایی که استفاده می‌کنیم ممکنه فرق کنه این قضیه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۲۸ ۱۶:۲۸

بستگی به مکان استفاده داره. اگر قرار است دو یا چند جستجو را انجام دهید، اینکارها 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 نشتی لایه سرویس شما در لایه‌های دیگر خواهد شد. IQueryable فقط یک expression است. هنوز اجرا نشده. زمانیکه First، ToList 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 بازگذاشته‌اید 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 نشتی نداشته باشد.
طراحی یک لایه سرویس که خروجی IQueryable دارد نشتی دار درنظر گرفته شده و توصیه نمی‌شود. اصطلاحا 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: رضا بزرگی
تاریخ: ۱۳۹۱/۰۶/۲۹ ۹:۱۱

تفاوت بازگشت متد از نوع List و IList در اینجا چیست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۲۹ ۹:۳۱

این‌ها بیشتر مباحث طراحی API است. اگر از List استفاده کنید، مصرف کننده کتابخانه شما مجبور است فقط از List استفاده کند. List صرفا یک پیاده سازی خاص از IList است.

اگر از اینترفیس و قرارداد IList استفاده شود، آزادی عمل بیشتری را در اختیار مصرف کننده خواهید گذاشت. در اینجا مصرف کننده می‌تواند از هر پیاده سازی دلخواهی از IList برای کار با API شما استفاده کند. حتی مواردی که در زمان طراحی API اصلی وجود خارجی نداشته‌اند و بعدها پیاده سازی خواهند شد.

عنوان: **نحوه‌ی صحیح فراخوانی SQL Aggregate Functions حین استفاده از LINQ**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۸/۱۳ ۱۸:۳۳:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ

SQL Aggregate Functions

که مد نظر شما هستند مانند Sum ، Max ، Min و امثال آن. بحث LINQ هم زمانیکه از الگوی Repository استفاده شود مستقل از نوع ORM مورد نظر خواهد شد؛ بنابراین در اینجا مقصود از LINQ می‌تواند LINQ to NHibernate ، LINQ to Entities ، LINQ to SQL و کلا هر نوع ORM دیگری با پشتیبانی از LINQ باشد.

صورت مساله هم این است: آیا نوشتن عبارت LINQ ایی به شکل زیر صحیح است؟

```
decimal amount = respository.Transactions
                                    .Where(t=>t.TransactionDate>new DateTime(2010,10,13))
                                    .Sum(t=>t.Amount);
```

پاسخ: خیر!

توضیحات:

عبارت LINQ فوق در نهایت به شکل زیر ترجمه خواهد شد:

```
-- Region Parameters
-- @p0: DateTime [2010/10/13 12:00:00 ظ.ق]
-- EndRegion
SELECT SUM([t0].[Amount]) AS [value]
FROM [Transactions] AS [t0]
WHERE [t0].[TransactionDate] > @p0
```

و اتفاقا در این سیستم پس از تاریخ 13/10/2010 هیچ تراکنشی ثبت نشده است؛ بنابراین خروجی این کوئری null خواهد بود و نه صفر. همینجا است که یکی از استثناهای زیر صادر شده و ادامه‌ی برنامه با مشکل مواجه خواهد شد:

```
- System.InvalidOperationException: The cast to value type 'decimal' failed because the materialized
value is null.
- InvalidOperationException: The null value cannot be assigned to a member with type decimal which is a
non-nullable value type.
```

مشکل هم از اینجا ناشی می‌شود که متغییری از نوع deciaml یا int و امثال آن، مقدار دریافتی نال را نمی‌پذیرند. برای رفع این مشکل باید عبارت LINQ فوق به صورت زیر بازنویسی شود (و اهمیتی هم ندارد که Sum است یا Max یا Avg و غیره؛ در مورد بکارگیری تمام SQL Aggregate Functions در یک عبارت LINQ ، این مورد باید لحاظ گردد):

```
decimal amount = respository.Transactions
```

```
.Where(t=>t.TransactionDate>new DateTime(2010,10,13))
```

```
.Sum(t=>
```

```
(?decimal)
```

```
t.Amount)
```

```
0??
```

```
;
```

دقیقا به همین علت است که در دات نت، nullable types تعریف شده‌اند. امکان ذخیره سازی null در یک متغیر برای مثال از نوع decimal وجود ندارد اما نوع decimal? (و یا Nullable<decimal> به بیانی دیگر) این قابلیت را دارد.
شاید بگوئید که در اینجا با تغییر تعریف متغیر به decimal? amount مشکل حل می‌شود، اما خیر. تعریف extension method مربوط به sum به صورت زیر است:

```
public static

TResult

Sum<TSource>(

this IQueryable<TSource> source,

Expression<Func<TSource,

TResult

>> selector)
```

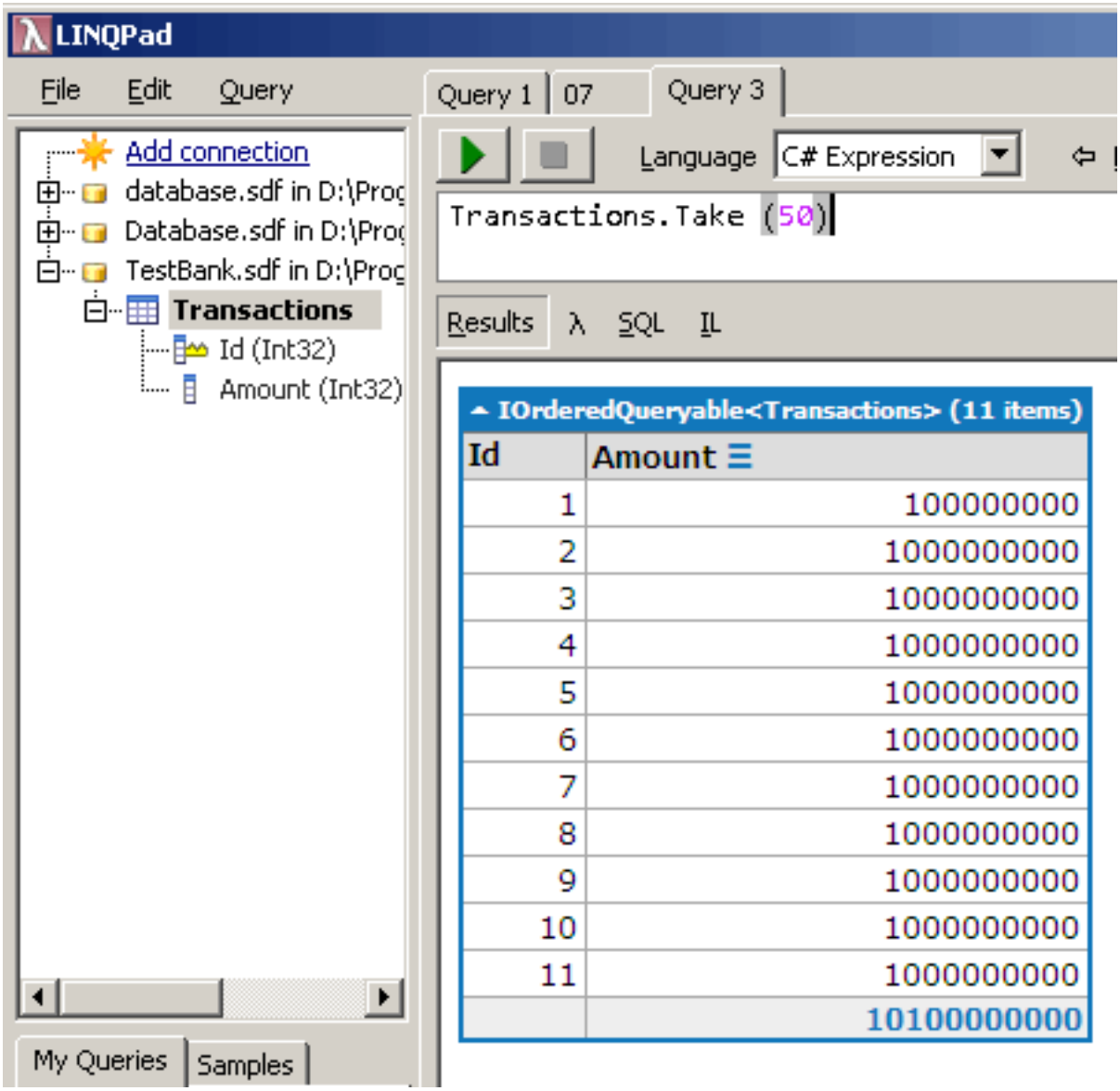
در این تعریف به TResult دقت نمائید؛ هم بیانگر نوع خروجی نهایی متد و هم مشخص سازنده‌ی نوع پارامتری است که خروجی Lambda Expression را تشکیل می‌دهد. به این معنا که سی شارپ، TResult را از lambda expression دریافت کرده و خروجی Sum را بر همان مبنا و نوع تشکیل می‌دهد. بنابراین برای دریافت خروجی nullable باید TResult ایی nullable را همانند مثال فوق ایجاد کنیم.

خلاصه بحث:
اگر در کدهای LINQ خود که با بانک اطلاعاتی سر و کار دارند از معادل‌های SQL Aggregate Functions استفاده کرده‌اید، آن‌ها را یافته و نکته‌ی nullable TResult فوق را به آن‌ها اعمال کنید؛ در غیر اینصورت منتظر باشید تا روزی برنامه شما به سادگی کرش کند.

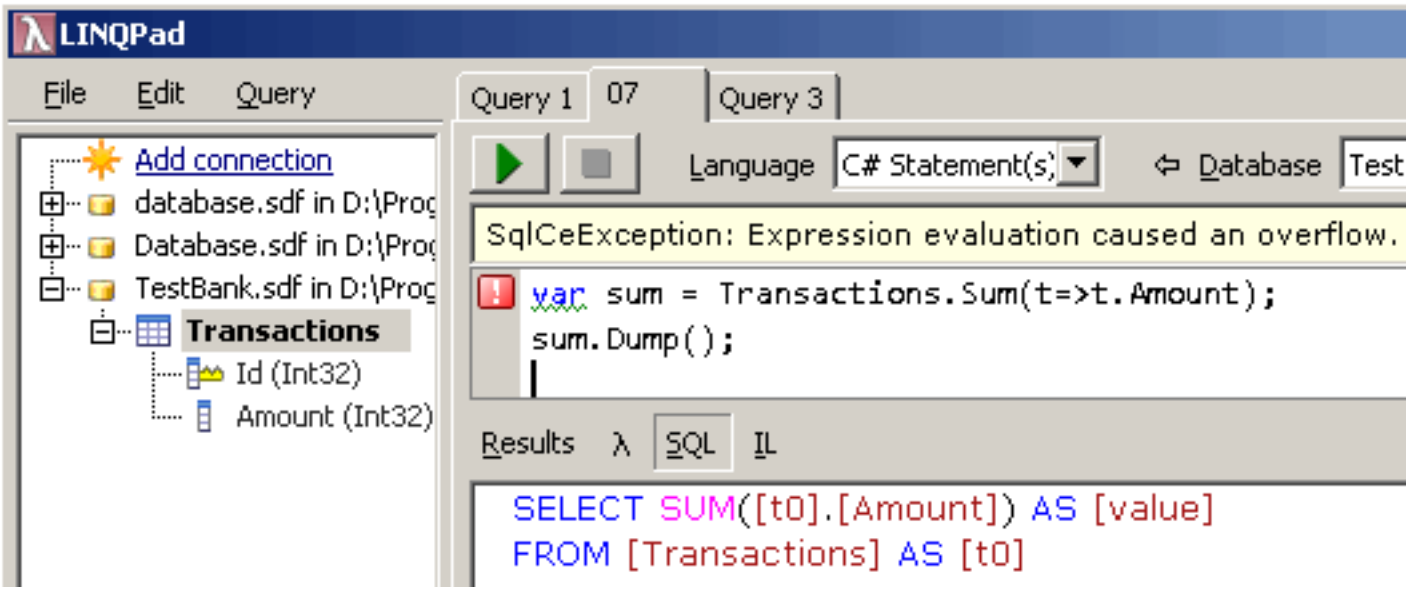
عنوان: **نحوه‌ی صحیح فراخوانی SQL Aggregate Functions حین استفاده از LINQ - قسمت دوم**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۸/۲۲ ۲۰:۳۶:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ

در قسمت قبل در مورد حالتی که کوئری انجام شده نتیجه‌ای را بر نگردانده است، بحث شد. در این قسمت یکی از شایع‌ترین مشکلات حین کار با تابع Sum بررسی خواهد شد.

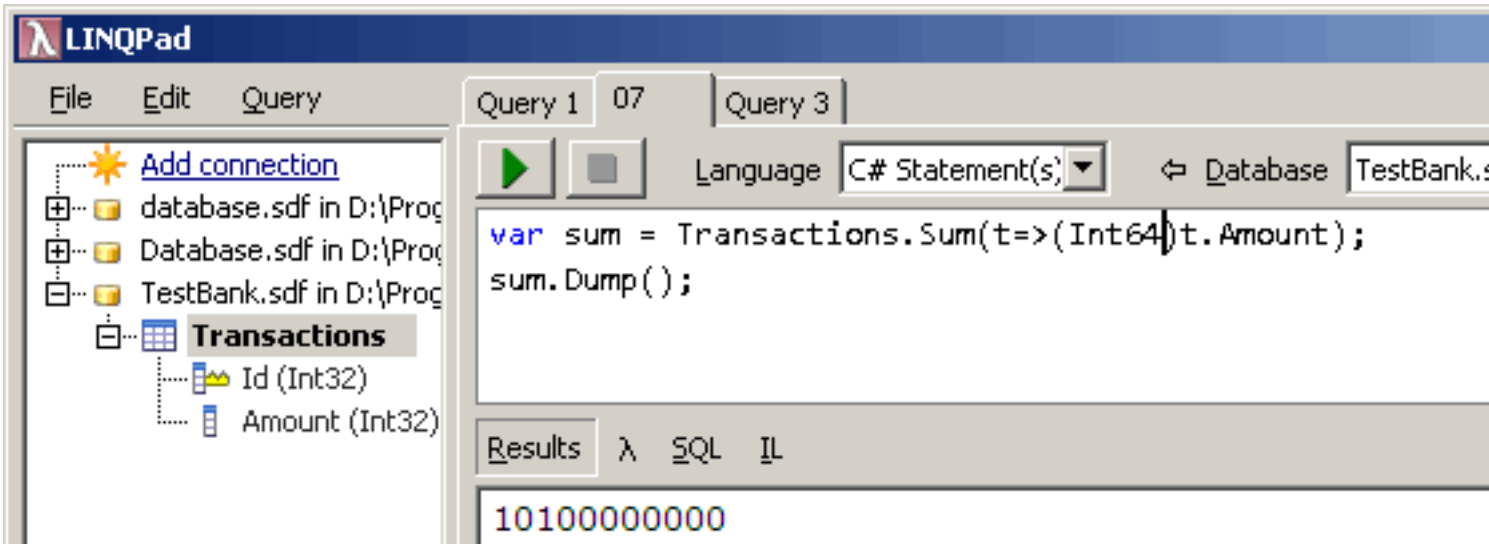
ابتدا جدول ساده Transactions را با دو فیلد Id و Amount مطابق تصویر زیر در نظر بگیرید:



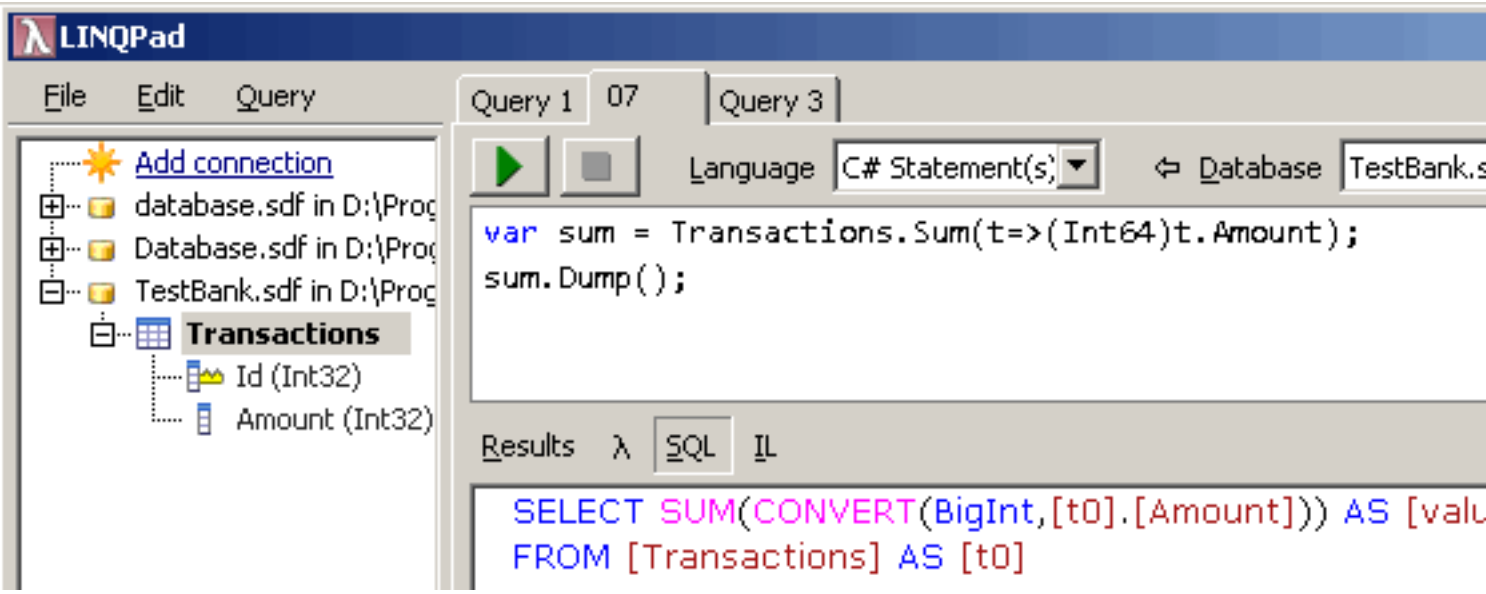
تعدادی رکورد در این جدول ثبت شده‌اند. اکنون می‌خواهیم جمع آن‌ها را محاسبه کنیم:



همانطور که ملاحظه می‌نمائید این عملیات میسر نیست، زیرا حاصل نهایی فراتر از بازه‌ی تعریف شده‌ی Int32 است.
برای رفع این مشکل باید Amount را تبدیل به BigInt (برای مثال مرتبط با نگارش‌های مختلف SQL Server) کرد. مطابق توضیحات قسمت قبل، این عملیات casting باید به lambda expression تعریف شده اعمال گردد، زیرا خروجی Sum بر مبنای آن تعیین می‌گردد.



در این حالت خروجی SQL آن نیز به صورت زیر در خواهد آمد:



هر چند این مباحث ساده به نظر می‌رسند ولی در صورت عدم رعایت سبب سرخ و سفید شدن در هنگام مقتضی خواهند گردید.

نظرات خوانندگان

نویسنده: سامان نام نیک
تاریخ: ۱۳۸۹/۰۸/۲۳ ۱۱:۴۸:۴۵

با سلام
مطالب مختصر و مفید بود
امیدورام به همین روند ادامه بدین
راستی آقای نصیری یه توضیح کوچولو در مورد Linqpad میدین .
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۸/۲۳ ۱۶:۵۲:۰۵

LINQPad یک برنامه‌ی نیمه رایگان است. به این معنا که دریافت آن رایگان است، استفاده از آن هیچ محدودیتی ندارد. فقط هنگام نوشتن کوئری‌ها intellisense ظاهر نخواهد شد. این یک مورد رایگان نیست و برای فعال شدن آن باید مقداری هزینه کنید. کیفیت intellisense آن هم قابل مقایسه است با VS.NET و بسیار مطلوب است.
LINQPad برای تست کردن سریع عبارات LINQ فوق العاده است. با استفاده از آن بدون نیاز به VS.NET خیلی سریع و در عرض چند ثانیه می‌تونید عبارات LINQ خودتون رو نوشته و تست کنید. این LINQ می‌تونه LINQ to Objects باشه یا LINQ to SQL یا LINQ to Entities و غیره.
خلاصه چیزی شبیه به management studio مخصوص SQL Server را تصور کنید که اینبار بجای SQL نویسی، LINQ می‌نویسید، حاصل را نمایش می‌دهد؛ علاوه بر آن خروجی SQL تولیدی و حتی IL نهایی را هم نمایش می‌دهد که برای دیباگ بسیار مفید است. به همراه آن یک سری مثال هم وجود دارد که جهت فراگیری LINQ یا حتی استفاده از آن‌ها به عنوان مرجع بی‌نظیر است.

نویسنده: Amir Madadi
تاریخ: ۱۳۹۰/۰۲/۰۷ ۰۶:۳۵:۲۶

تصاویر (کدهای) این مطلب دیده نمی شود ، لطفا اصلاح بفرمایید ، ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۲/۰۷ ۰۸:۱۸:۴۵

لطفا فایل خلاصه وبلاگ را از سمت راست، بالای سایت، قسمت گزیده‌ها دریافت کنید. تمام تصاویر در آن موجود است

عنوان: **تهیه گزارشات Crosstab به کمک LINQ**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۸/۲۲ ۲۲:۵۲:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ

در گزارشات Crosstab، ردیف‌های یک گزارش، تبدیل به ستون‌های آن می‌شوند؛ به همین جهت به آن‌ها Pivot tables هم می‌گویند.
برای مثال فرض کنید که قصد دارید گزارش تعداد ساعت کارکرد را به ازای هر پروژه در طول چند ماه تعیین کنید. گزارش متداول از این نوع اطلاعات، یک لیست بلند بالای بی‌مفهوم است. این گزارش تشکیل شده از صدها رکورد به ازای کارکنان مختلف در پروژه‌های مختلف و ... هیچ ارزش آماری خاصی ندارد. یک گزارش بدوی است. زمانیکه این گزارش را تبدیل به حالت crosstab می‌کنیم، اولین ستون فقط یک شماره پروژه خواهد بود و ستون‌های بعدی، مثلا نام ماه‌ها و مقادیر آن‌ها هم جمع کارکرد افراد بر روی یک پروژه مشخص.

مثال اول) تهیه گزارش Crosstab جمع هزینه‌های واحدهای مختلف به تفکیک ماه

کلاس هزینه‌های زیر را در نظر بگیرید که به کمک آن می‌توان به ازای هر واحد یا دپارتمان در تاریخ‌های متفاوت، هزینه‌ای را مشخص ساخت:

```
using System;

namespace Pivot.Sample1
{
    public class Expense
    {
        public DateTime Date { set; get; }
        public string Department { set; get; }
        public decimal Expenses { set; get; }
    }
}
```

با توجه به این کلاس، یک منبع داده آزمایشی جهت تهیه گزارشات، می‌تواند به صورت زیر باشد:

```
using System;
using System.Collections.Generic;

namespace Pivot.Sample1
{
    public class ExpenseDataSource
    {
        public static IList<Expense> ExpensesDataSource()
        {
            return new List<Expense>
            {
                new Expense { Date = new DateTime(2011,11,1), Department = "Computer", Expenses = 100
},
                new Expense { Date = new DateTime(2011,11,1), Department = "Math", Expenses = 200 },
                new Expense { Date = new DateTime(2011,11,1), Department = "Physics", Expenses = 150 },

                new Expense { Date = new DateTime(2011,10,1), Department = "Computer", Expenses = 75 },
                new Expense { Date = new DateTime(2011,10,1), Department = "Math", Expenses = 150 },
                new Expense { Date = new DateTime(2011,10,1), Department = "Physics", Expenses = 130 },

                new Expense { Date = new DateTime(2011,9,1), Department = "Computer", Expenses = 90 },
                new Expense { Date = new DateTime(2011,9,1), Department = "Math", Expenses = 95 },
                new Expense { Date = new DateTime(2011,9,1), Department = "Physics", Expenses = 100 }
            };
        }
```

```
    }
}
```

و اگر این لیست را به همین شکلی که هست نمایش دهیم، خروجی زیر را خواهیم داشت:

List<Expense> (9 items)

Date	Department	Expenses
2011/11/01 12:00:00 ق.ظ	Computer	100
2011/11/01 12:00:00 ق.ظ	Math	200
2011/11/01 12:00:00 ق.ظ	Physics	150
2011/10/01 12:00:00 ق.ظ	Computer	75
2011/10/01 12:00:00 ق.ظ	Math	150
2011/10/01 12:00:00 ق.ظ	Physics	130
2011/09/01 12:00:00 ق.ظ	Computer	90
2011/09/01 12:00:00 ق.ظ	Math	95
2011/09/01 12:00:00 ق.ظ	Physics	100
		1090

که ... خروجی مطلوبی نیست. در اینجا ما فقط 9 رکورد داریم؛ اما در عمل به ازای هر روز، یک رکورد می‌تواند وجود داشته باشد و این لیست طولانی، هیچ ارزش آماری خاصی ندارد. می‌خواهیم سرستون‌های گزارش ما مطابق جدول زیر باشند:

Month	ComputerDepartment	MathDepartment	PhysicsDepartment

یعنی اگر سه ماه را در نظر بگیریم با هر تعداد رکورد، فقط سه ردیف به ازای هر ماه باید حاصل شود و ستون‌های دیگر هم نام بخش‌ها یا واحدهای موجود باشند.
برای رسیدن به این خروجی Crosstab، می‌توان کوئری LINQ زیر را به کمک امکانات گروه بندی اطلاعات آن تهیه کرد:

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

namespace Pivot.Sample1
{
    public class PivotTable
    {
        public static IList ExpensesCrossTab()
        {
            return ExpenseDataSource
```

```
                        .ExpensesDataSource()
                        .GroupBy(t =>
                                  new
                                  {
                                      Year = t.Date.Year,
                                      Month = t.Date.Month
                                  })
                        .Select(myGroup =>
                                  new
                                  {
                                      //Year = myGroup.Key.Year,
                                      Month = myGroup.Key.Month,
                                      ComputerDepartment = myGroup.Where(x => x.Department ==
"Computer").Sum(x => x.Expenses),
                                      MathDepartment = myGroup.Where(x => x.Department ==
"Math").Sum(x => x.Expenses),
                                      PhysicsDepartment = myGroup.Where(x => x.Department ==
"Physics").Sum(x => x.Expenses)
                                  })
                        .ToList();
        }
    }
}
```

که اینبار خروجی زیر را تولید می‌کند.

IEnumerable<> (3 items)

Month	ComputerDepartment	MathDepartment	PhysicsDepartment
11	100	200	150
10	75	150	130
9	90	95	100
	265	445	380

اگر علاقمند باشید که مثال فوق را در برنامه‌ی LINQPad آزمایش کنید، این فایل را دریافت نموده و در آن برنامه باز نمائید.

مثال دوم) تهیه لیست Crosstab حضور و غیاب افراد در طول یک هفته

کلاس StudentStat را جهت ثبت اطلاعات حضور یک دانشجو، می‌توان به شکل زیر تعریف کرد:

```
using System;

namespace Pivot.Sample2
{
    public class StudentStat
    {
        public int Id { set; get; }
        public string Name { set; get; }
        public DateTime Date { set; get; }
        public bool IsPresent { set; get; }
    }
}
```

و بر همین اساس یک منبع داده فرضی جهت انجام گزارشات می‌تواند به نحو زیر تهیه شود:

```
using System;
using System.Collections.Generic;

namespace Pivot.Sample2
{
    public class StudentsStatDataSource
    {
        public static IList<StudentStat> CreateMonthlyReportDataSource()
        {
            var result = new List<StudentStat>();
            var rnd = new Random();

            for (int day = 1; day < 6; day++)
            {
                for (int student = 1; student < 6; student++)
                {
                    result.Add(new StudentStat
                    {
                        Id = student,
                        Date = new DateTime(2011, 11, day),
                        IsPresent = rnd.Next(-1, 1) == 0 ? true : false,
                        Name = "student " + student
                    });
                }
            }

            return result;
        }
    }
}
```

خروجی این گزارش هم در این حالت ساده با 5 دانشجو و فقط 5 روز، 25 رکورد خواهد بود:

List<StudentStat> (25 items)

Id	Name	Date	IsPresent
1	student 1	2011/11/01 12:00:00 ق.ظ	True
2	student 2	2011/11/01 12:00:00 ق.ظ	False
3	student 3	2011/11/01 12:00:00 ق.ظ	True
4	student 4	2011/11/01 12:00:00 ق.ظ	False
5	student 5	2011/11/01 12:00:00 ق.ظ	False
1	student 1	2011/11/02 12:00:00 ق.ظ	False
2	student 2	2011/11/02 12:00:00 ق.ظ	True
3	student 3	2011/11/02 12:00:00 ق.ظ	True
4	student 4	2011/11/02 12:00:00 ق.ظ	False
5	student 5	2011/11/02 12:00:00 ق.ظ	True
1	student 1	2011/11/03 12:00:00 ق.ظ	True
2	student 2	2011/11/03 12:00:00 ق.ظ	False
3	student 3	2011/11/03 12:00:00 ق.ظ	True
4	student 4	2011/11/03 12:00:00 ق.ظ	False
5	student 5	2011/11/03 12:00:00 ق.ظ	True
1	student 1	2011/11/04 12:00:00 ق.ظ	False
2	student 2	2011/11/04 12:00:00 ق.ظ	True
3	student 3	2011/11/04 12:00:00 ق.ظ	True
4	student 4	2011/11/04 12:00:00 ق.ظ	True
5	student 5	2011/11/04 12:00:00 ق.ظ	True
1	student 1	2011/11/05 12:00:00 ق.ظ	False
2	student 2	2011/11/05 12:00:00 ق.ظ	True
3	student 3	2011/11/05 12:00:00 ق.ظ	True
4	student 4	2011/11/05 12:00:00 ق.ظ	True
5	student 5	2011/11/05 12:00:00 ق.ظ	False

که ... این هم آنچنان از لحاظ آماری مطلوب و مفهوم نیست. می‌خواهیم سطرهای این گزارش همانند لیست واقعی حضورغیاب، فقط از نام افراد تشکیل شود و همچنین ستون‌ها مثلا شماره یا نام روزهای یک هفته یا ماه باشند. مثلا به شکل زیر:

Id	Name	Day1IsPresent	Day2IsPresent	Day3IsPresent	Day4IsPresent	Day5IsPresent	PresentsCount	AbsentsCount

برای رسیدن به این خروجی Crosstab، مثلا می‌توان از کوئری LINQ زیر کمک گرفت که بر اساس شماره دانشجویی اطلاعات را گروه بندی کرده است:

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

namespace Pivot.Sample2
{
    public class PivotTable
    {
        public static IList StudentsStatCrossTab()
        {
            return StudentsStatDataSource
                        .CreateWeeklyReportDataSource()
                        .GroupBy(x =>
                                 new
                                 {
                                     x.Id
                                 })
                        .Select(myGroup =>
                                 new
                                 {
                                     myGroup.Key.Id,
                                     Name = myGroup.First().Name,
                                     Day1IsPresent = myGroup.Where(x => x.Date.Day ==
1).First().IsPresent,
                                     Day2IsPresent = myGroup.Where(x => x.Date.Day ==
2).First().IsPresent,
                                     Day3IsPresent = myGroup.Where(x => x.Date.Day ==
3).First().IsPresent,
                                     Day4IsPresent = myGroup.Where(x => x.Date.Day ==
4).First().IsPresent,
                                     Day5IsPresent = myGroup.Where(x => x.Date.Day ==
5).First().IsPresent,
                                     PresentsCount = myGroup.Where(x => x.IsPresent).Count(),
                                     AbsentsCount = myGroup.Where(x => !x.IsPresent).Count()
                                 })
                        .ToList();
        }
    }
}
```

و این کوئری خروجی زیر را تولید می‌کند که از هر لحاظ نسبت به لیست قبلی مفهوم‌تر است:

IEnumerable<> (5 items)

Id	Name	Day1IsPresent	Day2IsPresent	Day3IsPresent	Day4IsPresent	Day5IsPresent	PresentsCount	AbsentsCount
1	student 1	True	False	False	True	True	3	2
2	student 2	False	True	False	True	False	2	3
3	student 3	False	False	False	False	False	0	5
4	student 4	True	True	True	False	True	4	1
5	student 5	True	True	False	True	True	4	1
							13	12

فایل LINQPad این مثال را می‌توانید از اینجا دریافت کنید.

عنوان: **تهیه گزارشات Crosstab به کمک LINQ - قسمت دوم**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۸/۲۳ ۲۲:۴۳:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ

اگر به قسمت اول « تهیه گزارشات Crosstab به کمک LINQ » دقت کرده باشید، یک مشکل کوچک دارد و آن هم لزوم مشخص سازی دقیق ستون‌هایی است که می‌خواهیم در گزارش ظاهر شوند. مثلا دقیقا مشخص کنیم که نام واحد چیست یا دقیقا روز را مشخص کنیم. این مورد برای گزارش‌های کوچک مشکلی ندارد؛ ولی اگر همان مثال دوم را در نظر گرفته و بازه را کمی بیشتر کنیم، مثلا یک ماه، آن وقت باید حداقل 30 بار بنویسیم Day1IsPresent تا ... Day30IsPresent و یا اگر بازه‌ی گزارشگیری به اختیار کاربر باشد آن وقت چه باید کرد؟ مثلا یکبار 7 روز پایان ماه را انتخاب کند، یکبار 14 روز را، شاید یک بار هم مثلا 90 روز را مد نظر داشته باشد (تعداد ستون‌ها متغیر باشد یا به عبارتی Dynamic Crosstab نیاز است ایجاد شود).
برای حل این مساله، می‌توان از متد الحاقی زیر از سایت extensionmethod.net کمک گرفت:

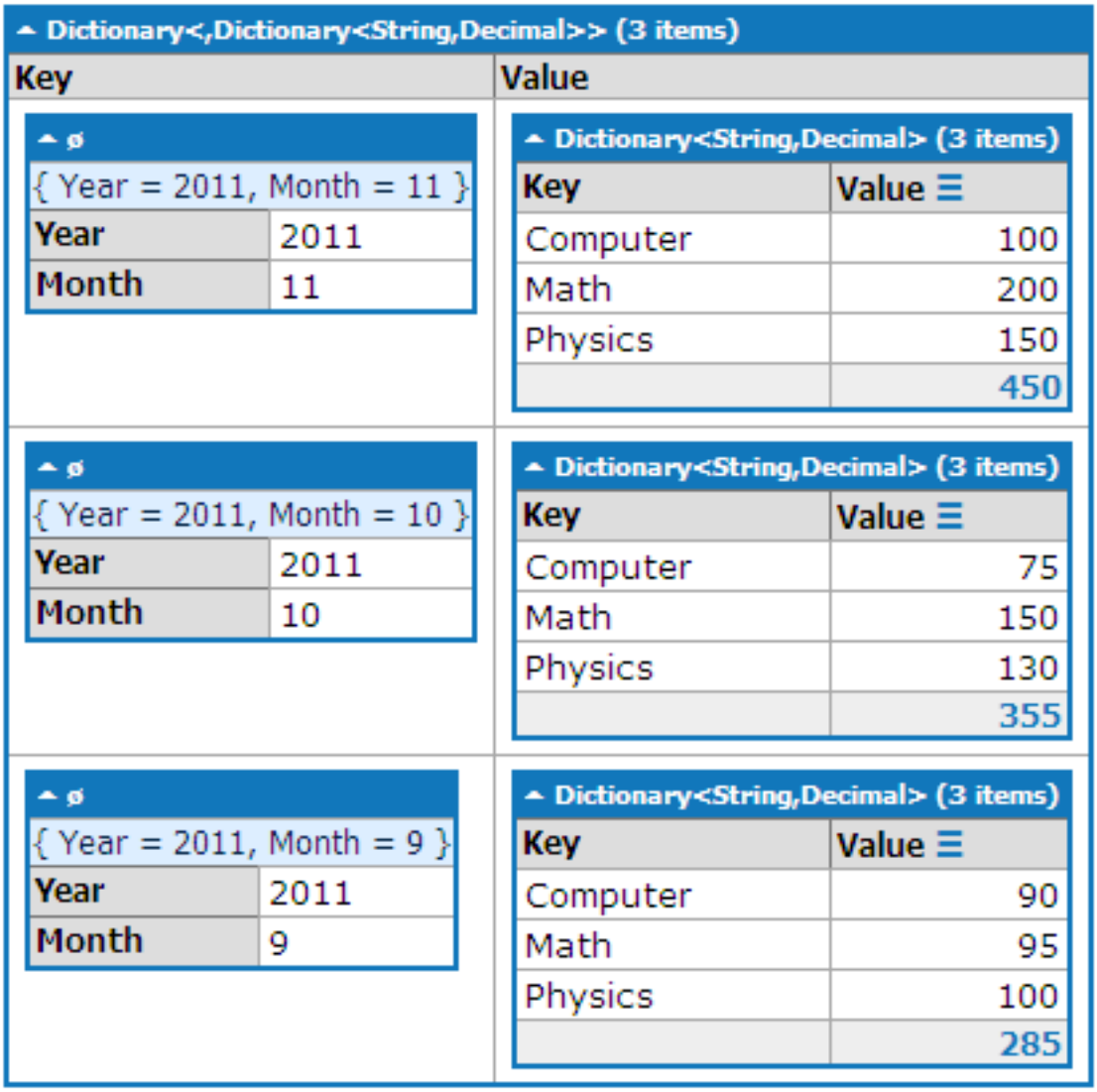
```
using System;
using System.Collections.Generic;
using System.Linq;

namespace PivotExtensions
{
    public static class Ext
    {
        public static Dictionary<TKey1, Dictionary<TKey2, TValue>>
                        Pivot<TSource, TKey1, TKey2, TValue>
                        (
                            this IEnumerable<TSource> source,
                            Func<TSource, TKey1> key1Selector,
                            Func<TSource, TKey2> key2Selector,
                            Func<IEnumerable<TSource>, TValue> aggregate
                        )
        {
            return source.GroupBy(key1Selector)
                         .Select(
                            key1Group => new
                                {
                                    Key = key1Group.Key,
                                    Value = key1Group.GroupBy(key2Selector)
                                         .Select(
                                            key2Group => new
                                               {
                                                   K = key2Group.Key,
                                                   V = aggregate(key2Group)
                                               })
                                         .ToDictionary(e => e.K, o => o.V)
                                })
                         .ToDictionary(e => e.Key, o => o.Value);
        }
    }
}
```

در این متد:
key1Selector مشخص کننده ستون‌های ثابت و مشخص سمت راست یا چپ (بر اساس جهت صفحه) گزارش است. در سیستم‌های مختلف این ستون‌ها نام‌هایی مانند leftColumn ، keyColumn و Row Heading ممکن است داشته باشند.
key2Selector ستون‌های پویای گزارش را تشکیل می‌دهد. در سایر سیستم‌ها این پارامتر، VariableColumn، pivotNameColumn ، topField و یا Column Heading هم نامیده می‌شود.
Aggregate در اینجا مشخص می‌کند که مقادیر ستون‌های پویای یاد شده چگونه باید محاسبه شوند.

با توجه به این متد، برای نمونه جهت حل مثال اول قسمت قبل خواهیم داشت:

```
var list = ExpenseDataSource.ExpensesDataSource();
var pivotList = list.Pivot(
                x =>
                    new
                    {
                        x.Date.Year,
                        x.Date.Month
                    },
                  x1 => x1.Department,
                  x2 => x2.Sum(x => x.Expenses));
```

با خروجی



Dictionary<,Dictionary<String,Decimal>> (3 items)

Key	Value
ø: { Year = 2011, Month = 11 } — Year: 2011, Month: 11	Dictionary<String,Decimal> (3 items): Computer 100, Math 200, Physics 150, 450
ø: { Year = 2011, Month = 10 } — Year: 2011, Month: 10	Dictionary<String,Decimal> (3 items): Computer 75, Math 150, Physics 130, 355
ø: { Year = 2011, Month = 9 } — Year: 2011, Month: 9	Dictionary<String,Decimal> (3 items): Computer 90, Math 95, Physics 100, 285

فایل LINQPad آن از اینجا قابل دریافت است.

و برای حل مثال دوم قسمت قبل می‌توان نوشت:

```
var list2 = StudentsStatDataSource.CreateWeeklyReportDataSource();
var lst = list2.Pivot(
                x =>
                    new
                    {
                        x.Id,
                        x.Name
                    },
            x1 => "Day " + x1.Date.Day,
            x2 => x2.First().IsPresent);
```

با خروجی

Dictionary<,Dictionary<String,Boolean>> (5 items)

Key	Value
ø { Id = 1, Name = student 1 } Id: 1 Name: student 1	Dictionary<String,Boolean> (5 items) Day 1: False Day 2: False Day 3: True Day 4: False Day 5: True
ø { Id = 2, Name = student 2 } Id: 2 Name: student 2	Dictionary<String,Boolean> (5 items) Day 1: False Day 2: True Day 3: True Day 4: True Day 5: True
ø { Id = 3, Name = student 3 } Id: 3 Name: student 3	Dictionary<String,Boolean> (5 items) Day 1: False Day 2: True Day 3: True Day 4: True Day 5: False
ø { Id = 4, Name = student 4 } Id: 4 Name: student 4	Dictionary<String,Boolean> (5 items) Day 1: True Day 2: True Day 3: False Day 4: True Day 5: True
ø { Id = 5, Name = student 5 } Id: 5 Name: student 5	Dictionary<String,Boolean> (5 items) Day 1: False Day 2: False Day 3: True Day 4: False Day 5: True

فایل LINQPad آن از اینجا قابل دریافت است.

نظرات خوانندگان

نویسنده: ZF
تاریخ: ۰۹:۲۹:۲۱ ۱۳۹۰/۰۸/۲۵

سلام آقای نصیری
ممنون از مطلب مفیدتون.شما در اول این مطلب فرمودید :« اگر بازه‌ی گزارشگیری به اختیار کاربر باشد» اما مثالهایی که زدین رو باید از اول دونست که چه ستونهایی رو میخواهیم.منطورم اینه که فرض کنید که اگر ما یک چک لیست برای ماههای سال داشته باشیم که کاربر بتونه هر ترتیبی از 12 ماه رو انتخاب کنه چکار باید کرد؟ با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۰:۲۲:۵۷ ۱۳۹۰/۰۸/۲۵

سلام،
نه؛ ما اینجا هم نمی‌دونیم که مثلا CreateWeeklyReportDataSource چی هست. فقط می‌دونیم که یک لیست نهایی تهیه و به متد Pivot ارسال شده. شما در این قسمت (در حین تهیه متد CreateWeeklyReportDataSource) فرصت دارید که دیتاسورس مناسبی رو تهیه کنید.
در مورد چک لیست هم به همین صورت. مهم تشکیل List دیتاسورس اولیه است. مابقی توسط متد Pivot مدیریت می‌شود.
یک مثال جدید LINQPad رو اینجا اضافه کردم که در آن تعداد روزها 30 هست و ضمنا یک شرط Where هم به آن اعمال شده که مثلا کاربر روزهای 10 تا 23 رو به دلخواه انتخاب کرده (و برنامه از اول نمی‌دونه که چه بازه‌ای مد نظر هست): sample05.linq

نویسنده: ZF
تاریخ: ۱۰:۴۸:۰۹ ۱۳۹۰/۰۸/۲۵

بسیار عالی بود متشکرم

نویسنده: وحید نصیری
تاریخ: ۱۲:۱۲:۴۱ ۱۳۹۰/۰۸/۲۹

کتابی اخیرا منتشر شده به نام Pivot Table Data Crunching, Microsoft Excel 2010 که این مفاهیم Pivot و crosstab رو مفصل در طی 380 صفحه توضیح داده.

عنوان: **LINQ to Sharepoint Class**
نویسنده: محمد باقر سیف اللهی
تاریخ: ۱۳۹۱/۰۴/۰۷ ۱۱:۷
آدرس: www.dotnettips.info
برچسب‌ها: LINQ, SharePoint

شیرپوینت قابلیت استفاده از دستورات LINQ را برای دسترسی به لیست‌های خود می‌دهد . این قابلیت جایگزین خوبی برای استفاده سنتی از **CAML queries** می باشد. (*CAML*) Collaborative Application Markup Language برای تعریف کوئری‌ها درون لیست داده‌ها استفاده می‌شود و بر مبنای XML می‌باشد بیشتر
برای این منظور می‌توان از دستور زیر استفاده کرد تا Sharepoint Foundation برای شما کلاسی بسازد تا بتوانید به تمام اعضای لیست داده‌های Sharepoint دسترسی داشته باشید
لازم است بدانید که برای ساخت این کلاس از LINQ Provider Code Generator خود شیرپوینت به نام SPMETAL.EXE که در پوشه bin در زیر مجموعه 14 قرار دارد استفاده می‌شود .

```
spmetal.exe / web : http://DOMAIN /code : c:\EntityModuleSharePoint.cs /user :DOMAIN \ USERNAME /password :
********** /namespace : EntityModuleSharePoint
```

توجه داشته باشید که پسوند فایلی که نوشته اید مشخص می‌کند که دستورات بر مبنای چه زبانی ساخته شوند (قسمت قرمز رنگ بعد از سوییچ code) مثلا با تغیر پسوند به VB فایل ایجاد شده با دستورات vb قابل استفاده است

برای مشاهده یک مثال از این مورد ، به اینجا مراجعه نمایید
موفق باشید

نظرات خوانندگان

نویسنده: هیمن روحانی
تاریخ: ۱۳۹۲/۰۵/۲۹ ۱۳:۱۴

لطفا اگه امکان داره در مورد join لیستهای شیرپوینت با LINQ توضیح بدید.

نویسنده: محمد باقر سیف اللهی
تاریخ: ۱۳۹۲/۰۶/۰۱ ۰:۳۳

منبع در مورد Join زیاد هست :

http://msdn.microsoft.com/en-us/library/ee539975(v=office.14).aspx http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spquery.joins(office.14).aspx

http://blogs.msdn.com/b/kaevans/archive/2012/01/20/sharepoint-2010-caml-list-joins.aspx

/http://espidi.com.au/blogs/left-join-with-multiple-lists-in-sharepoint-2010caml-and-linq

http://msdn.microsoft.com/en-us/library/ff798478.aspx

/http://solutionizing.net/2009/04/30/join-sharepoint-lists-with-linq

http://www.nullskull.com/a/1412/join-lists-with-linq--sharepoint-2010.aspx

http://msdn.microsoft.com/en-us/library/ee538250(v=office.14).aspx

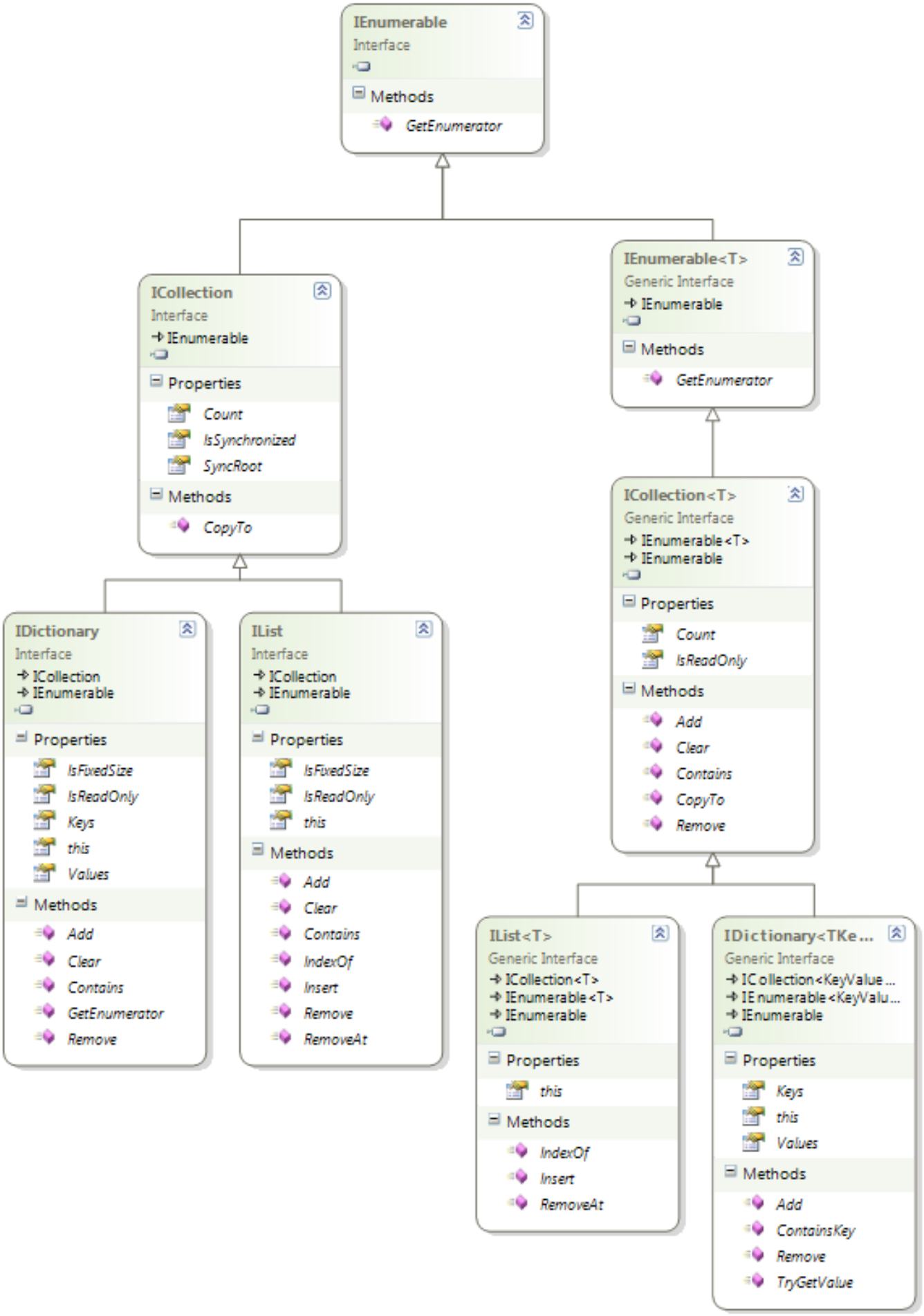
و ...

عنوان: **اشیاء Enumerable و Enumerator و استفاده از قابلیت‌های yield (قسمت اول)**
نویسنده: ابراهیم بیاگوی
تاریخ: ۱۳۹۱/۰۵/۱۷ ۱۸:۲۵
آدرس: www.dotnettips.info
برچسب‌ها: C#, LINQ, Enumerator, Enumerable

در این مقاله می‌خواهیم نحوهٔ ساخت اشیایی با خصوصیات Enumerable را بررسی کنیم. بررسی ویژگی این اشیاء دارای اهمیت است حداقل به این دلیل که پایهٔ یکی از قابلیت مهم زبانی سی‌شارپ یعنی LINQ هستند. برای یافتن پیش‌زمینه‌ای در این موضوع خواندن این مقاله‌های بسیار خوب (۱ و ۲) نیز توصیه می‌شود.

Enumerableها

اشیاء Enumerable یا به‌عبارت دیگر اشیائی که اینترفیس IEnumerable را پیاده‌سازی می‌کنند، دامنهٔ گسترده‌ای از Collectionهای CLI را شامل می‌شوند. همانطور که در نمودار زیر نیز می‌توانید مشاهده کنید IEnumerable (از نوع غیر Generic آن) در بالای سلسله مراتب اینترفیس‌های Collectionهای CLI قرار دارد:


IEnumerable
Interface
Methods
GetEnumerator
ICollection
Interface
IEnumerable
Properties
Count
IsSynchronized
SyncRoot
Methods
CopyTo
IEnumerable<T>
Generic Interface
IEnumerable
Methods
GetEnumerator
ICollection<T>
Generic Interface
IEnumerable<T>
IEnumerable
Properties
Count
IsReadOnly
Methods
Add
Clear
Contains
CopyTo
Remove
IDictionary
Interface
ICollection
IEnumerable
Properties
IsFixedSize
IsReadOnly
Keys
this
Values
Methods
Add
Clear
Contains
GetEnumerator
Remove
IList
Interface
ICollection
IEnumerable
Properties
IsFixedSize
IsReadOnly
this
Methods
Add
Clear
Contains
IndexOf
Insert
Remove
RemoveAt
IList<T>
Generic Interface
ICollection<T>
IEnumerable<T>
IEnumerable
Properties
this
Methods
IndexOf
Insert
RemoveAt
IDictionary<TKe...
Generic Interface
ICollection<KeyValue...
IEnumerable<KeyValu...
IEnumerable
Properties
Keys
this
Values
Methods
Add
ContainsKey
Remove
TryGetValue

درخت اینترفیس‌های Collectionها در CLI منبع

IEnumerableها همچنین دارای اهمیت دیگری نیز هستند؛ قابلیت‌های LINQ که از دات‌نت ۳.۵ به دات‌نت اضافه شدند به‌عنوان Extensionهای این اینترفیس تعریف شده‌اند و پیاده‌سازی Linq to Objects را می‌توانید در کلاس استاتیک System.Linq.Enumerable در System.Core مشاهده کنید. (می‌توانید برای دیدن آن را با ILDasm یا Reflector باز کنید یا پیاده‌سازی آزاد آن در پروژهٔ Mono را اینجا مشاهده کنید که برای شناخت بیشتر LINQ واقعاً مفید است.)

همچنین این Enumerableها هستند که foreach را امکان‌پذیر می‌کنند. به عبارتی دیگر هر شئ‌ای که قرار باشد در foreach (var x in object) قرار بگیرد و بدین طریق اشیاء درونی‌اش را برای پیمایش یا عملی خاص قرار دهد باید Enumerable باشد.

همانطور که قبلاً هم اشاره شد IEnumerable از نوع غیر Generic در بالای نمودار Collectionها قرار دارد و حتی IEnumerable از نوع Generic نیز باید آن را پشتیبانی کند. این موضوع به احتمال به این دلیل در طراحی لحاظ شد که مهاجرت به .NET 2.0 که قابلیت‌های Generic را افزوده بود ساده‌تر کند. IEnumerable همچنین قابلیت covariance که از قابلیت‌های جدید C# 4.0 هست را دارا است (در اصل IEnumerable دارای Generic از نوع out است).

Enumerableها همانطور که از اسم اینترفیس IEnumerable انتظار می‌رود اشیایی هستند که می‌توانند یک شئ Enumerator که IEnumerator را پیاده‌سازی کرده‌است را از خود ارائه دهند. پس طبیعی است برای فهم و درک دلیل وجودی Enumerable باید Enumerator را بررسی کنیم.

Enumeratorها

Enumerator شئ است که در یک پیمایش یا به‌عبارت دیگر گذر از روی تک‌تک عضوها ایجاد می‌شود که با حفظ موقعیت فعلی و پیمایش امکان ادامهٔ پیمایش را برای ما فراهم می‌آورد. اگر بخواهید آن را در حقیقت بازسازی کنید شئ Enumerator به‌مانند کاغذ یا جسمی است که بین صفحات یک کتاب قرار می‌دهید که مکانی که در آن قرار دارید را گم نکنید؛ در این مثال، Enumerable همان کتاب است که قابلیت این را دارد که برای پیمایش به وسیلهٔ قرار دادن یک جسم در وسط آن را دارد.

حال برای اینکه دید بهتری از رابطهٔ بین Enumerable و Enumerator از نظر برنامه‌نویسی به این موضوع پیدا کنیم یک کد نمونهٔ عملی را بررسی می‌کنیم.

در اینجا نمونهٔ ساده و خوانایی از استفاده از یک List برای پیشمایش تمامی اعداد قرار دارد:

```
List<int> list = new List<int>();
list.Add(1);
list.Add(2);
list.Add(3);
foreach (int i in list)
{
    Console.WriteLine(i);
}
```

همانطور که قبلاً اشاره foreach نیاز به یک Enumerable دارد و List هم با پیاده‌سازی IList که گسترشی از IEnumerable هست نیز یک نوع Enumerable هست. اگر این کد را Compile کنیم و IL آن را بررسی کنیم متوجه می‌شویم که CLI در اصل چنین کدی را برای اجرا می‌بینید:

```
List<int> list = new List<int>();
list.Add(1);
list.Add(2);
list.Add(3);
IEnumerator<int> listIterator = list.GetEnumerator();
while (listIterator.MoveNext())
{
    Console.WriteLine(listIterator.Current);
```

```
}
listIterator.Dispose();
```

(می‌توان از using استفاده نمود که Dispose را خود انجام دهد که اینجا برای سادگی استفاده نشده‌است.)

همانطور که می‌بینیم یک Enumerator برای Enumerable ما (یعنی List) ایجاد شد و پس از آن با پرسش این موضوع که آیا این پیمایش امکان ادامه دارد، کل اعضا پیموده‌شده و عمل مورد نظر ما بر آن‌ها انجام شده‌است.

خب، تا اینجای کار با خصوصیات و اهمیت Enumeratorها و Enumerableها آشنا شدیم، حال نوبت به آن می‌رسد که بررسی کنیم آن‌ها را چگونه می‌سازند و بعد از آن با کاربردهای فراتری از آن‌ها نسبت به پیمایش یک List آشنا شویم.

ساخت Enumeratorها و Enumerableها

همانطور که اشاره شد ایجاد اشیاء Enumerable به اشیاء Enumerator مربوط است، پس ما در یک قطعه کد که پیمایش از روی یک آرایه را فراهم می‌آورد ایجاد هر دوی آن‌ها و رابطهٔ بینشان را بررسی می‌کنیم.

```
public class ArrayEnumerable<T> : IEnumerable<T>
    {
        private T[] _array;
        public ArrayEnumerable(T[] array)
        {
            _array = array;
        }


        public IEnumerator<T> GetEnumerator()
        {
            return new ArrayEnumerator<T>(_array);
        }

        System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()
        {
            return GetEnumerator();
        }
    }

    public class ArrayEnumerator<T> : IEnumerator<T>
    {
        private T[] _array;
        public ArrayEnumerator(T[] array)
        {
            _array = array;
        }

        public int index = -1;

        public T Current { get { return _array[index]; } }

        object System.Collections.IEnumerator.Current { get { return this.Current; } }

        public bool MoveNext()
        {
            index++;
            return index < _array.Length;
        }

        public void Reset()
        {
            index = 0;
        }

        public void Dispose() { }
    }
```

ادامه

نظرات خوانندگان

نویسنده: مرتضی
تاریخ: ۱۳۹۱/۰۵/۱۷ ۱۹:۲۰

درخت اینترفیس‌های Collectionها در سی‌شارپ منبع: http://www.mbaldinger.com/post/NET-Collection-Interface-Hierarchy.aspx
بجای سی‌شارپ به دانت‌نت تغییرش بدید
درخت اینترفیس‌های Collectionها در دانت‌نت

نویسنده: ابراهیم بیاگوی
تاریخ: ۱۳۹۱/۰۵/۱۷ ۱۹:۲۹

من شخصاً اطمینان ندارم که همهٔ زبان‌های CLI از همین Collectionها استفاده کنند و البته این نمودار با Syntax سی‌شارپ بود به همین دلیل سی‌شارپ نوشته بودم با این حال آن را به Collectionهای CLI تبدیل کردم.

عنوان: Parallel Programming در Vb.Net
نویسنده: میثم ثوامری
تاریخ: ۱۳۹۱/۰۵/۱۹ ۱۲:۱۴
آدرس: www.dotnettips.info
برچسب‌ها: LINQ, Task Programming Language, Parallel Extensions, VB.NET

حقیقتا تا این لحظه تو پروژه ای استفاده نکردم ولی فکر میکنم یادگیری و استفادش ضروری باشه. ظهورش برمیگرده به .net1 با عنوان Threading. اما کار با Threading خیلی مشکله. من که اینطوری فکر میکنم. حالا با اصلاح کلاس Threading و آمده task خیلی بهتر شده.

گام اول:Threading.Tasks را بعنوان namespace اضافه کنید

یک مثال: این loop در نظر بگیرید

```
Private Sub work()
        While True
        End While
End Sub
```

میخوام برا متد بالا یک task تعریف کنم

```
Task.Factory.StartNew(Sub() work())
```

مثال دوم: یک لیست تعریف میکنم و با استفاد از یک loop میخوام اجزا لیستو چاپ کنم.

```
Dim lst As New List(Of String) From {"meysam", ".nettips", "vahidnasirii"}
Parallel.ForEach(lst, Sub(item) Console.WriteLine("name:{0}", item))
```

مثال سوم: میخوام از این تکنیک تو linq استفاده کنیم:

```
Dim no(9) As Integer
   For i As Integer = 0 To no.Length - 1
        no(i) = i
   Next
Dim result As IEnumerable(Of Double) = no.AsParallel.Select(Function(q) Math.Pow(q,
3)).OrderBy(Function(q) q)
   For Each items In result
        Console.WriteLine(items)
   Next
```

موفق باشید.

نظرات خوانندگان

نویسنده: مرتضی
تاریخ: ۱۳۹۱/۰۵/۱۹ ۱۳:۹

سلام
این کد از لحاظ منطقی درسته و جواب می‌ده ولی کاملا اشتباست چون sub رو بی‌دلیل نوشتی

```
Task.Factory.StartNew(Sub() work())
        'نحوه‌ی صحیح نوشتنش

        Task.Factory.StartNew(AddressOf work)
        '---- یا
        Task.Factory.StartNew(Sub()
                                  While True
                                  End While
                              End Sub)
```

نویسنده: میثم ثوامری
تاریخ: ۱۳۹۱/۰۵/۱۹ ۱۹:۳۵

AddressOf در دستور Threading که قدیمی هست استفاده میشه که عمدتا بصورت:

```
Dim t As New Threading.Thread(AddressOf work)
t.Start()
```

متد Work برای این تعریف شده که مفوم کد برسونه.

نویسنده: مرتضی
تاریخ: ۱۳۹۱/۰۵/۱۹ ۱۹:۴۶

سلام میثم جان اشتباه نکن

AddressOf

ربطی به Thread و یا Task نداره

از AddressOf برای ارجاع به Procedure و Functionها استفاده میشه

نویسنده: میثم ثوامری
تاریخ: ۱۳۹۱/۰۵/۲۰ ۱:۱۲

دوست من منظور من این نبود که AddressOf ارتباطی با Threading داره. منظور من این بود که از زمانی که من Parallel Programmimg کار کردم جایی ندیدم از AddressOf تو دستور Task یا Parallel استفاده کنن.از این دستور تو Thread یا BackgroundWorking استفاده میشد که نسبتا تو نسخه‌های قدیمی net. هستن.

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.

نویسنده: احمدعلی شفیعی
تاریخ: ۱۳۹۳/۰۹/۱۲ ۱۸:۱۴

آیا با این روش ORM هم می‌فهمه که فقط اون اطلاعات رو باید از دیتابیس بگیره؟

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۹/۱۲ ۱۸:۳۶

همیشه میشه خروجی دقیق ORM رو بدون حدس و گمان لاگ و بررسی کرد: نمایش خروجی SQL کدهای Entity framework 6 در کنسول دیباگ ویژوال استودیو

عنوان: **انتخاب پویای فیلد ها در LINQ**
نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۳/۰۳ ۸:۳۰
آدرس: www.dotnettips.info
گروه‌ها: C#, LINQ

LINQ یک DLS بر مبنای .NET می باشد که برای پرس و جو در منابع داده ای مانند پایگاه‌های داده ، فایل‌های XML و یا لیستی از اشیاء درون حافظه کاربرد دارد.

یکی از بزرگترین مزیت‌های آن Syntax آسان و خوانا آن می‌باشد.

LINQ از 2 نوع نمادگذاری پشتیبانی می‌کند:

LINQ Inline یا query expressions :

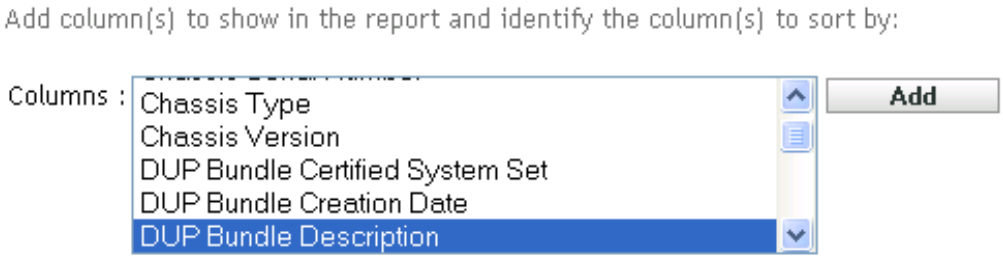
```
var result =
    from product in dbContext.Products
    where product.Category.Name == "Toys"
    where product.Price >= 2.50
    select product.Name;
```

Fluent Syntax :

```
var result = dbContext.Products
    .Where(p => p.Category.Name == "Toys" && p.Price >= 250)
    .Select(p => p.Name);
```

در پرس و جوهای بالا فیلدهای مورد نیاز در قسمت Select در زمان Compile شناخته شده هستند . اما گاهی ممکن است فیلدهای مورد نیاز در زمان اجرا مشخص شوند.

به عنوان مثال یک گزارش ساز پویا که کاربر مشخص می‌کند چه ستون هایی در خروجی نمایش داده شوند یا یک جستجوی پیشرفته که ستون‌های خروجی به اختیار کاربر در زمان اجرا مشخص می‌شوند.



این مدل را در نظر داشته باشید :

```
public class Student
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Field1 { get; set; }
```

```
        public string Field2 { get; set; }
        public string Field3 { get; set; }


        public static IEnumerable<Student> GetStudentSource()
        {
            for (int i = 0; i < 10; i++)
            {
                yield return new Student
                                {
                                    Id = i,
                                    Name = "Name " + i,
                                    Field1 = "Field1 " + i,
                                    Field2 = "Field2 " + i,
                                    Field3 = "Field3 " + i
                                };
            }
        }
    }
```

ستون‌های کلاس Student را در رابط کاربری برنامه جهت انتخاب به کاربر نمایش می‌دهیم. سپس کاربر یک یا چند ستون را انتخاب می‌کند که قسمت Select کوئری برنامه باید بر اساس فیلدهای مورد نظر کاربر مشخص شود.

یکی از روش هایی که می‌توان از آن بهره برد استفاده از کتاب خانه Dynamic LINQ معرفی شده در اینجا می باشد.

این کتابخانه جهت سهولت در نصب به کمک NuGet در این
آدرس قرار دارد.

فرض بر این است که فیلدهای انتخاب شده توسط کاربر با " , " از یکدیگر جدا شده اند.

```
public class Program
    {
        private static void Main(string[] args)
        {
            System.Console.WriteLine("Specify the desired fields : ");
            string fields = System.Console.ReadLine();
            IEnumerable<Student> students = Student.GetStudentSource();
            IQueryable output = students.AsQueryable().Select(string.Format("new({0})", fields));
            foreach (object item in output)
            {
                System.Console.WriteLine(item);
            }

            System.Console.ReadKey();
        }

    }
```

همانطور که در عکس ذیل مشاهده می‌کنید پس از اجرای برنامه ، فیلدهای انتخاب شده توسط کاربر از منبع داده‌ی دریافت شده و در خروجی نمایش داده شده اند.

```
Specify the desired fields :

Field1,Field2,Id,Name

{Field1=Field1 0, Field2=Field2 0, Id=0, Name=Name 0}
{Field1=Field1 1, Field2=Field2 1, Id=1, Name=Name 1}
{Field1=Field1 2, Field2=Field2 2, Id=2, Name=Name 2}
{Field1=Field1 3, Field2=Field2 3, Id=3, Name=Name 3}
{Field1=Field1 4, Field2=Field2 4, Id=4, Name=Name 4}
{Field1=Field1 5, Field2=Field2 5, Id=5, Name=Name 5}
{Field1=Field1 6, Field2=Field2 6, Id=6, Name=Name 6}
{Field1=Field1 7, Field2=Field2 7, Id=7, Name=Name 7}
{Field1=Field1 8, Field2=Field2 8, Id=8, Name=Name 8}
{Field1=Field1 9, Field2=Field2 9, Id=9, Name=Name 9}
```

این روش مزایا و معایب خودش را دارد ، به عنوان مثال خروجی یک لیست از شیء Student نیست یا این Select فقط برای روی یک شیء IQueryable قابل انجام است.

روش دیگری که می‌توان از آن بهره جست استفاده از یک متد کمکی جهت تولید پویای عبارت Lambda ورودی Select می باشد :

```
public  class SelectBuilder <T>
    {
        public static Func<T, T> CreateNewStatement(string fields)
        {
            // input parameter "o"
            var xParameter = Expression.Parameter(typeof(T), "o");


            // new statement "new T()"
            var xNew = Expression.New(typeof(T));

            // create initializers
            var bindings = fields.Split(',').Select(o => o.Trim())
                .Select(o =>
                {

                    // property "Field1"
                    var property = typeof(T).GetProperty(o);

                    // original value "o.Field1"
                    var xOriginal = Expression.Property(xParameter, property);

                    // set value "Field1 = o.Field1"
                    return Expression.Bind(property, xOriginal);
                }
            ).ToList();

            // initialization "new T { Field1 = o.Field1, Field2 = o.Field2 }"
            var xInit = Expression.MemberInit(xNew, bindings);

            // expression "o => new T { Field1 = o.Field1, Field2 = o.Field2 }"
            var lambda = Expression.Lambda<Func<T, T>>(xInit, xParameter);

            // compile to Func<T, T>
            return lambda.Compile();
        }
    }
```

برای استفاده از متد CreateNewStatement باید اینگونه عمل کرد :

```
IEnumerable<Student> result = students.Select(SelectBuilder<Student>.CreateNewStatement("Field1,
Field2")).ToList();

            foreach (Student student in result)
            {
                System.Console.WriteLine(student.Field1);
            }
```

خروجی یک لیست از Student می باشد.

نحوه‌ی کارکرد CreateNewStatement :

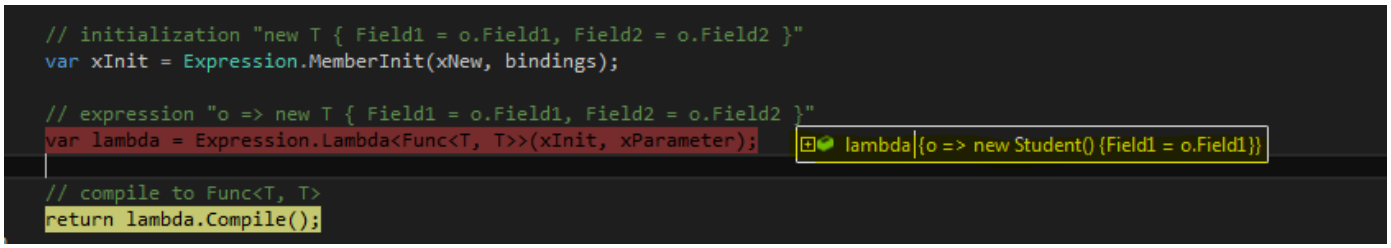
ابتدا فیلدهای انتخابی کاربر که با "," جدا شده اند به ورودی پاس داده می‌شود سپس یک statement خالی ایجاد می‌شود :

```
o=>new Student()
```

فیلدهای ورودی از یکدیگر تفکیک می‌شوند و به کمک Reflection پراپرتی معادل فیلد رشته ای در کلاس Student پیدا می‌شود :

```
var property = typeof(T).GetProperty(o);
```

سپس عبارت Select و تولید شیء جدید بر اساس فیلدهای ورودی تولید می‌شود و برای استفاده Compile به Func می‌شود. در نهایت Func تولید شده به Select پاس داده می‌شود و لیستی از Student بر مبنای فیلدهای انتخابی تولید می‌شود.



دریافت مثال : DynamicSelect.zip

نظرات خوانندگان

نویسنده: sorosh
تاریخ: ۱۳۹۲/۱۱/۱۲ ۷:۴۴

با سلام؛ با ایجاد ستون ردیف با Select new در LINQ مشکل دارم. طوریکه بصورت اتوماتیک یک ستون ردیف ایجاد نماییم:

```
var all = (from x in db.tblZones
select new
{
  RowNuber=???????????
  Code = x.xCode,
  Caption = x.xCaption,
  Comment = x.xComments,
  DT_RowId = "tr" + x.xCode.ToString(),

});
```

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۱/۱۲ ۱۱:۳۲

یک متغیر count قبل از عبارتی که نوشتی ایجاد کن. اینبار جلوی RowNumber بنویس count++.

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۱۱/۱۲ ۱۱:۴۰

متد Select یک Overload دیگر دارد که Index را فراهم می‌کند :

```
string[] weekDays = { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday" };
weekDays.Select((day, index) => new { Day = day, Index = index })
        .Where(x => x.Day.Contains("s"))
        .Select(x => x.Index)
        .ToArray();
```

البته در این کد از Lambda Syntax استفاده شده که برای کد شما هم ممکن است.

عنوان: **نکاتی درباره پرس و جو با استفاده از پردازش موازی**
نویسنده: سیدمجتبی حسینی
تاریخ: ۱۳۹۲/۰۳/۱۰ ۱۷:۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Parallel Processing, PLINQ, LINQ

برای انجام عملیات پرس و جوی LINQ با استفاده از روش پردازش موازی به راحتی میتوان الحاقیه AsParallel را به هر داده‌ای از نوع IEnumerable<T> > افزود:

```
var data =
new int[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
// پرس و جوی عادی
var q1 = from i in data select i;
// پرس و جو به شیوه موازی
var q2 = from i in data.AsParallel() select i;
```

الحاقیه .AsParallel() در پرس و جوی q2 نسخه موازی LINQ را بر روی متغیر data اجرا میکند و اگر همه چیز به صورت صحیح انجام شود هر دو پرس و جو باید نتایج یکسانی داشته باشند، اما نتایج عبارتند از :

```
//نتیجه اجرای q1
0 1 2 3 4 5 6 7 8 9 10
//نتیجه اجرای q2
0 6 1 7 2 8 3 9 4 10 5
```

همانطور که ملاحظه میکنید ترتیب واقعی نتایج اجرای پرس و جوها با یکدیگر متفاوت‌اند و نکته جالبتر آنکه با هر بار اجرای برنامه نتیجه اجرای پرس و جوی q2 با نتیجه سری قبل خودش متفاوت است که این تفاوت به چگونگی تقسیم بندی انجام کار میان هسته‌های سی پی یو، بستگی دارد. نکته بسیار مهم آن است که عملیات پردازش موازی خود را ملزم به حفظ ترتیب داده‌ها نمی‌داند مگر آنکه مجبورش کنیم و این رفتار پردازش موازی به دلیل بالا بردن راندمان عملیات است در نتیجه انجام پرس و جوهای موازی توسط الحاقیه .AsParallel() خیلی هم ساده نیست و ممکن است منجر به تولید نتایج ناخواسته شود.

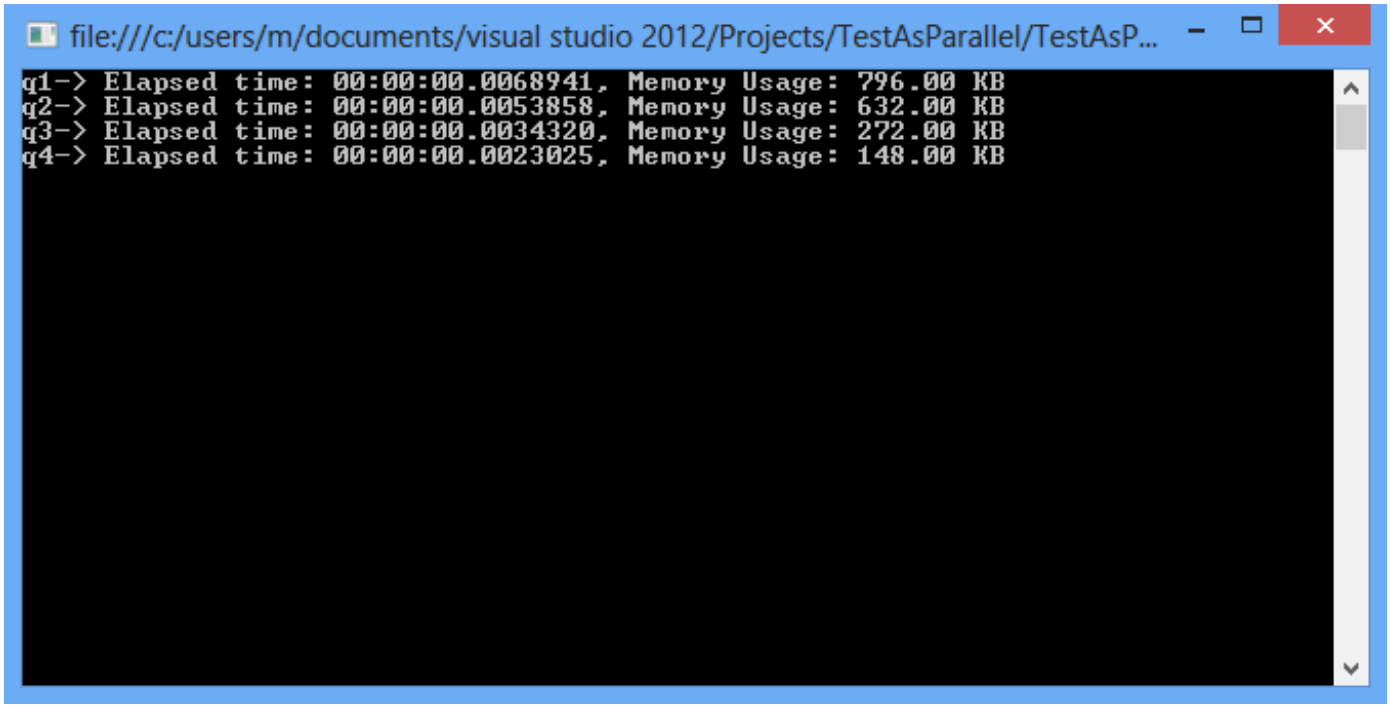
حال اگر چگونگی ترتیب داده‌ها، برایمان مهم است به دو روش می‌توانیم آن را انجام دهیم:

1- افزودن عبارت orderby به پرس و جو

2- استفاده از الحاقیه AsOrdered

```
var q3 = from i in data.AsParallel() orderby i  select i;

var q4 = from i in data.AsParallel().AsOrdered() select i;
```

که نتیجه انجام هر دو پرس و جوی بالا یکی خواهد بود. حال مسأله دیگر این است که آیا همیشه استفاده از پردازش موازی مفید خواهد بود یا خیر؟ پاسخ این سؤال وابسته است به نوع مسأله و حجم داده مورد نظر و مشخصات سیستمی که قرار است از آن کد استفاده کند. چگونگی اندازه سرعت و مقدار مصرف حافظه در اجرای چهار پرس و جوی فوق در کامپیوتر من با پردازنده Intel Q9550 به شکل زیر است:


file:///c:/users/m/documents/visual studio 2012/Projects/TestAsParallel/TestAsP...
q1-> Elapsed time: 00:00:00.0068941, Memory Usage: 796.00 KB
q2-> Elapsed time: 00:00:00.0053858, Memory Usage: 632.00 KB
q3-> Elapsed time: 00:00:00.0034320, Memory Usage: 272.00 KB
q4-> Elapsed time: 00:00:00.0023025, Memory Usage: 148.00 KB

نظرات خوانندگان

نویسنده: رضا
تاریخ: ۱۳۹۳/۰۷/۱۶ ۱۰:۲۹

سلام میشه لطفا دستوری که میزان حافظه و زمان پرس و جوهای بالا را برمی گرداند را در سایت قرار دهید

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۳/۰۷/۱۶ ۱۲:۳۶

کلاس مورد نظر در این مقاله قرار دارد

```
public class PerformanceHelper
    {
        public static string RunActionMeasurePerformance(Action action)
        {
            GC.Collect();
            long initMemUsage = Process.GetCurrentProcess().WorkingSet64;

            var stopwatch = new Stopwatch();
            stopwatch.Start();

            action();

            stopwatch.Stop();

            var currentMemUsage = Process.GetCurrentProcess().WorkingSet64;
            var memUsage = currentMemUsage - initMemUsage;
            if (memUsage < 0) memUsage = 0;

            return string.Format("Elapsed time: {0}, Memory Usage: {1:N2} KB", stopwatch.Elapsed,
memUsage / 1024);
        }
    }
```

عنوان: **سرعت واکشی اطلاعات در List و Dictionary**
نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۳/۱۷ ۱۳:۵
آدرس: www.dotnettips.info
برچسب‌ها: C#, LINQ, .NET, Performance

دسترسی به داده‌ها پیش شرط انجام همه‌ی منطق‌های اکثر نرم افزارهای تجاری می‌باشد. داده‌های ممکن در حافظه ، پایگاه داده ، فایل‌های فیزیکی و هر منبع دیگری قرار گرفته باشند.
هنگامی که حجم داده‌ها کم باشد شاید روش دسترسی و الگوریتم مورد استفاده اهمیتی نداشته باشد اما با افزایش حجم داده‌ها روش‌های بهینه‌تر تاثیر مستقیم در کارایی برنامه دارند.
در این مثال سعی بر این است که در یک سناریوی خاص تفاوت بین Dictionary و List را بررسی کنیم :
فرض کنید 2 کلاس Student و Grade موجود است که وظیفه‌ی نگهداری اطلاعات دانش آموز و نمره را بر عهده دارند.

```
public class Grade
    {
        public Guid StudentId { get; set; }
        public string Value { get; set; }

        public static IEnumerable<Grade> GetData()
        {
            for (int i = 0; i < 10000; i++)
            {
                yield return new Grade
                                {
                                    StudentId = GuidHelper.ListOfIds[i], Value = "Value " + i
                                };
            }
        }
    }

    public class Student
    {
        public Guid Id { get; set; }
        public string Name { get; set; }
        public string Grade { get; set; }

        public static IEnumerable<Student> GetStudents()
        {
            for (int i = 0; i < 10000; i++)
            {
                yield return new Student
                                {
                                    Id = GuidHelper.ListOfIds[i],
                                    Name = "Name " + i
                                };
            }
        }
    }
```

از کلاس GuidHelper برای تولید و نگهداری شناسه‌های یکتا برای دانش آموز کمک گرفته شده است :

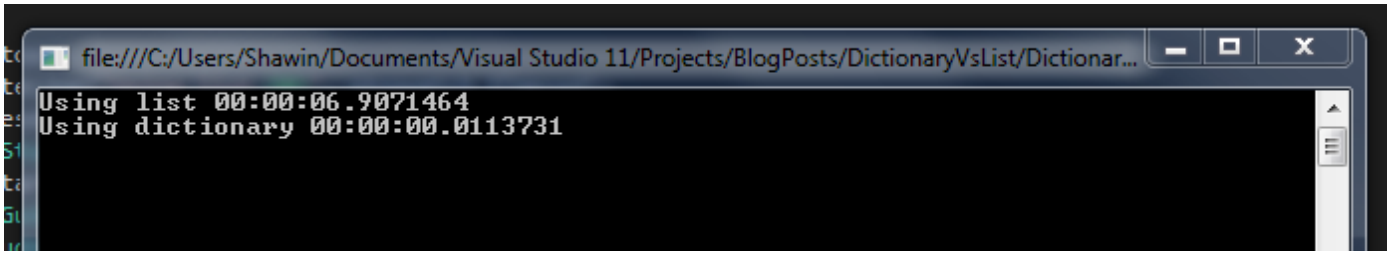
```
public class GuidHelper
    {
        public static List<Guid> ListOfIds=new List<Guid>();

        static GuidHelper()
        {
            for (int i = 0; i < 10000; i++)
            {
                ListOfIds.Add(Guid.NewGuid());
            }
        }
    }
```

سپس لیستی از دانش آموزان و نمرات را درون حافظه ایجاد کرده و با یک حلقه نمره‌ی هر دانش آموز به Property مورد نظر مقدار داده می‌شود.

ابتدا از LINQ روی لیست برای پیدا کردن نمره‌ی مورد نظر استفاده کرده و در روش دوم برای پیدا کردن نمره‌ی هر دانش آموز از Dictionary استفاده شده :

```
internal class Program
    {
        private static void Main(string[] args)
        {
            var stopwatch = new Stopwatch();
            List<Grade> grades = Grade.GetData().ToList();
            List<Student> students = Student.GetStudents().ToList();

            stopwatch.Start();
            foreach (Student student in students)
            {
                student.Grade = grades.Single(x => x.StudentId == student.Id).Value;
            }
            stopwatch.Stop();
            Console.WriteLine("Using list {0}", stopwatch.Elapsed);
            stopwatch.Reset();
            students = Student.GetStudents().ToList();
            stopwatch.Start();
            Dictionary<Guid, string> dictionary = Grade.GetData().ToDictionary(x => x.StudentId, x =>
x.Value);

            foreach (Student student in students)
            {
                student.Grade = dictionary[student.Id];
            }
            stopwatch.Stop();
            Console.WriteLine("Using dictionary {0}", stopwatch.Elapsed);
            Console.ReadKey();
        }
    }
```

نتیجه‌ی مقایسه در سیستم من اینگونه می‌باشد :



همانگونه که مشاهده می‌شود در این سناریو خواندن نمره از روی Dictionary بر اساس 'کلید' بسیار سریع‌تر از انجام یک پرس و جوی LINQ روی لیست است.

زمانی که از LINQ on list

```
student.Grade = grades.Single(x => x.StudentId == student.Id).Value;
```

برای پیدا کردن مقدار مورد نظر یک به یک روی اعضا لیست حرکت می‌کند تا به مقدار مورد نظر برسد در نتیجه پیچیدگی زمانی آن O n هست. پس هر چه میزان داده‌ها بیشتر باشد این روش کندتر می‌شود.

زمانی که از Dictonary

```
student.Grade = dictionary[student.Id];
```

برای پیدا کردن مقدار استفاده می‌شود با اولین تلاش مقدار مورد نظر یافت می‌شود پس پیچیدگی زمانی آن 1 O می‌باشد.

در نتیجه اگر نیاز به پیدا کردن اطلاعات بر اساس یک مقدار یکتا یا کلید باشد تبدیل اطلاعات به Dictionary و خواندن از آن بسیار به صرفه‌تر است.

تفاوت این 2 روش وقتی مشخص می‌شود که میزان داده‌ها زیاد باشد.

در همین رابطه (1 ، 2)

DictionaryVsList.zip

نظرات خوانندگان

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۲/۰۳/۱۷ ۲۱:۳۵

یه نگاهی هم به این بندازید. جالبه: http://stackoverflow.com/questions/1009107/what-net-collection-provides-the-fastest-search

نویسنده: مهدی فرزاد
تاریخ: ۱۳۹۲/۰۳/۱۸ ۰:۲

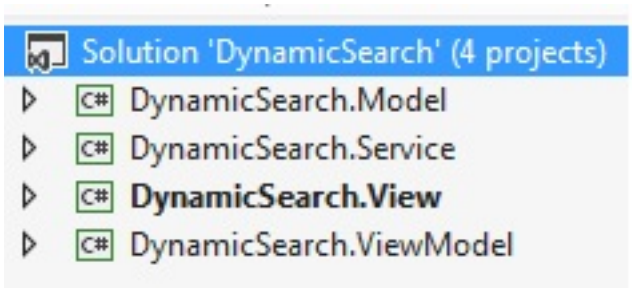
با تشکر از دوست خوبم ، یک سئوال مطرح میشه شما این نتیجه رو از روی داده‌های موجود در حافظه انجام دادید ، اگر این داده‌ها در دیتا بیس باشه و با استفاده از یک ORM مثل EF به داده‌ها دسترسی داشته باشیم برای استفاده از Dictionary ابتدا تمام داده‌ها یک بار واکشی شده و در نتیجه جستجو میشه؟ آیا این مطلب درسته؟ اگر آره پس نتیجه به نفع Linq تغییر میکنه

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۳/۱۸ ۰:۲۴

نه. ToList یا ToDictionary اصطلاحا یک نوع Projection هستند و پس از دریافت اطلاعات مطابق کوئری لینک شما اعمال خواهند شد (شکل دادن به اطلاعات دریافت شده از بانک اطلاعاتی؛ فرضا 100 رکورد دریافت شده، حالا شما خواستید از این رکوردها برای استفاده، List درست کنید یا دیکشنری یا حالت‌های دیگر).

عنوان: **ایجاد فرم جستجوی پویا با استفاده از Expression ها**
نویسنده: محمد عیدی مراد
تاریخ: ۱۳۹۲/۰۳/۲۵ ۹:۲۵
آدرس: www.dotnettips.info
برچسب‌ها: LINQ, MVVM, Expression, DynamicLINQ

در مواردی نیاز است کاربر را جهت انتخاب فیلدهای مورد جستجو آزاد نگه داریم. برای نمونه جستجویی را در نظر بگیرید که کاربر قصد دارد: "دانش آموزانی که نام آنها برابر علی است و شماره دانش آموزی آنها از 100 کمتر است" را پیدا کند در شرایطی که فیلدهای نام و شماره دانش آموزی و عمل گر کوچک‌تر را خود کاربر به دلخواه برگزیرده.
روش‌های زیادی برای پیاده سازی این نوع جستجوها وجود دارد. در این مقاله سعی شده گام‌های ایجاد یک ساختار پایه برای این نوع فرم‌ها و یک ایجاد فرم نمونه بر پایه ساختار ایجاد شده را با استفاده از یکی از همین روش‌ها شرح دهیم.
اساس این روش تولید عبارت Linq بصورت پویا با توجه به انتخاب‌های کاربرمی باشد.
1- برای شروع یک سلوشن خالی با نام DynamicSearch ایجاد می‌کنیم. سپس ساختار این سلوشن را بصورت زیر شکل می‌دهیم.



در این مثال پیاده سازی در قالب ساختار MVVM در نظر گرفته شده. ولی محدودتی از این نظر برای این روش قائل نیستیم.
2- کار را از پروژه مدل آغاز می‌کنیم. جایی که ما برای سادگی کار، 3 کلاس بسیار ساده را به ترتیب زیر ایجاد می‌کنیم:

```
namespace DynamicSearch.Model
{
    public class Person
    {
        public Person(string name, string family, string fatherName)
        {
            Name = name;
            Family = family;
            FatherName = fatherName;
        }

        public string Name { get; set; }
        public string Family { get; set; }
        public string FatherName { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DynamicSearch.Model
{
    public class Teacher : Person
    {
        public Teacher(int id, string name, string family, string fatherName)
            : base(name, family, fatherName)
        {
            ID = id;
        }

        public int ID { get; set; }

        public override string ToString()
```

```
        {
            return string.Format("Name: {0}, Family: {1}", Name, Family);
        }
    }
}

namespace DynamicSearch.Model
{
    public class Student : Person
    {
        public Student(int stdId, Teacher teacher, string name, string family, string fatherName)
            : base(name, family, fatherName)
        {
            StdID = stdId;
            Teacher = teacher;
        }

        public int StdID { get; set; }
        public Teacher Teacher { get; set; }
    }
}
```

-3 در پروژه سرویس یک کلاس بصورت زیر ایجاد می‌کنیم:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using DynamicSearch.Model;

namespace DynamicSearch.Service
{
    public class StudentService
    {
        public IList<Student> GetStudents()
        {
            return new List<Student>
                {
                    new Student(1,new Teacher(1,"Ali","Rajabi","Reza"),"Mohammad","Hoeyni","Sadegh"),
                    new Student(2,new Teacher(2,"Hasan","Noori","Mohsen"),"Omid","Razavi","Ahmad"),
                };
        }
    }
}
```

-4 تا اینجا تمامی داده‌ها صرفا برای نمونه بود. در این مرحله ساخت اساس جستجو گر پویا را شرح می‌دهیم.

جهت ساخت عبارت، نیاز به سه نوع جزء داریم:

-اتصال دهنده عبارات ("و" ، "یا")

-عملوند (در اینجا فیلدی که قصد مقایسه با عبارت مورد جستجوی کاربر را داریم)

-عملگر (">" ، "<" ، "=" ، ....)

برای ذخیره المان‌های انتخاب شده توسط کاربر، سه کلاس زیر را ایجاد می‌کنیم (همان سه جزء بالا):

```
using System;
using System.Linq.Expressions;

namespace DynamicSearch.ViewModel.Base
{
    public class AndOr
    {
        public AndOr(string name, string title,Func<Expression,Expression,Expression> func)
        {
            Title = title;
            Func = func;
            Name = name;
        }

        public string Title { get; set; }
        public Func<Expression, Expression, Expression> Func { get; set; }
        public string Name { get; set; }
```

```
    }
}

using System;

namespace DynamicSearch.ViewModel.Base
{
    public class Feild : IEquatable<Feild>
    {
        public Feild(string title, Type type, string name)
        {
            Title = title;
            Type = type;
            Name = name;
        }

        public Type Type { get; set; }
        public string Name { get; set; }
        public string Title { get; set; }
        public bool Equals(Feild other)
        {
            return other.Title == Title;
        }
    }
}

using System;
using System.Linq.Expressions;

namespace DynamicSearch.ViewModel.Base
{
    public class Operator
    {
        public enum TypesToApply
        {
            String,
            Numeric,
            Both
        }

        public Operator(string title, Func<Expression, Expression, Expression> func, TypesToApply
typeToApply)
        {
            Title = title;
            Func = func;
            TypeToApply = typeToApply;
        }

        public string Title { get; set; }
        public Func<Expression, Expression, Expression> Func { get; set; }
        public TypesToApply TypeToApply { get; set; }
    }
}
```

توسط کلاس زیر یک سری اعمال متداول را پیاده سازی کرده ایم و پیاده سازی اضافات را بعهده کلاس‌های ارث برنده از این کلاس گذاشته ایم:

```
using System.Collections.ObjectModel;
using System.Linq;
using System.Linq.Expressions;

namespace DynamicSearch.ViewModel.Base
{
    public abstract class SearchFilterBase<T> : BaseViewModel
    {
        protected SearchFilterBase()
        {
            var containOp = new Operator("شامل باشد", (expression, expression1) =>
Expression.Call(expression, typeof(string).GetMethod("Contains"), expression1),
Operator.TypesToApply.String);
            var notContainOp = new Operator("شامل نباشد", (expression, expression1) =>
            {
                var contain = Expression.Call(expression, typeof(string).GetMethod("Contains"),
expression1);
                return Expression.Not(contain);
            }, Operator.TypesToApply.String);
```

```
            var equalOp = new Operator("=", Expression.Equal, Operator.TypesToApply.Both);
            var notEqualOp = new Operator("<>", Expression.NotEqual, Operator.TypesToApply.Both);
            var lessThanOp = new Operator("<", Expression.LessThan, Operator.TypesToApply.Numeric);
            var greaterThanOp = new Operator(">", Expression.GreaterThan,
Operator.TypesToApply.Numeric);
            var lessThanOrEqual = new Operator("<=", Expression.LessThanOrEqual,
Operator.TypesToApply.Numeric);
            var greaterThanOrEqual = new Operator(">=", Expression.GreaterThanOrEqual,
Operator.TypesToApply.Numeric);

            Operators = new ObservableCollection<Operator>
                {
                      equalOp,
                      notEqualOp,
                      containOp,
                      notContainOp,
                      lessThanOp,
                      greaterThanOp,
                      lessThanOrEqual,
                      greaterThanOrEqual,
                };


            SelectedAndOr = AndOrs.FirstOrDefault(a => a.Name == "Suppress");
            SelectedFeild = Feilds.FirstOrDefault();
            SelectedOperator = Operators.FirstOrDefault(a => a.Title == "=");
        }

        public abstract IQueryable<T> GetQuarable();

        public virtual ObservableCollection<AndOr> AndOrs
        {
            get
            {
                return new ObservableCollection<AndOr>
                    {
                        new AndOr("And","و", Expression.AndAlso),
                        new AndOr("Or","یا",Expression.OrElse),
                        new AndOr("Suppress","نادیده",(expression, expression1) => expression),
                    };
            }
        }
        public virtual ObservableCollection<Operator> Operators
        {
            get { return _operators; }
            set { _operators = value; NotifyPropertyChanged("Operators"); }
        }
        public abstract ObservableCollection<Feild> Feilds { get; }

        public bool IsOtherFilters
        {
            get { return _isOtherFilters; }
            set { _isOtherFilters = value; }
        }
        public string SearchValue
        {
            get { return _searchValue; }
            set { _searchValue = value; NotifyPropertyChanged("SearchValue"); }
        }
        public AndOr SelectedAndOr
        {
            get { return _selectedAndOr; }
            set { _selectedAndOr = value; NotifyPropertyChanged("SelectedAndOr");
NotifyPropertyChanged("SelectedFeildHasSetted"); }
        }
        public Operator SelectedOperator
        {
            get { return _selectedOperator; }
            set { _selectedOperator = value; NotifyPropertyChanged("SelectedOperator"); }
        }
        public Feild SelectedFeild
        {
            get { return _selectedFeild; }
            set
            {
                Operators = value.Type == typeof(string) ? new
ObservableCollection<Operator>(Operators.Where(a => a.TypeToApply == Operator.TypesToApply.Both ||
a.TypeToApply == Operator.TypesToApply.String)) : new ObservableCollection<Operator>(Operators.Where(a
=> a.TypeToApply == Operator.TypesToApply.Both || a.TypeToApply == Operator.TypesToApply.Numeric));
                if (SelectedOperator == null)
                {
```

```
                    SelectedOperator = Operators.FirstOrDefault(a => a.Title == "=");
                }

                NotifyPropertyChanged("SelectedOperator");
                NotifyPropertyChanged("SelectedFeild");
                _selectedFeild = value;
                NotifyPropertyChanged("SelectedFeildHasSetted");
            }
        }
        public bool SelectedFeildHasSetted
        {
            get
            {
                return SelectedFeild != null &&
                       (SelectedAndOr.Name != "Suppress" || !IsOtherFilters);
            }
        }

        private ObservableCollection<Operator> _operators;
        private Feild _selectedFeild;
        private Operator _selectedOperator;
        private AndOr _selectedAndOr;
        private string _searchValue;
        private bool _isOtherFilters = true;
    }
}
```

توضیحات: در این ویو مدل پایه سه لیست تعریف شده که برای دو تای آنها پیاده سازی پیش فرضی در همین کلاس دیده شده ولی برای لیست فیلدها پیاده سازی به کلاس ارث برنده واگذار شده است.

در گام بعد، یک کلاس کمکی برای سهولت ساخت عبارات ایجاد می‌کنیم:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Reflection;
using AutoMapper;

namespace DynamicSearch.ViewModel.Base
{
  public static  class ExpressionExtensions
    {
        public static List<T> CreateQuery<T>(Expression whereCallExpression, IQueryable entities)
        {
            return entities.Provider.CreateQuery<T>(whereCallExpression).ToList();
        }

        public static MethodCallExpression CreateWhereCall<T>(Expression condition, ParameterExpression
pe, IQueryable entities)
        {
            var whereCallExpression = Expression.Call(
                typeof(Queryable),
                "Where",
                new[] { entities.ElementType },
                entities.Expression,
                Expression.Lambda<Func<T, bool>>(condition, new[] { pe }));
            return whereCallExpression;
        }

        public static void CreateLeftAndRightExpression<T>(string propertyName, Type type, string
searchValue, ParameterExpression pe, out Expression left, out Expression right)
        {
            var typeOfNullable = type;
            typeOfNullable = typeOfNullable.IsNullableType() ? typeOfNullable.GetTypeOfNullable() :
typeOfNullable;
            left = null;

            var typeMethodInfos = typeOfNullable.GetMethods();
            var parseMethodInfo = typeMethodInfos.FirstOrDefault(a => a.Name == "Parse" &&
a.GetParameters().Count() == 1);

            var propertyInfos = typeof(T).GetProperties();
            if (propertyName.Contains("."))
```

```
            {
                left = CreateComplexTypeExpression(propertyName, propertyInfos, pe);
            }
            else
            {
                var propertyInfo = propertyInfos.FirstOrDefault(a => a.Name == propertyName);
                if (propertyInfo != null) left = Expression.Property(pe, propertyInfo);
            }

            if (left != null) left = Expression.Convert(left, typeOfNullable);

            if (parseMethodInfo != null)
            {
                var invoke = parseMethodInfo.Invoke(searchValue, new object[] { searchValue });
                right = Expression.Constant(invoke, typeOfNullable);
            }
            else
            {
                //type is string
                right = Expression.Constant(searchValue.ToLower());
                var methods = typeof(string).GetMethods();
                var firstOrDefault = methods.FirstOrDefault(a => a.Name == "ToLower" &&
!a.GetParameters().Any());
                if (firstOrDefault != null) left = Expression.Call(left, firstOrDefault);
            }
        }

        public static Expression CreateComplexTypeExpression(string searchFilter,
IEnumerable<PropertyInfo> propertyInfos, Expression pe)
        {
            Expression ex = null;
            var infos = searchFilter.Split('.');
            var enumerable = propertyInfos.ToList();
            for (var index = 0; index < infos.Length - 1; index++)
            {
                var propertyInfo = infos[index];
                var nextPropertyInfo = infos[index + 1];
                if (propertyInfos == null) continue;
                var propertyInfo2 = enumerable.FirstOrDefault(a => a.Name == propertyInfo);
                if (propertyInfo2 == null) continue;
                var val = Expression.Property(pe, propertyInfo2);
                var propertyInfos3 = propertyInfo2.PropertyType.GetProperties();
                var propertyInfo3 = propertyInfos3.FirstOrDefault(a => a.Name == nextPropertyInfo);
                if (propertyInfo3 != null) ex = Expression.Property(val, propertyInfo3);
            }

            return ex;
        }

        public static Expression AddOperatorExpression(Func<Expression, Expression, Expression> func,
Expression left, Expression right)
        {
            return func.Invoke(left, right);
        }

        public static Expression JoinExpressions(bool isFirst, Func<Expression, Expression, Expression>
func, Expression expression, Expression ex)
        {
            if (!isFirst)
            {
                return func.Invoke(expression, ex);
            }

            expression = ex;
            return expression;
        }
    }
}
```

-5 ایجاد کلاس فیلتر جهت معرفی فیلدها و معرفی منبع داده و ویو مدلی ارث برنده از کلاس‌های پایه ساختار، جهت ایجاد فرم نمونه:

```
using System.Collections.ObjectModel;
using System.Linq;
using DynamicSearch.Model;
using DynamicSearch.Service;
```

```
using DynamicSearch.ViewModel.Base;

namespace DynamicSearch.ViewModel
{
    public class StudentSearchFilter : SearchFilterBase<Student>
    {
        public override ObservableCollection<Feild> Feilds
        {
            get
            {
                return new ObservableCollection<Feild>
                    {
                        new Feild("نام دانش آموز",typeof(string),"Name"),
                         new Feild("نام خانوادگی دانش آموز",typeof(string),"Family"),
                        new Feild("نام خانوادگی معلم",typeof(string),"Teacher.Name"),
                        new Feild("شماره دانش آموزی",typeof(int),"StdID"),
                    };
            }
        }

        public override IQueryable<Student> GetQuarable()
        {
            return new StudentService().GetStudents().AsQueryable();
        }
    }
}
```

-6 ایجاد ویو نمونه:

در نهایت زمل فایل موجود در پروژه ویو:

```
<Window x:Class="DynamicSearch.View.MainWindow"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:viewModel="clr-namespace:DynamicSearch.ViewModel;assembly=DynamicSearch.ViewModel"
        xmlns:view="clr-namespace:DynamicSearch.View"
        mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
    <Window.Resources>
        <viewModel:StudentSearchViewModel x:Key="StudentSearchViewModel" />
        <view:VisibilityConverter x:Key="VisibilityConverter" />
    </Window.Resources>
    <Grid   DataContext="{StaticResource StudentSearchViewModel}">
        <WrapPanel Orientation="Vertical">
            <DataGrid AutoGenerateColumns="False" Name="asd" CanUserAddRows="False"
ItemsSource="{Binding BindFilter}">
                <DataGrid.Columns>
                    <DataGridTemplateColumn>
                        <DataGridTemplateColumn.CellTemplate>
                            <DataTemplate DataType="{x:Type viewModel:StudentSearchFilter}">
                                <ComboBox MinWidth="100"  DisplayMemberPath="Title"
ItemsSource="{Binding AndOrs}" Visibility="{Binding IsOtherFilters,Converter={StaticResource
VisibilityConverter}}"
                                          SelectedItem="{Binding
SelectedAndOr,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}" />
                            </DataTemplate>
                        </DataGridTemplateColumn.CellTemplate>
                    </DataGridTemplateColumn>
                    <DataGridTemplateColumn >
                        <DataGridTemplateColumn.CellTemplate>
                            <DataTemplate DataType="{x:Type viewModel:StudentSearchFilter}">
                                <ComboBox IsEnabled="{Binding SelectedFeildHasSetted}" MinWidth="100"
DisplayMemberPath="Title" ItemsSource="{Binding Feilds}" SelectedItem="{Binding
SelectedFeild,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged }"/>
                            </DataTemplate>
                        </DataGridTemplateColumn.CellTemplate>
                    </DataGridTemplateColumn>
                    <DataGridTemplateColumn>
                        <DataGridTemplateColumn.CellTemplate>
                            <DataTemplate DataType="{x:Type viewModel:StudentSearchFilter}">
                                <ComboBox MinWidth="100"  DisplayMemberPath="Title"
ItemsSource="{Binding Operators}" IsEnabled="{Binding SelectedFeildHasSetted}"
                                          SelectedItem="{Binding
```

```
SelectedOperator,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}" />
                            </DataTemplate>
                        </DataGridTemplateColumn.CellTemplate>
                    </DataGridTemplateColumn>
                    <DataGridTemplateColumn Width="*">
                        <DataGridTemplateColumn.CellTemplate>
                            <DataTemplate DataType="{x:Type viewModel:StudentSearchFilter}">
                                <TextBox IsEnabled="{Binding SelectedFeildHasSetted}" MinWidth="200"
Text="{Binding SearchValue,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>
                                <!--<TextBox Text="{Binding
SearchValue,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>-->
                            </DataTemplate>
                        </DataGridTemplateColumn.CellTemplate>
                    </DataGridTemplateColumn>
                </DataGrid.Columns>
            </DataGrid>

            <Button Content="+" HorizontalAlignment="Left" Command="{Binding AddFilter}"/>
            <Button Content="Result" Command="{Binding ExecuteSearchFilter}"/>
            <DataGrid ItemsSource="{Binding Results}">

            </DataGrid>
        </WrapPanel>
    </Grid>
</Window>
```

در این مقاله، هدف معرفی روند ایجاد یک جستجو گر پویا با قابلیت استفاده مجدد بالا بود و عمدا از توضیح جزء به جزء کدها صرف نظر شده. علت این امر وجود منابع بسیار راجب ابزارهای بکار رفته در این مقاله و سادگی کدهای نوشته شده توسط اینجانب می‌باشد.

برخی منابع جهت آشنایی با Expression ها:

http://msdn.microsoft.com/en-us/library/bb882637.aspx

انتخاب پویای فیلدها در LINQ

http://www.persiadevelopers.com/articles/dynamiclinqquery.aspx

نکته: کدهای نوشته شده در این مقاله، نسخه‌های نخستین هستند و طبیعتا جا برای بهبود بسیار دارند. دوستان می‌توانند در این امر به بنده کمک کنند.

پیشنهادات جهت بهبود:
- جداسازی کدهای پیاده کننده منطق از ویو مدل‌ها جهت افزایش قابلیت نگهداری کد و سهولت استفاده در سایر ساختارها
- افزودن توضیحات به کد
- انتخاب نامگذاری‌های مناسب تر

DynamicSearch.zip

عنوان: Func یا Expression Func در EF
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۲۶ ۸:۰
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework, LINQ

با بررسی کدهای مختلف Entity framework گاهی از اوقات در امضای توابع کمکی نوشته شده، Func<> مشاهده می‌شود و در بعضی از موارد Expression<Func<>> و ... به نظر استفاده کنندگان دقیقا نمی‌دانند که تفاوت این دو در چیست و کدامیک را باید/یا بهتر است بکار برد.

ابتدا مثال کامل ذیل را در نظر بگیرید:

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Data.Entity.Migrations;
using System.Linq;
using System.Linq.Expressions;

namespace Sample
{
    public abstract class BaseEntity
    {
        public int Id { set; get; }
    }

    public class Receipt : BaseEntity
    {
        public int TotalPrice { set; get; }
    }

    public class MyContext : DbContext
    {
        public DbSet<Receipt> Receipts { get; set; }
    }

    public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(MyContext context)
        {
            if (!context.Receipts.Any())
            {
                for (int i = 0; i < 20; i++)
                {
                    context.Receipts.Add(new Receipt { TotalPrice = i });
                }
            }
            base.Seed(context);
        }
    }

    public static class EFUtils
    {
        public static IList<T> LoadEntities<T>(this DbContext ctx, Expression<Func<T, bool>> predicate)
where T : class
        {
            return ctx.Set<T>().Where(predicate).ToList();
        }

        public static IList<T> LoadData<T>(this DbContext ctx, Func<T, bool> predicate) where T : class
        {
            return ctx.Set<T>().Where(predicate).ToList();
        }
    }

    public static class Test
    {
        public static void RunTests()
        {
```

```
            startDB();

            using (var context = new MyContext())
            {
                var list1 = context.LoadEntities<Receipt>(x => x.TotalPrice == 10);
                var list2 = context.LoadData<Receipt>(x => x.TotalPrice == 20);
            }
        }

        private static void startDB()
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyContext, Configuration>());
            // Forces initialization of database on model changes.
            using (var context = new MyContext())
            {
                context.Database.Initialize(force: true);
            }
        }
    }
}
```

در این مثال ابتدا کلاس Receipt تعریف شده و سپس توسط کلاس MyContext در معرض دید EF قرار گرفته است. در ادامه توسط کلاس Configuration نحوه آغاز بانک اطلاعاتی مشخص گردیده است؛ به همراه ثبت تعدادی رکورد نمونه.
نکته اصلی مورد بحث، کلاس کمکی EFUtils است که در آن دو متد الحاقی LoadEntities و LoadData تعریف شده‌اند. در متد LoadEntities، امضای متد شامل Expression Func است و در متد LoadData فقط Func ذکر شده است.
در ادامه اگر برنامه را توسط فراخوانی متد RunTests اجرا کنیم، به نظر شما خروجی SQL حاصل از list1 و list2 چیست؟
احتمالا شاید عنوان کنید که هر دو یک خروجی SQL دارند (با توجه به اینکه بدنه متدهای LoadEntities و LoadData دقیقا/یا به نظر یکی هستند) اما یکی از پارامتر 10 استفاده می‌کند و دیگری از پارامتر 20. تفاوت دیگری ندارند.
اما ... اینطور نیست!
خروجی SQL متد LoadEntities در متد RunTests به صورت زیر است:

```
SELECT
[Extent1].[Id] AS [Id],
[Extent1].[TotalPrice] AS [TotalPrice]
FROM [dbo].[Receipts] AS [Extent1]
WHERE 10 = [Extent1].[TotalPrice]
```

و ... خروجی متد LoadData به نحو زیر:

```
SELECT
[Extent1].[Id] AS [Id],
[Extent1].[TotalPrice] AS [TotalPrice]
FROM [dbo].[Receipts] AS [Extent1]
```

بله. در لیست دوم هیچ فیلتری انجام نشده (در حالت استفاده از Func خالی) و کل اطلاعات موجود در جدول Receipts، بازگشت داده شده است.
چرا؟
Func اشاره‌گری است به یک متد و Expression Func بیانگر ساختار درختی عبارت lambda نوشته شده است. این ساختار درختی صرفا بیان می‌کند که عبارت lambda منتسب، چه کاری را قرار است یا می‌تواند انجام دهد؛ بجای انجام واقعی آن.

```
public static IQueryable<TSource> Where<TSource>(this IQueryable<TSource> source,
Expression<Func<TSource, bool>> predicate);
public static IEnumerable<TSource> Where<TSource>(this IEnumerable<TSource> source, Func<TSource, bool>
predicate);
```

اگر از Expression Func استفاده شود، از متد Where ایی استفاده خواهد شد که خروجی IQueryable دارد. اگر از Func استفاده شود، از overload دیگری که خروجی و ورودی IEnumerable دارد به صورت خودکار استفاده می‌گردد.
بنابراین هرچند بدنه دو متد LoadEntities و LoadData به ظاهر یکی هستند، اما بر اساس نوع ورودی Where ایی که دریافت می‌کنند، اگر Expression Func باشد، EF فرصت آنالیز و ترجمه عبارت ورودی را خواهد یافت اما اگر Func باشد، ابتدا باید کل

اطلاعات را به صورت یک لیست IEnumerable دریافت و سپس سمت کلاینت، خروجی نهایی را فیلتر کند.
اگر برنامه را اجرا کنید نهایتا هر دو لیست یک و دو، بر اساس شرط عنوان شده عمل خواهند کرد و فیلتر خواهند شد. اما در حالت اول این فیلتر شدن سمت بانک اطلاعاتی است و در حالت دوم کل اطلاعات بارگذاری شده و سپس سمت کاربر فیلتر می‌شود (که کارآیی پایینی دارد).

نتیجه گیری

به امضای متد Where ایی که در حال استفاده است دقت کنید. همینطور در مورد Count ، Sum و یا موارد مشابه دیگری که predicate قبول می‌کنند.

نظرات خوانندگان

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۲/۰۴/۲۶ ۲۰:۱

اوایل که از Entity Framework استفاده میکردم دچار همین مشکل شدم. در یک برنامه تمام متدهای دارای شرط لایه سرویسم رو با استفاده از Func پیاده سازی کرده بودم و بعد از مدتی متوجه شدم که برای دریافت یک رکورد از جدول هم برنامه خیلی کند عمل میکنه. کد زیر رو نگاه کنید:

```
public virtual TEntity Find(Func<TEntity, bool> predicate)
{
    return _tEntities.Where(predicate).FirstOrDefault();
}
```

در این حالت همه رکوردها از جدول مورد نظر واکشی میشه و بعد فقط یکی از آنها (اولین رکورد) در سمت کلاینت جدا و بازگشت داده میشه.

نویسنده: Saleh
تاریخ: ۱۳۹۲/۰۵/۱۰ ۱۰:۴۲

با تشکر از شما
جهت اطلاع دوستان:
کتاب Functional Programming In CS نوشته Oliver Sturm به طور کامل مبحث Genericها را پوشش داده و موضوعات Func و Expressionها را مفصل تشریح کرده است.

نویسنده: Saleh
تاریخ: ۱۳۹۲/۰۵/۱۰ ۱۲:۰

استفاده از Predicate<> چه تفاوتی با استفاده شما از Func دارد؟

حتی آقای نصیری هم به همین صورت استفاده کرده اند.

```
public virtual TEntity Find(Expression<Func<TEntity, bool>> predicate)
```

```
public virtual TEntity Find(Expression<Predicate<TEntity>> predicate)
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۱۰ ۱۲:۳۸

استفاده نشده. فرق است بین یک پارامتر با *نام* predicate و یک delegate به نام Predicate . ضمنا Func هم یک Delegate است .

عنوان: **استفاده از OfType یا Cast در Linq**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۴/۲۸ ۲۰:۳۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ, Generics, Casting, Collections

تقریبا تمام توسعه دهندگان دات نت با تکنولوژی Linq و Lambda Expressionها آشنایی دارند. همان طور که می‌دانیم Extension Methodهای موجود در فضای نام System.Linq فقط بر روی مجموعه ای از داده‌ها که اینترفیس IEnumerable<t> که در فضای نام System.Collections.Generic قرار دارد را پیاده سازی کرده باشند قابل اجرا هستند. مجموعه داده‌های جنریک فقط قابلیت نگهداری از یک نوع داده که به عنوان پارامتر T برای این مجموعه تعریف می‌شود را داراست.
نکته: البته در مجموعه هایی نظیر Dictionary یا سایر Collectionها امکان تعریف چند نوع داده به عنوان پارامتر وجود دارد. نکته مهم این است که داده‌های استفاده شده در این مجموعه ها، حتما باید از نوع پارامتر تعریف شده باشند.
اگر در یک مجموعه داده قصد داشته باشیم که داده هایی با نوع مختلف را ذخیره کنیم و در جای مناسب آن‌ها را بازیابی کرده و در برنامه استفاده نماییم چه باید کرد. به عنوان یک پیشنهاد می‌توان از مجموعه‌های موجود در فضای نام System.Collection بهره بگیریم. اما همان طور که واضح است این مجموعه از داده‌ها به صورت جنریک نمی‌باشند و امکان استفاده از Queryهای Linq در آن‌ها به صورت معمول امکان پذیر نیست. برای حل این مشکل در دات نت دو متد تعبیه شده است که وظیفه آن تبدیل این مجموعه از داده‌ها به مجموعه ای است که بتوان بر روی آن‌ها Queryهای از جنس Linq یا Lambda Expression را اجرا کرد.
Cast
OfType

#مثال 1
فرض کنید یک مجموعه مثل زیر داریم:

```
ArrayList myList = new ArrayList();

  myList.Add( "Value1" );
  myList.Add( "Value2" );
  myList.Add( "Value3" );

  var myCollection = myList.Cast<string>();
```

در مثال بالا یک Collection از نوع ArrayList ایجاد کردیم که در فضای نام System.Collection قرار دارد. شما در این مجموعه می‌توانید از هر نوع داده ای که مد نظرتان است استفاده کنید. با استفاده از اپراتور Cast توانستیم این مجموعه را به نوع مورد نظر خودمان تبدیل کنیم و در نهایت به یک مجموعه از IEnumerable<T> برسیم. حال امکان استفاده از تمام متدهای Linq امکان پذیر است.
#مثال دوم:

```
ArrayList myList = new ArrayList();

    myList.Add( "Value1" );
    myList.Add( 10 );
    myList.Add( 10.2 );

   var myCollection = myList.Cast<string>();
```

در مثال بالا در خط آخر با یک runtime Error مواجه خواهیم شد. دلیلش هم این است که ما از در ArrayList خود داده‌های غیر از string نظیر int یا double داریم. درنتیجه هنگام تبدیل داده‌های int یا double به string یک Exception رخ خواهد داد. در این گونه موارد که در لیست مورد نظر داده‌های غیر هم نوع وجود دارد باید متد OfType را جایگزین کنیم.

```
ArrayList myList = new ArrayList();

myList.Add( "Value1" );
myList.Add( 10 );
myList.Add( 10.2 );
```

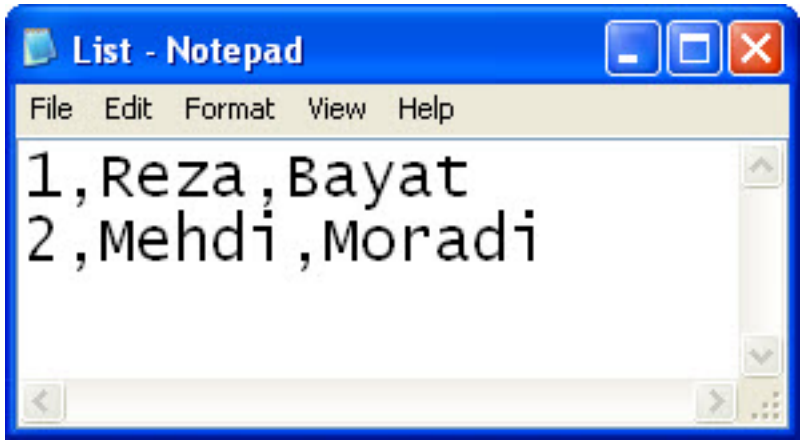
```
var doubleNumber = myList.OfType<double>().Single();
var integerNumber = myList.OfType<int>().Single();
var stringValue = myList.OfType<string>().Single();
```

تفاوت بین متد Cast و OfType در این است که متد Cast سعی دارد تمام داده‌های موجود در مجموعه را به نوع مورد نظر تبدیل کند ولی متد OfType فقط داده‌های از نوع مشخص شده را برگشت خواهد داد. حتی اگر هیچ آیتمی از نوع مورد نظر در این مجموعه نباشد یک مجموعه بدون هیچ داده ای برگشت داده می‌شود.

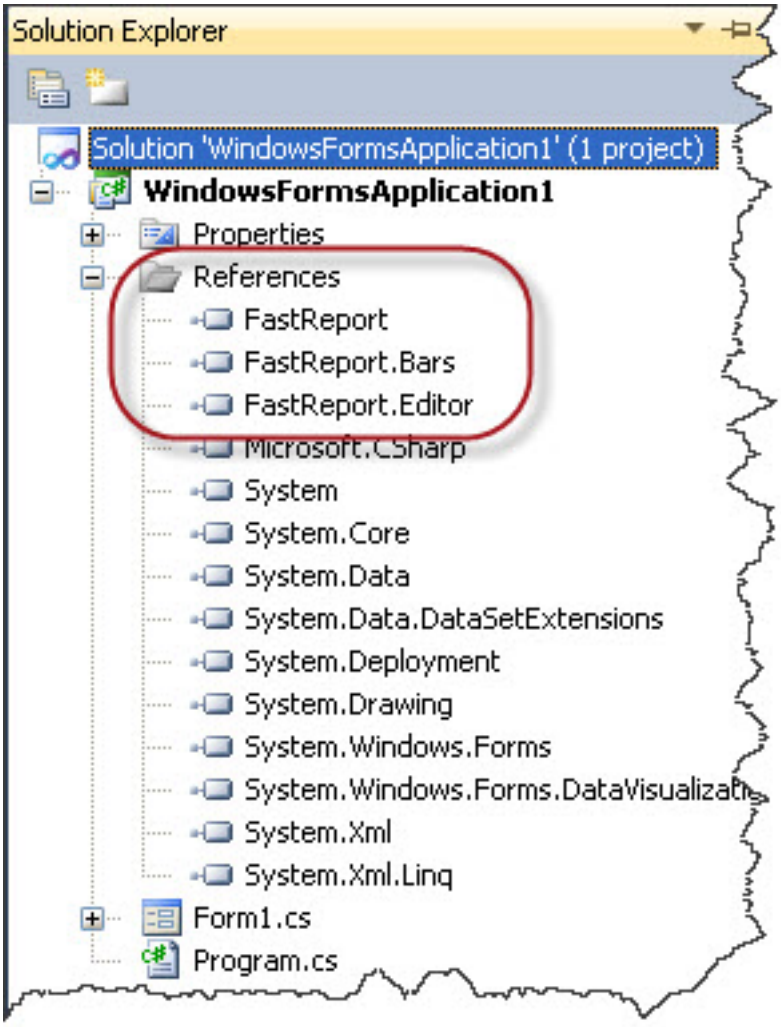
عنوان: **ساخت گزارش با استفاده از FastReport & Linq**
نویسنده: رضابیات
تاریخ: ۱۳۹۲/۰۵/۱۵ ۹:۴۵
آدرس: www.dotnettips.info
برچسب‌ها: LINQ, Reporting, FastReport

یک روش ساده جهت ساخت گزارش به کمک FastReport استفاده از منبع داده ایجاد شده توسط Linq است. بعنوان نمونه در اینجا اطلاعات داخل یک فایل متنی (List.txt) ذخیره شده است. با استفاده از دستورات زبان Linq اطلاعات فایل متنی استخراج و داخل Query قرار گرفته است. یک نمونه از Report ایجاد و با استفاده از report.RegisterData منبع داده را به FastReport معرفی می‌کنیم. ابتدا از report.Design جهت طراحی گزارش استفاده و سپس با report.Load گزارش ساخته شده (در اینجا با نام List.frx ذخیره شده) را بارگذاری و توسط report.Show نمایش میدهیم

محتوای فایل نمونه List.txt



افزودن اسمبلی‌های مورد نیاز به مجموع Referencesها

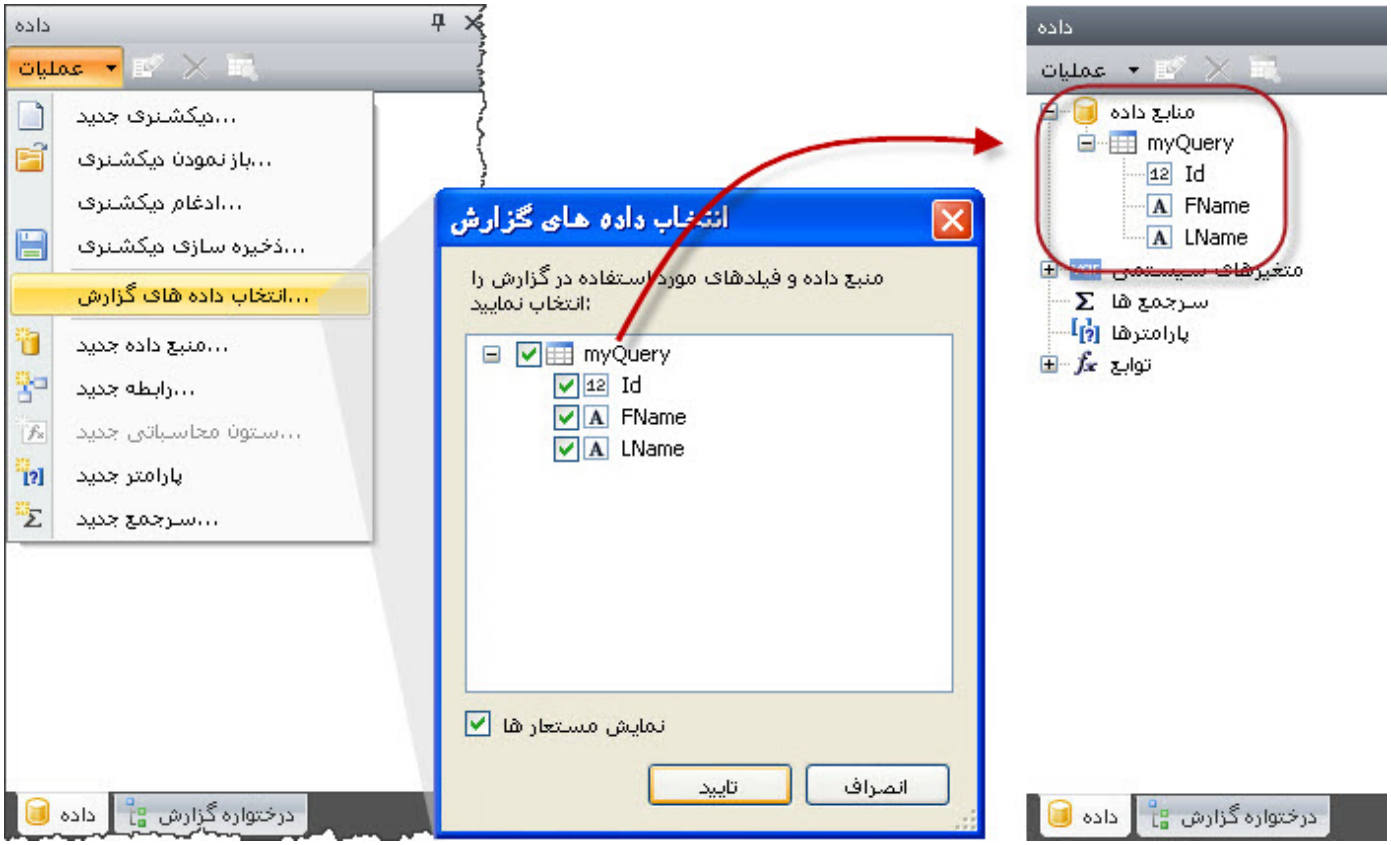


کد استفاده شده جهت طراحی گزارش

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using FastReport;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            string[] List = System.IO.File.ReadAllLines("List.TXT");
            var Query = from list in List
                        let items = list.Split(',')
                        select new
                        {
                            Id = Convert.ToInt32(items[0]),
```

```
                            FName = items[1],
                            LName = items[2]
                        };

            using (Report report = new Report())
            {
                report.RegisterData(Query.ToList(), "myQuery");
                report.Design();
            }
        }
    }
}
```

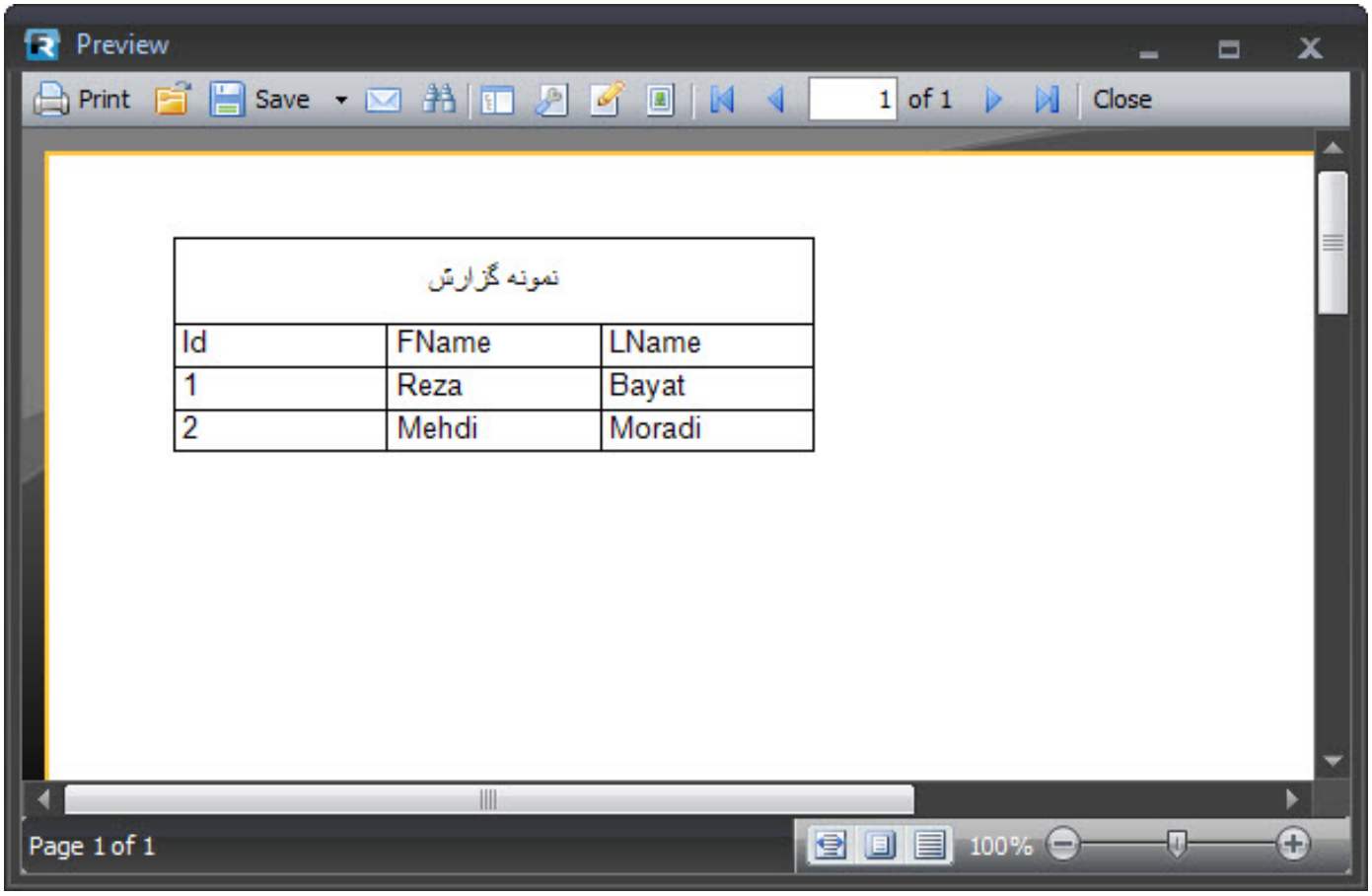
نحوه شناسایی منبع و فیلدها در FastReport



نمایش گزارش ذخیره شده در List.frx با استفاده از کد زیر

```
report.Load("List.frx");
 report.Show();
```

خروجی گزارش ساخته شده

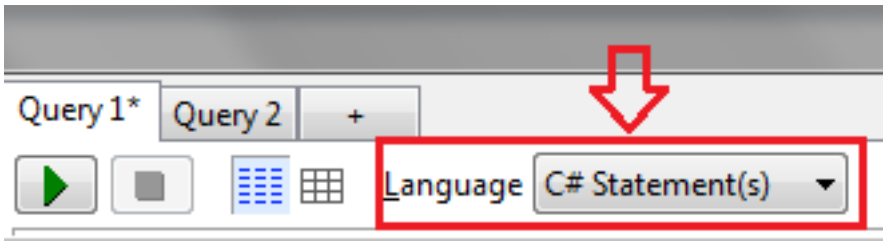


سورس برنامه نمونه

Linq_FastReport-sample.rar

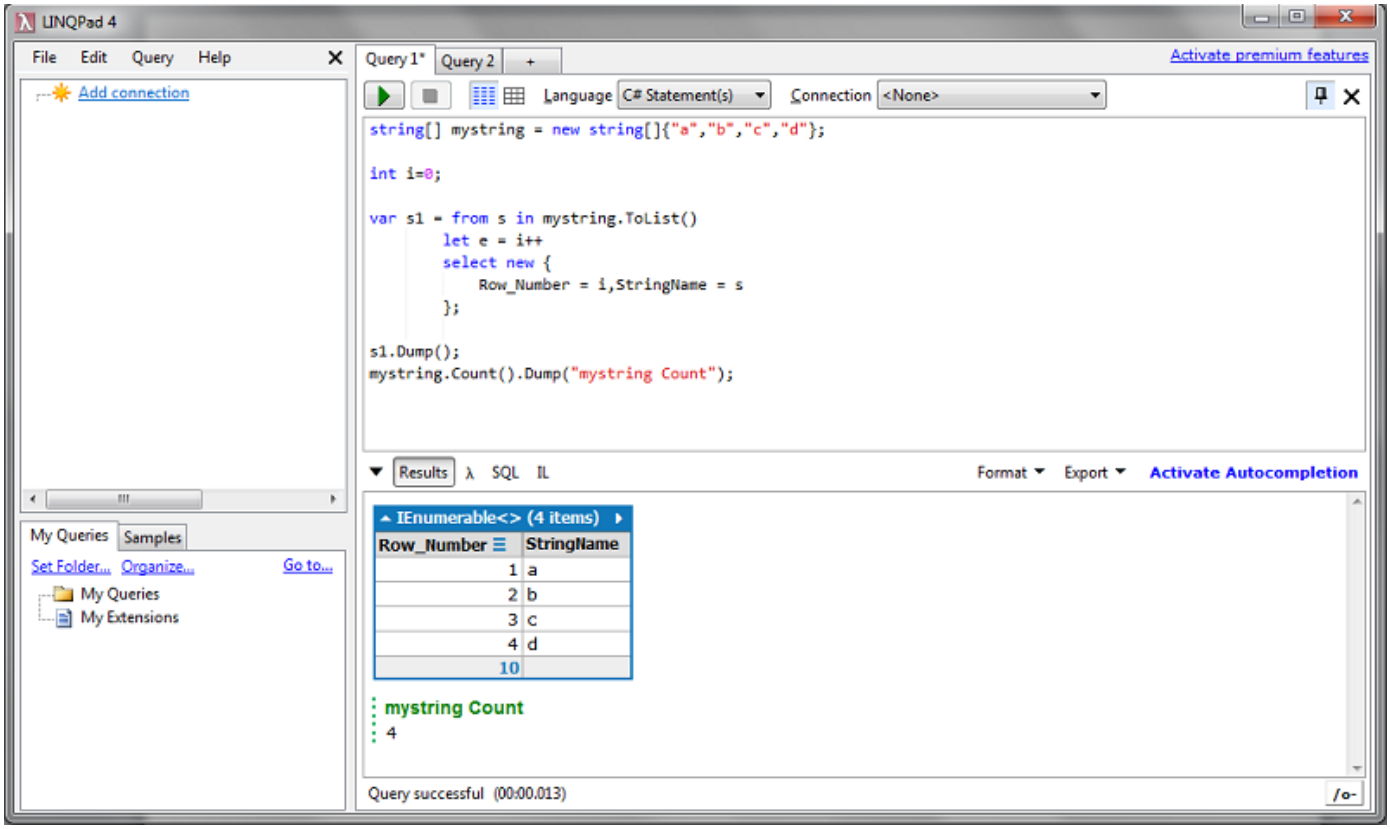
عنوان: **نحوه ایجاد شمارنده Row_Number() Sql Server در LINQ**
نویسنده: بهمن خلفی
تاریخ: ۱۳۹۲/۰۵/۱۹ ۱۰:۵
آدرس: www.dotnettips.info
برچسب‌ها: LINQ, window function, linqpad

چند روز پیش برای انجام یک بخشی از کار پروژه خودم باید از توابع و window functionها در sql server استفاده میکردم که در سایت جاری آشنایی با Row_Number،Rank،Dense_Rank،NTILE و آشنایی با Window Functionها در SQL Server بصورت مفصل توضیح داده شده است.
حال اگر بخواهیم یکی از پرکاربردترین این توابع که Row_Number می‌باشد را در LINQ استفاده کنیم باید به چه صورت عمل کنیم.
من برای پیاده سازی از برنامه نیمه رایگان LINQPad استفاده کردم که میتوانید از سایت اصلی این نرم افزار دانلود نمائید.
پس از دانلود و اجرای آن ، در قسمت بالایی زبان linqpad را به C# Statement(s) تغییر دهید.

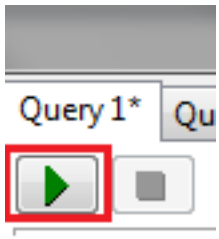


سپس کد زیر را به بخش query انتقال دهید.

```
string[] mystring = new string[]{"a","b","c","d"};

int i=0;

var s1 = from s in mystring.ToList()
let e = i++
select new {
Row_Number = i,StringName = s
};

s1.Dump();
mystring.Count().Dump("mystring Count");
```



سپس با زدن کلید F5 یا دکمه اجرای query نتیجه را مشاهده نمائید.



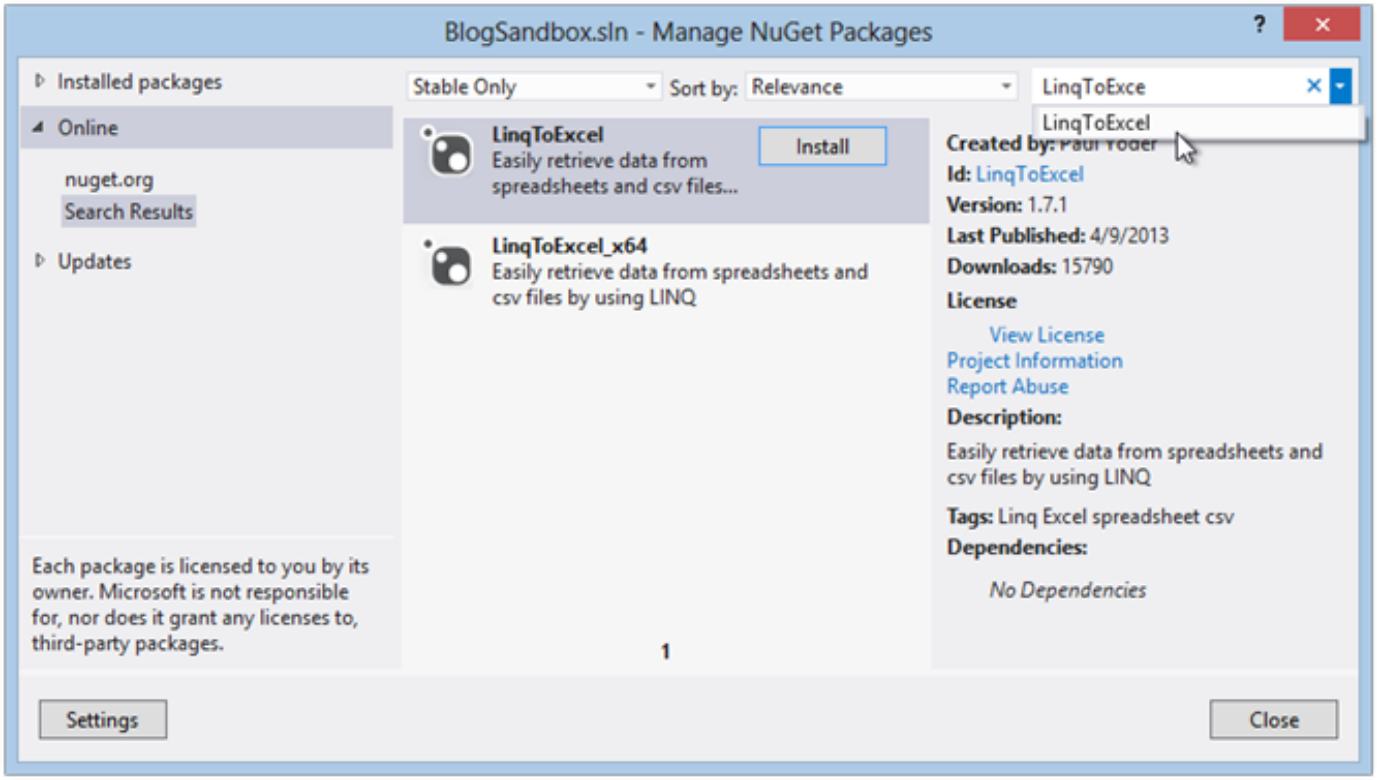
use-row_number-in-Linq.linq

عنوان: **خواندن اطلاعات از فایل اکسل با استفاده از LinqToExcel**
نویسنده: مسعود حق شناس
تاریخ: ۱۳۹۲/۰۸/۰۶ ۲۲:۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ, Excel

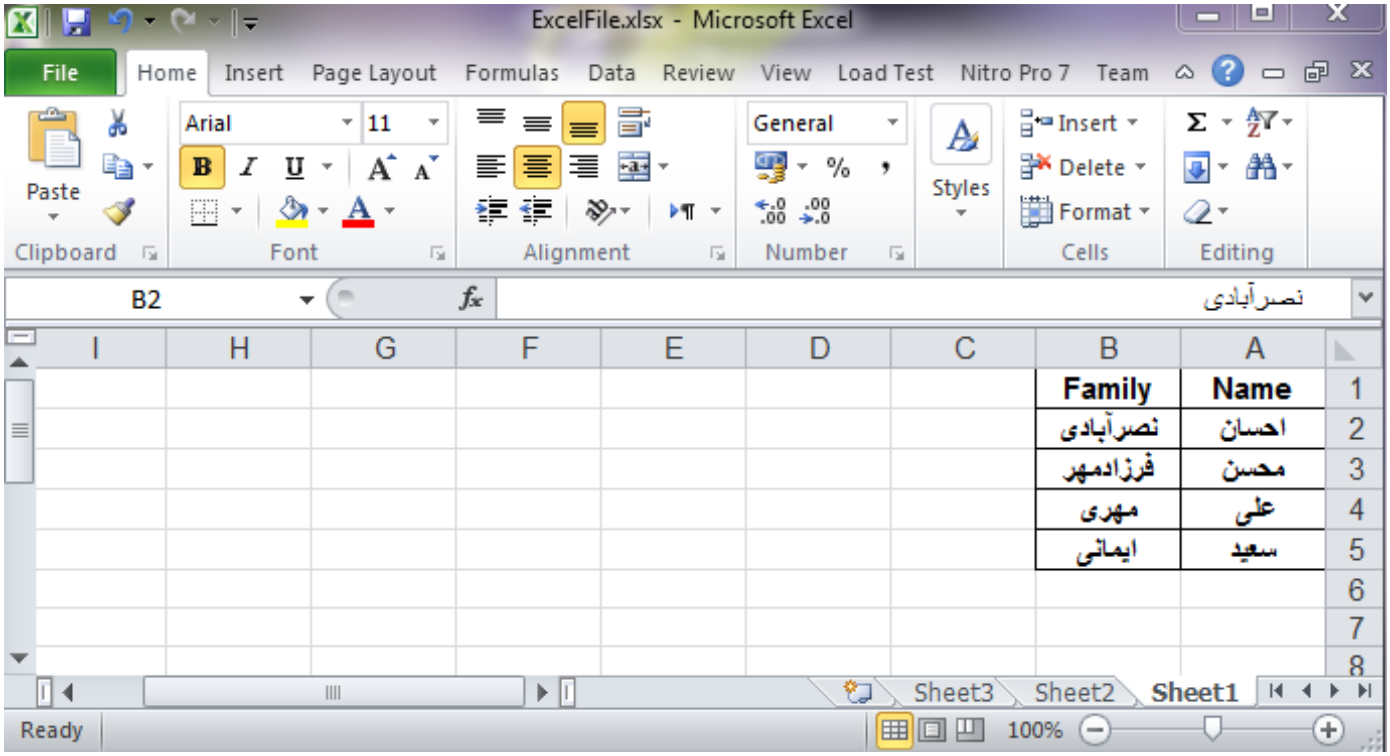
در این مقاله مروری سریع و کاربردی خواهیم داشت بر توانایی‌های مقدماتی LinqToExcel
در ابتدا می‌بایست **LinqToExcel** را از طریق NuGet به پروژه افزود.

```
PM> Install-Package LinqToExcel
```

و یا از طریق solution Explorer گزینه Manage NuGet Packages



اکنون فایل اکسل ذیل را در نظر بگیرید.



روش خواندن اطلاعات از فایل اکسل فوق تحت فرامین Linq و با مشخص کردن نام sheet مورد نظر توسط شئ ExcelQueryFactory بصورت زیر است.

```
string pathToExcelFile = @"C:\Users\MASOUD\Desktop\ExcelFile.xlsx";
var excel = new ExcelQueryFactory(pathToExcelFile);
            string sheetName = "Sheet1";
            var persons = from a in excel.Worksheet(sheetName) select a;
            foreach (var a in persons)
            {
                MessageBox.Show(a["Name"]+" "+a["Family"]);
            }
```

در صورتیکه بخواهیم انتقال اطلاعات فایل اکسل به جداول بانک اطلاعاتی مانند Sql Server بطور مثال با روش EF Entity Framework را انجام دهیم کلاس زیر با نام person را فرض نمایید.

```
public class Person
        {
            public string Name { get; set; }
            public string Family { get; set; }
        }
```

باید بدانید که بصورت پیشفرض سطر اول از فایل اکسل به عنوان نام ستون انتخاب می‌شود و می‌بایست جهت نگاشت با نام propertyهای کلاس ما دقیقاً همنام باشد.

```
string pathToExcelFile = @"C:\Users\MASOUD\Desktop\ExcelFile.xlsx";
            var excel = new ExcelQueryFactory(pathToExcelFile);
            string sheetName = "Sheet1";
            var persons = from a in excel.Worksheet<Person>(sheetName) select a;
            foreach (var a in persons)
            {
                MessageBox.Show(a.Name+" "+a.Family);
```

```
            }
```

اگر فایل اکسل ما ستون‌های بیشتری داشته باشد تنها ستونهای همنام با propertyهای کلاس ما به کلاس نگاشت پیدا می‌کند و سایر ستونها نادیده گرفته می‌شود.
در صورتیکه نام ستونهای فایل اکسل(سطر اول) با نام propertyهای کلاس یکسان نباشد جهت نگاشت آنها در کلاس می‌توان از متد AddMapping استفاده نمود.

```
string pathToExcelFile = @"C:\Users\MASOUD\Desktop\ExcelFile.xlsx";
            var excel = new ExcelQueryFactory(pathToExcelFile);
            string sheetName = "Sheet1";
            excel.AddMapping("Name","نام");
            excel.AddMapping("Family", "نام خانوادگی");
            var persons = from a in excel.Worksheet<Person>(sheetName) select a;
            foreach (var a in persons)
            {
                MessageBox.Show(a.Name+" "+a.Family);
            }
```

در کدهای بالا در صورتی که sheetName قید نشود بصورت پیشفرض Sheet1 از فایل اکسل انتخاب می‌شود.

```
var persons = from a in excel.Worksheet<Person>() select a;
```

همچنین می‌توان از اندیس جهت مشخص نمودن Sheet مورد نظر استفاده نمود که اندیس‌ها از صفر شروع می‌شوند.

```
var persons = from a in excel.Worksheet<Person>(0) select a;
```

توسط متد GetWorksheetNames می توان نام sheetها را بدست آورد.

```
public IEnumerable<string> getWorkSheets()
{
string pathToExcelFile = @"C:\Users\MASOUD\Desktop\ExcelFile.xlsx";

    var excel = new ExcelQueryFactory(pathToExcelFile);

    return excel.GetWorksheetNames();
}
```

و توسط متد GetColumnNames می توان نام ستونها را بدست آورد.

```
var SheetColumnNames = excel.GetColumnNames(sheetName);
```

همانطور که می‌بینید با روش توضیح داده شده در این مقاله به راحتی از فرامین Linq مانند where می‌توان در انتخاب اطلاعات از فایل اکسل استفاده نمود و سپس نتیجه را به جداول مورد نظر انتقال داد.

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۸/۰۶ ۲۲:۴۲

با تشکر از شما. این کتابخانه LinqToExcel کار کیست؟ سایت اصلی آن کجاست؟ مجوز استفاده از آن به چه صورتی است؟

نویسنده: مسعود حق شناس
تاریخ: ۱۳۹۲/۰۸/۰۶ ۲۳:۴۹

یک ویدیوی آموزشی کوتاه جالب با توضیحات راجع به این کتابخانه اینجا وجود داره که پیشنهاد میکنم ببینید. فقط من بدون فیلترشکن نتونستم صفحه اش و باز کنم.

نویسنده: مهدی
تاریخ: ۱۳۹۲/۰۸/۰۷ ۱۰:۱۹

سلام
اگر نام ستون‌ها رو نداشته باشیم و فقط بخواهیم اطلاعات داخل شیت رو بدست بیاریم چیکار باید کرد در ضمن آیا باز نیازی به اسمبلی Microsoft.Office.Interop.Excel هست یا نه ؟

نویسنده: مسعود حق شناس
تاریخ: ۱۳۹۲/۰۸/۰۷ ۲۲:۳۹

با استفاده از متد WorksheetNoHeader و با وارد کردن شماره اندیس می‌توانید به اطلاعات سلولها دست پیدا کنید. در مثال زیر تمام سطرهایی که ستون دوم آنها شهر مشهد است انتخاب می‌شوند و سپس سلولهای آن سطرها نمایش داده می‌شوند.

```
var excel = new ExcelQueryFactory(pathToExcelFile);
            string sheetName = "Sheet1";
            var persons = from a in excel.WorksheetNoHeader(sheetName)
                          where a[1] == "Mashhad" //مقدار در ستون دوم
                          select a;
            foreach (var a in persons)
              {
                  for(int i=0;i<a.Count;i++)
                        MessageBox.Show(a[i]);
              }
```

به اسمبلی Microsoft.Office.Interop.Excel نیازی نیست و
LinqToExcel.dll و Remotion.Data.Linq.dll مورد نیاز است.

نویسنده: بهراد
تاریخ: ۱۳۹۲/۰۸/۰۸ ۸:۴۹

سلام من این خطا رو میگیرم

```
The 'Microsoft.ACE.OLEDB.12.0' provider is not registered on the local machine.
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۰۸ ۱۰:۴۶

مطابق توضیحات آن ، نیاز به AccessDatabaseEngine نیز دارد (Microsoft.ACE.OLEDB.12.0 مربوط به اکسس 2010 است).
احتمالا برای سازگاری با نگارش‌های قدیمی اکسل که با فرمت OpenXML نیستند از این نوع رشته اتصالی مخصوص اکسس 2010 در پشت صحنه استفاده کرده:

```
Driver={Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb)};DBQ=path to xls/xlsx/xlsm/xlsb file
```

نویسنده: noth50
تاریخ: ۱۳۹۲/۰۹/۱۱ ۱۰:۴۴

بادرود
ممنون از اطلاعات مفید شما .
من در یک پروژه تحت وب از طریق فایل اکسل اطلاعات را از کاربر دریافت می‌کنم و نیاز دارم این اطلاعات را در داخل دیتابیس ذخیره کنم از آنجا که تعداد رکوردهای فایل اکسل نامشخص است به چه صورت باید این کد را بنویسم
ممنون

عنوان: **ایجاد سرویس چندلایه‌ی WCF با Entity Framework در قالب پروژه - 1**
نویسنده: حامد قنادی
تاریخ: ۱۳۹۲/۱۰/۲۵ ۸:۲۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, LINQ, WCF

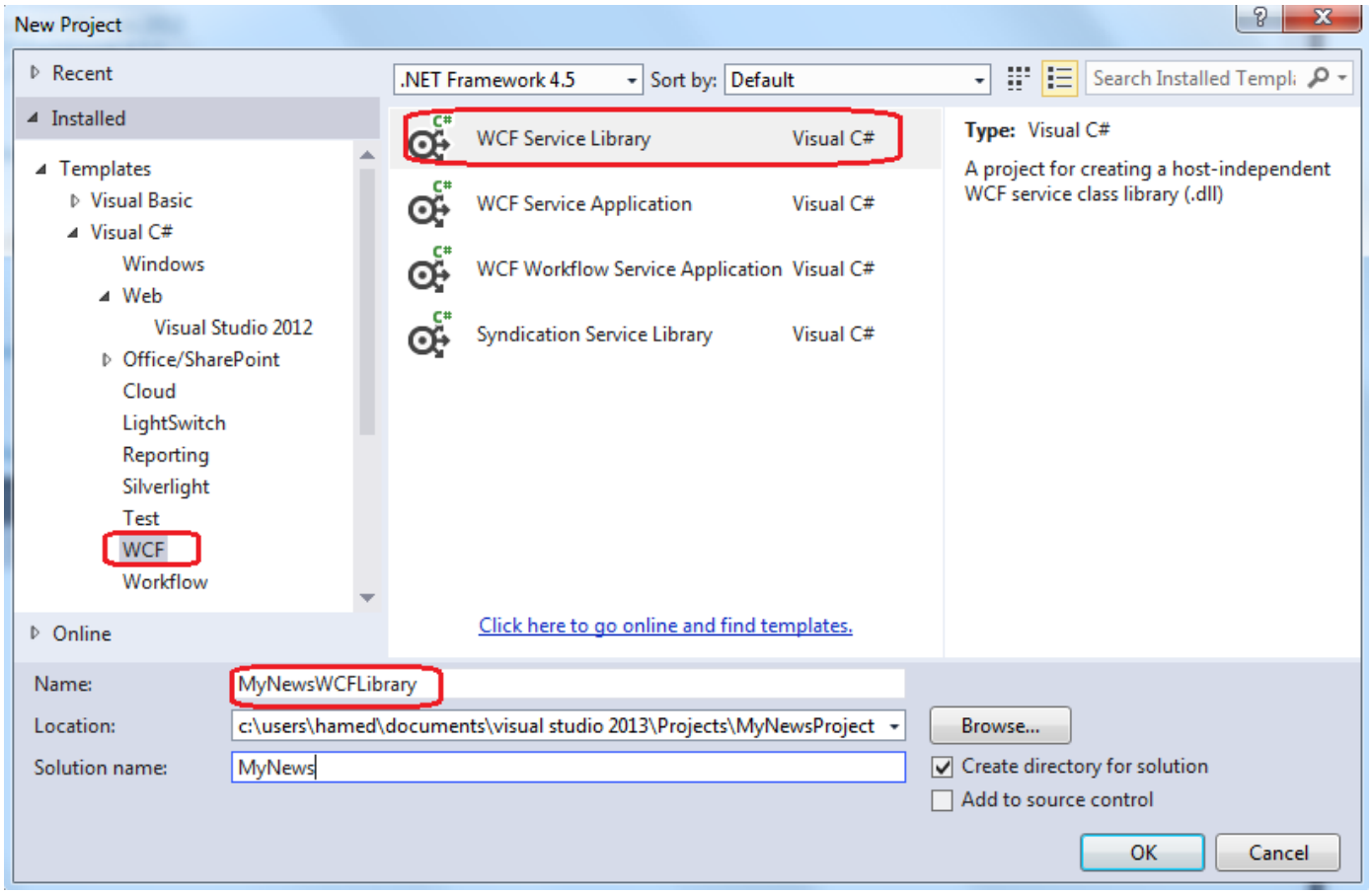
در این نوشتار که به صورت آموزش تصویری ارائه می‌شود؛ یک سرویس WCF در Visual Studio 2013 ایجاد می‌کنم، سپس روش استفاده از آن‌را در یک برنامه ویندوزی آموزش خواهم داد. در اینجا در نظرگرفته شده است که شما افزونه‌ی Resharper را روی ویژوال استودیوی خود نصب دارید. پس در صورتیکه هنوز به سراغ آن نرفته اید درنگ نکنید و واپسین نگارش آن را دانلود کنید.

در این پروژه‌ی ساده در نظر می‌گیریم که دو جدول یکی برای اخبار، شامل عنوان، متن خبر و تاریخ ثبت و دسته بندی و دیگری برای نگهداری دسته‌ها در پایگاه داده داریم و می‌خواهیم سرویس‌های مناسب با این دو جدول را بسازیم. با کد زیر، پایگاه داده‌ی dbTest و جدول‌های tblNews و tblCategory در SQL Server 2012 ساخته می‌شود:

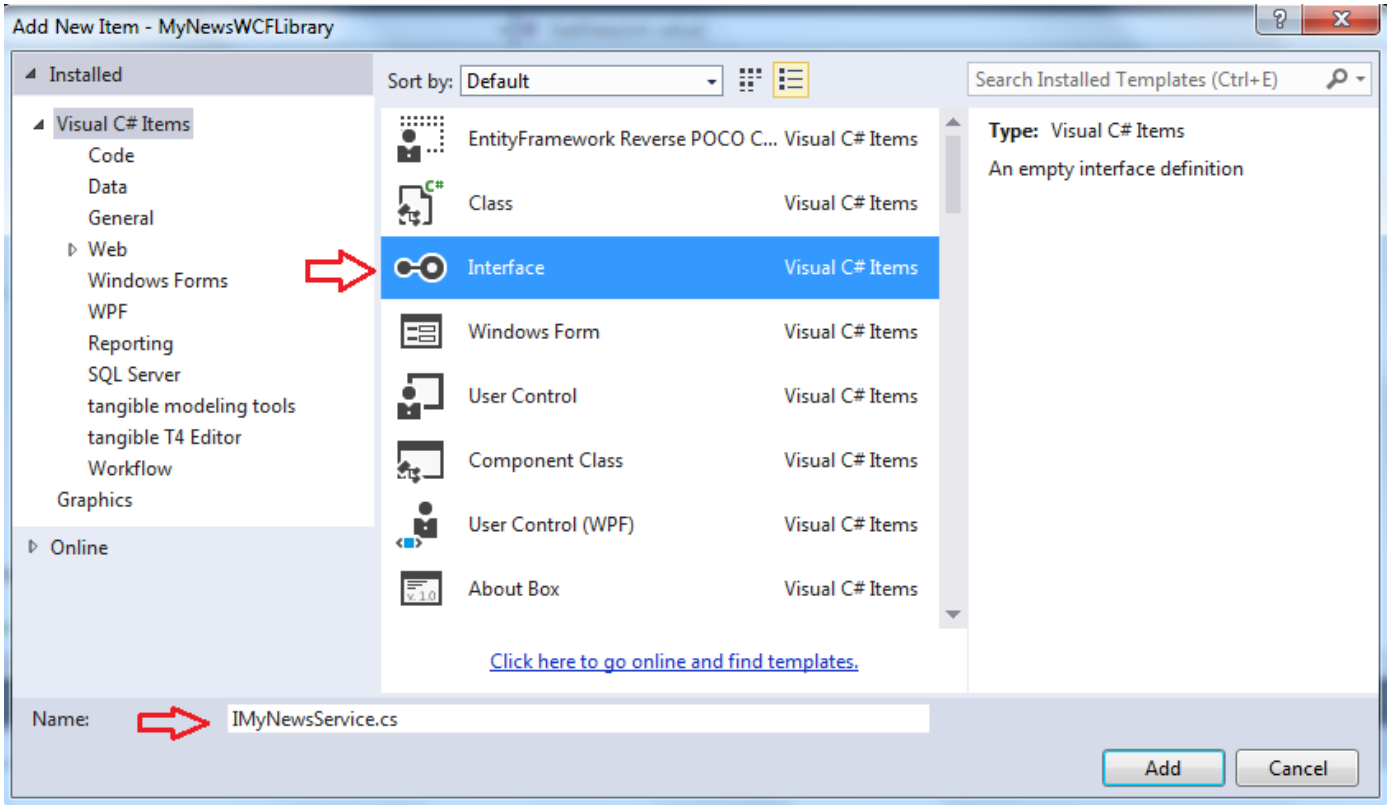
```
USE [master]
GO
/****** Object:  Database [dbMyNews]    Script Date: 2014/01/14 09:46:04 ب.ظ ******/
CREATE DATABASE [dbMyNews]
 CONTAINMENT = NONE
 ON  PRIMARY
( NAME = N'dbMyNews', FILENAME = N'D:\dbMyNews.mdf' , SIZE = 5120KB , MAXSIZE = UNLIMITED, FILEGROWTH =
1024KB )
 LOG ON
( NAME = N'dbMyNews_log', FILENAME = N'D:\dbMyNews_log.ldf' , SIZE = 1024KB , MAXSIZE = 2048GB ,
FILEGROWTH = 10%)
GO
USE [dbMyNews]
GO
/****** Object:  Table [dbo].[tblCategory]    Script Date: 2014/01/14 09:46:04 ب.ظ ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[tblCategory](
[tblCategoryId] [int] IDENTITY(1,1) NOT NULL,
[CatName] [nvarchar](50) NOT NULL,
[IsDeleted] [bit] NOT NULL,
 CONSTRAINT [PK_tblCategory] PRIMARY KEY CLUSTERED
(
[tblCategoryId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
/****** Object:  Table [dbo].[tblNews]    Script Date: 2014/01/14 09:46:04 ب.ظ ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[tblNews](
[tblNewsId] [int] IDENTITY(1,1) NOT NULL,
[tblCategoryId] [int] NOT NULL,
[Title] [nvarchar](50) NOT NULL,
[Description] [nvarchar](max) NOT NULL,
[RegDate] [datetime] NOT NULL,
[IsDeleted] [bit] NULL,
 CONSTRAINT [PK_tblNews] PRIMARY KEY CLUSTERED
(
[tblNewsId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

GO
ALTER TABLE [dbo].[tblNews]  WITH CHECK ADD  CONSTRAINT [FK_tblNews_tblCategory] FOREIGN
KEY([tblCategoryId])
REFERENCES [dbo].[tblCategory] ([tblCategoryId])
GO
ALTER TABLE [dbo].[tblNews] CHECK CONSTRAINT [FK_tblNews_tblCategory]
GO
USE [master]
```

```
GO
ALTER DATABASE [dbMyNews] SET  READ_WRITE
GO
```

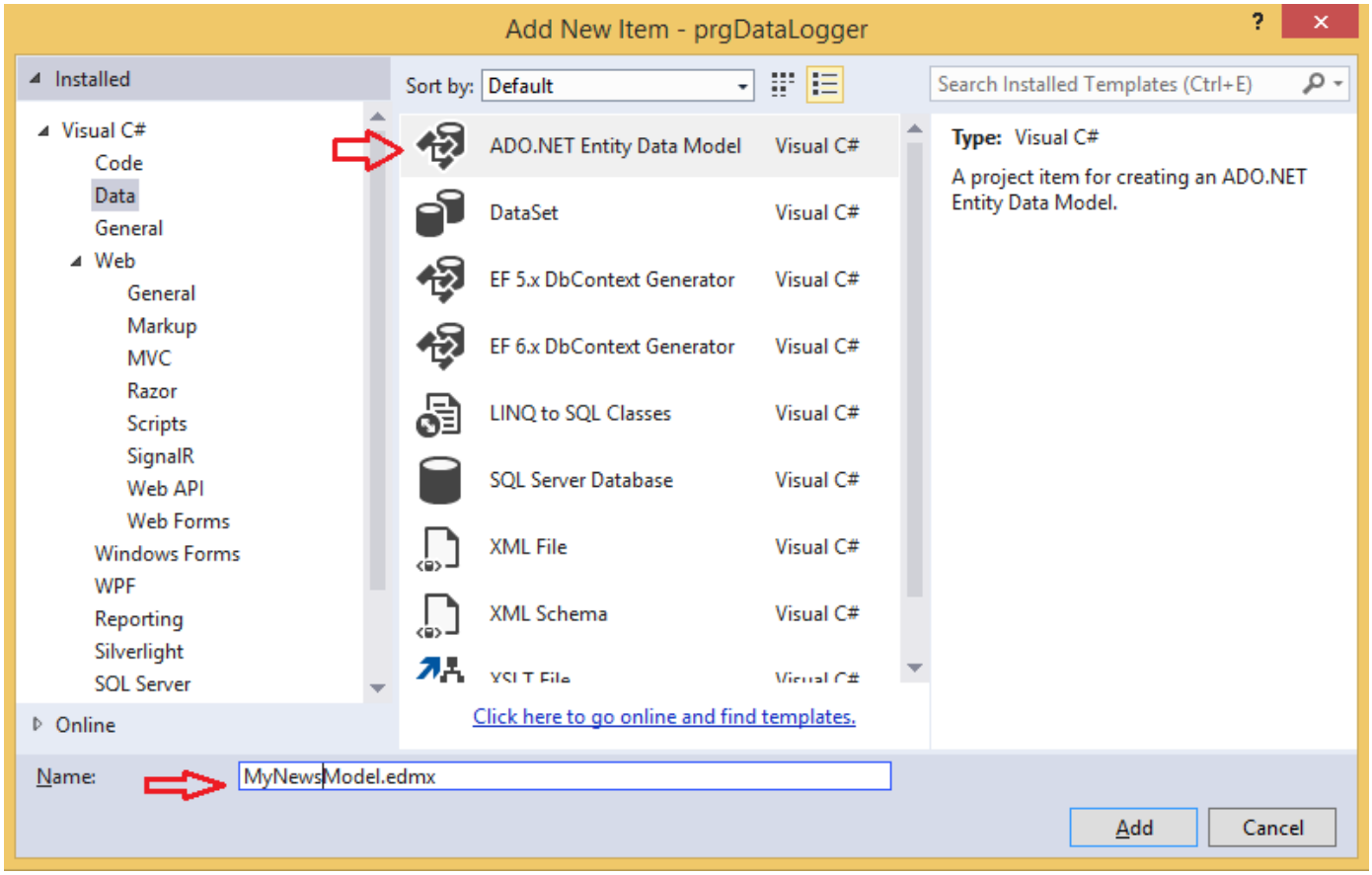
اکنون Visual Studio 2013 را بازکنید سپس روی گزینه New Project کلیک کنید و برابر با نگاره‌ی زیر عمل کنید:

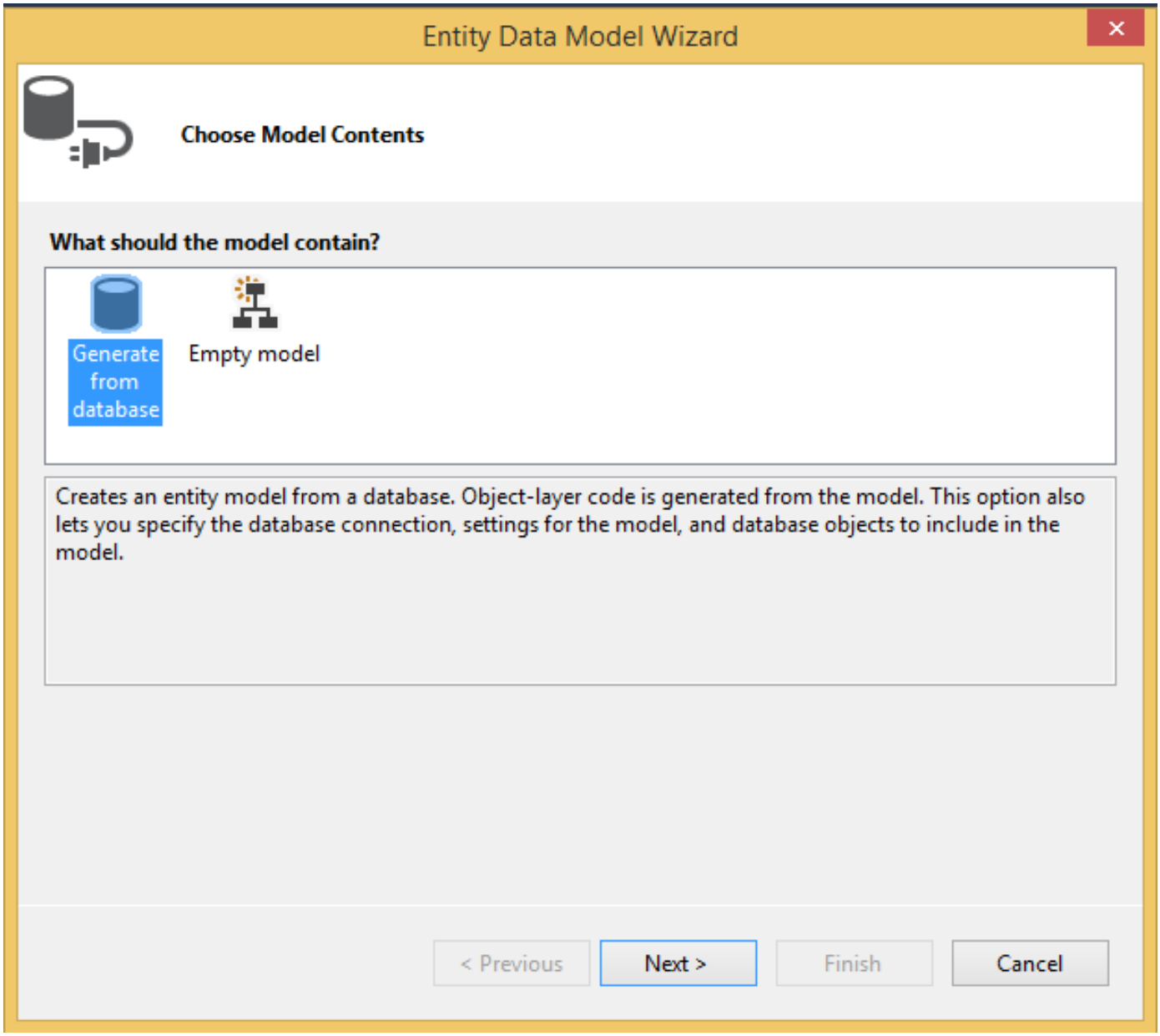


پروژه MyNewsWCFLibrary در راه حل MyNews ساخته می‌شود. این پروژه به صورت پیش‌گزیده دارای یک کلاس به نام Service و یک interface به نام IService است. هر دو را حذف کنید و سپس روی نام پروژه راست‌کلیک کرده، از منوی بازشده گزینه‌ی Add -> New Item را انتخاب کنید. سپس برابر با نگاره‌ی زیر عمل کنید:

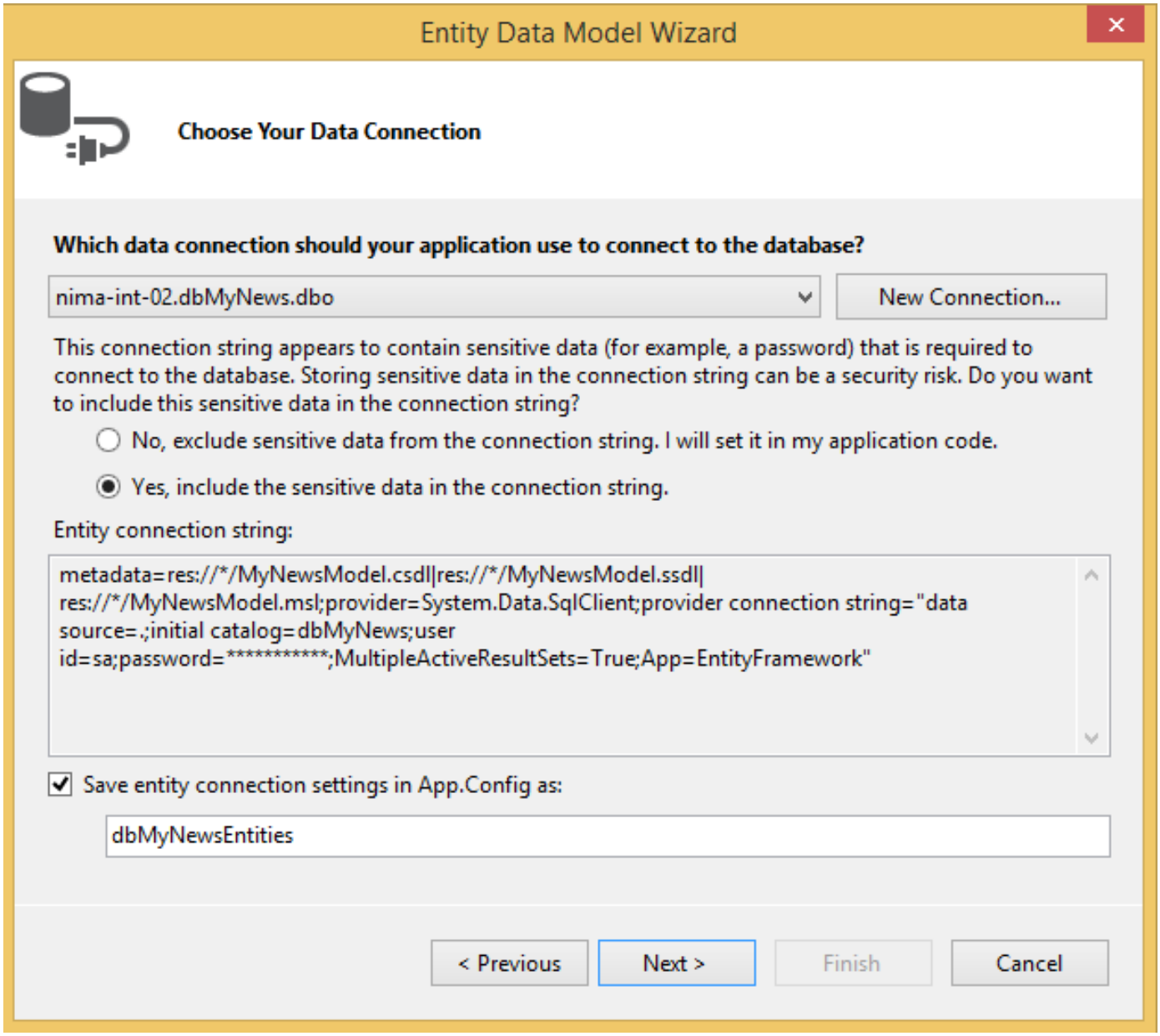


در لایه‌ی Service Interface کلیه‌ی روال‌های مورد نیاز برای ارتباط با پایگاه داده را می‌سازیم. پیش از آن باید یک Model برای ارتباط با پایگاه داده ساخته باشیم. برای این کار از پنجره Add New Item و از زیرمجموعه Data، گزینه ADO.NET Entity Data Model را انتخاب کنید و به‌سان زیر پیش روید:


Add New Item - prgDataLogger
Installed
Visual C#
Code
Data
General
Web
General
Markup
MVC
Razor
Scripts
SignalR
Web API
Web Forms
Windows Forms
WPF
Reporting
Silverlight
SQL Server
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
ADO.NET Entity Data Model Visual C#
DataSet Visual C#
EF 5.x DbContext Generator Visual C#
EF 6.x DbContext Generator Visual C#
LINQ to SQL Classes Visual C#
SQL Server Database Visual C#
XML File Visual C#
XML Schema Visual C#
Click here to go online and find templates.
Type: Visual C#
A project item for creating an ADO.NET Entity Data Model.
Name: MyNewsModel.edmx
Add
Cancel

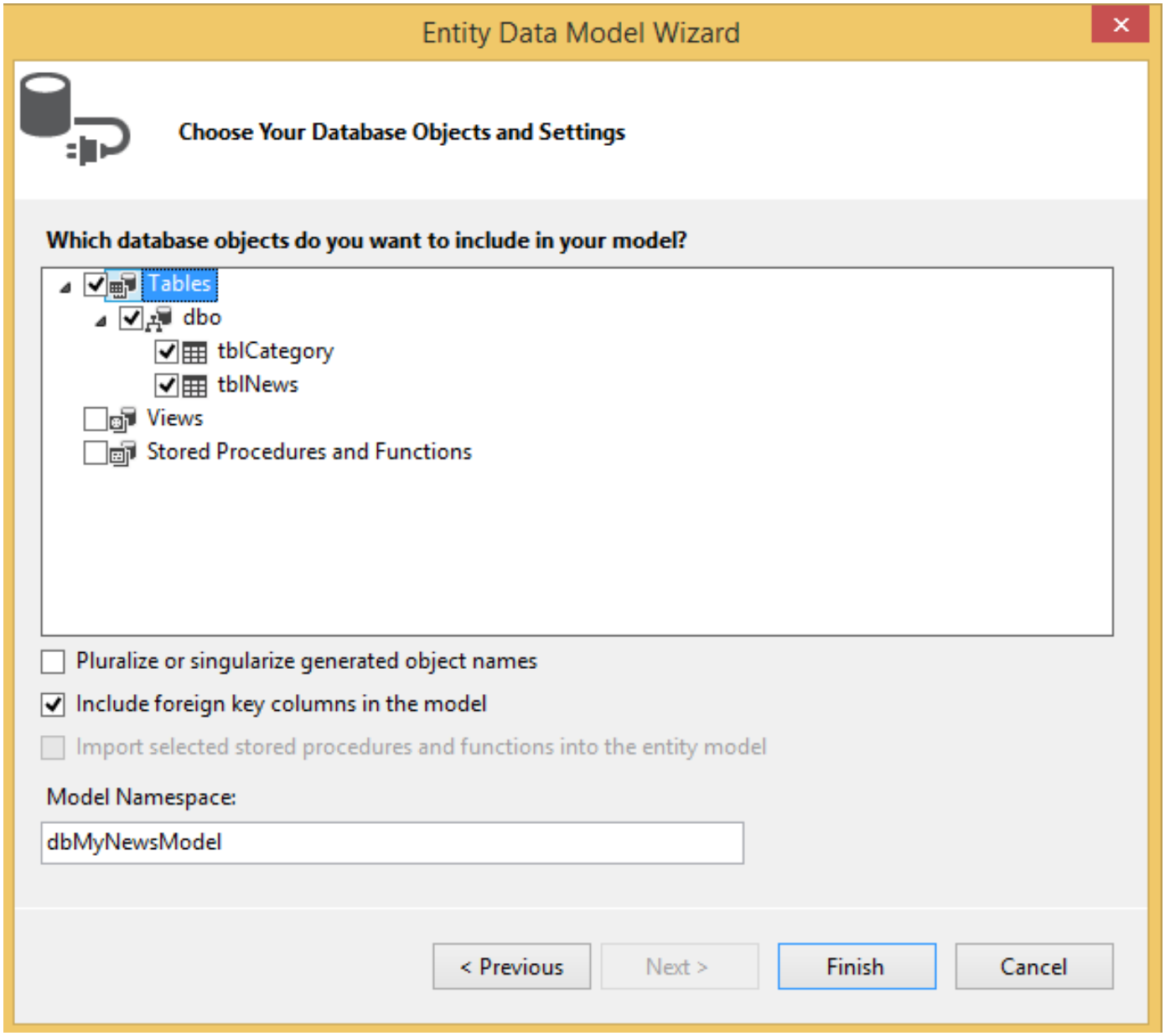


در گام پسین روی دکمه New Connection کلیک کنید و رشته‌ی اتصال به پایگاه داده‌ی dbMyNews را بسازید. سپس همانند تنظیمات نگاره‌ی زیر ادامه دهید:

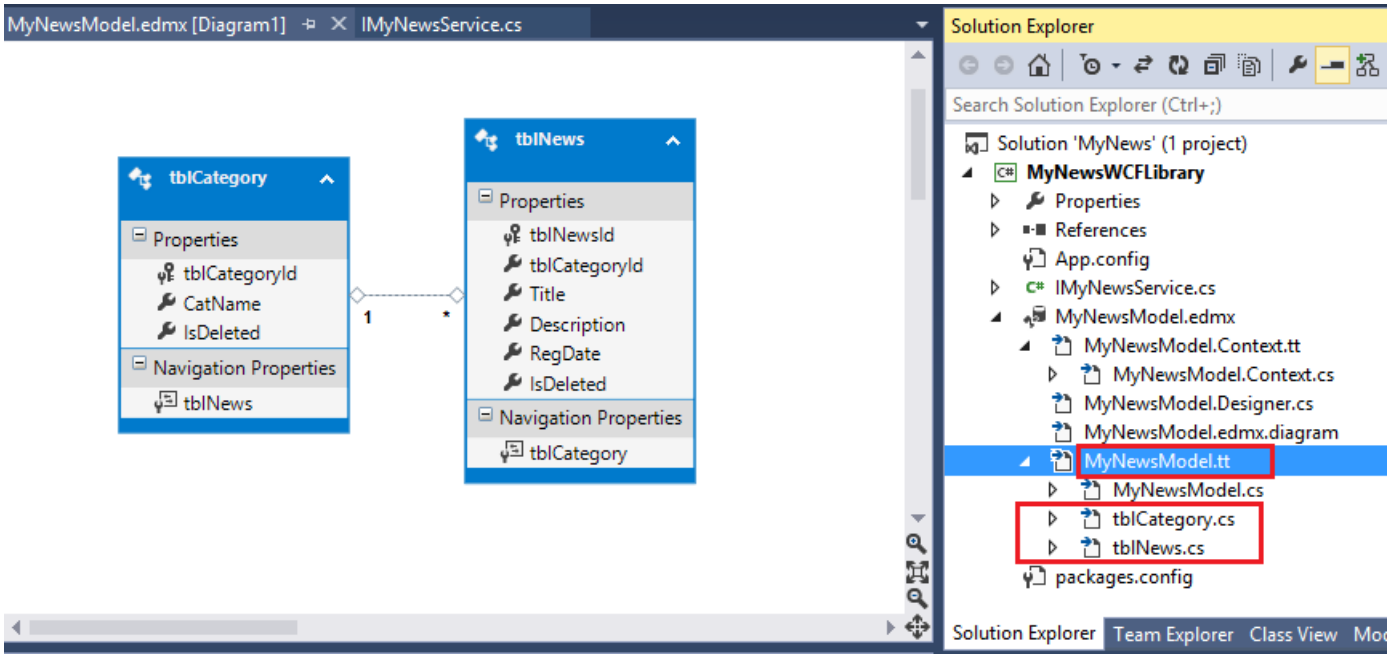


در گام پسین گزینه‌ی Entity Framework 6.0 را برگزینید و روی دکمه‌ی Next کلیک کنید.

در پنجره نشان داده شده، جدول‌های مورد نیاز را همانند نگاره‌ی زیر انتخاب کرده و روی دکمه Finish کلیک کنید:



در پایان مدل ما همانند نگاره‌ی زیر خواهد بود.



در بخش پسین درباره‌ی شیوه‌ی دست‌کاری کلاس‌های Entity خواهم نوشت.

عنوان: **ایجاد سرویس چندلایه‌ی WCF با Entity Framework در قالب پروژه - 2**
نویسنده: حامد قنادی
تاریخ: ۱۳۹۲/۱۰/۲۵ ۲۰:۳۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, LINQ, WCF, Entity Framework 6

برای استفاده از کلاس‌های Entity که در نوشتار پیشین ایجاد کردیم در WCF باید آن کلاس‌ها را دست‌کاری کنیم. متن کلاس tblNews را در نظر بگیرید:

```
namespace MyNewsWCFLibrary
{
    using System;
    using System.Collections.Generic;

    public partial class tblNews
    {
        public int tblNewsId { get; set; }
        public int tblCategoryId { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public System.DateTime RegDate { get; set; }
        public Nullable<bool> IsDeleted { get; set; }

        public virtual tblCategory tblCategory { get; set; }
    }
}
```

مشاهده می‌کنید که برای تعریف کلاس‌ها از کلمه کلیدی partial استفاده شده است. استفاده از کلمه کلیدی partial به شما اجازه می‌دهد که یک کلاس را در چندین فایل جداگانه تعریف کنید. به عنوان مثال می‌توانید فیلدها، ویژگی ها و سازنده‌ها را در یک فایل و متدها را در فایل دیگر قرار دهید.
به صورت خودکار کلیه‌ی ویژگی‌ها به توجه به پایگاه داده ساخته شده اند. برای نمونه ما برای فیلد IsDeleted در SQL Server ستون Allow Nulls را کلیک کرده بودیم که در نتیجه در اینجا عبارت Nullable پیش از نوع فیلد نشان داده شده است. برای استفاده از این کلاس در WCF باید صفت DataContract را به کلاس داد. این قرارداد به ما اجازه استفاده از ویژگی‌هایی که صفت DataMember را می‌گیرند را می‌دهد.
کلاس بالا را به شکل زیر بازنویسی کنید:

```
using System.Runtime.Serialization;

namespace MyNewsWCFLibrary
{
    using System;
    using System.Collections.Generic;

    [DataContract]
    public partial class tblNews
    {
        [DataMember]
        public int tblNewsId { get; set; }
        [DataMember]
        public int tblCategoryId { get; set; }
        [DataMember]
        public string Title { get; set; }
        [DataMember]
        public string Description { get; set; }
        [DataMember]
        public System.DateTime RegDate { get; set; }
        [DataMember]
        public Nullable<bool> IsDeleted { get; set; }

        public virtual tblCategory tblCategory { get; set; }
    }
}
```

هم‌چنین کلاس tblCategory را به صورت زیر تغییر دهید:

```
namespace MyNewsWCFLibrary
{
    using System;
```

```
    using System.Collections.Generic;
    using System.Runtime.Serialization;

    [DataContract]
    public partial class tblCategory
    {
        public tblCategory()
        {
            this.tblNews = new HashSet<tblNews>();
        }

        [DataMember]
        public int tblCategoryId { get; set; }
        [DataMember]
        public string CatName { get; set; }
        [DataMember]
        public bool IsDeleted { get; set; }

        public virtual ICollection<tblNews> tblNews { get; set; }
    }
}
```

با انجام کد بالا از بابت مدل کارمان تمام شده است. ولی فرض کنید در اینجا تصمیم به تغییری در پایگاه داده می‌گیرید. برای نمونه می‌خواهید ویژگی Allow Nulls فیلد IsDeleted را نیز False کنیم و مقدار پیش‌گزیده به این فیلد بدهید. برای این کار باید دستور زیر را در SQL Server اجرا کنیم:

```
BEGIN TRANSACTION
GO
ALTER TABLE dbo.tblNews
DROP CONSTRAINT FK_tblNews_tblCategory
GO
ALTER TABLE dbo.tblCategory SET (LOCK_ESCALATION = TABLE)
GO
COMMIT
BEGIN TRANSACTION
GO
CREATE TABLE dbo.Tmp_tblNews
(
tblNewsId int NOT NULL IDENTITY (1, 1),
tblCategoryId int NOT NULL,
Title nvarchar(50) NOT NULL,
Description nvarchar(MAX) NOT NULL,
RegDate datetime NOT NULL,
IsDeleted bit NOT NULL
)  ON [PRIMARY]
 TEXTIMAGE_ON [PRIMARY]
GO
ALTER TABLE dbo.Tmp_tblNews SET (LOCK_ESCALATION = TABLE)
GO
ALTER TABLE dbo.Tmp_tblNews ADD CONSTRAINT
DF_tblNews_IsDeleted DEFAULT 0 FOR IsDeleted
GO
SET IDENTITY_INSERT dbo.Tmp_tblNews ON
GO
IF EXISTS(SELECT * FROM dbo.tblNews)
 EXEC('INSERT INTO dbo.Tmp_tblNews (tblNewsId, tblCategoryId, Title, Description, RegDate, IsDeleted)
SELECT tblNewsId, tblCategoryId, Title, Description, RegDate, IsDeleted FROM dbo.tblNews WITH (HOLDLOCK
TABLOCKX)')
GO
SET IDENTITY_INSERT dbo.Tmp_tblNews OFF
GO
DROP TABLE dbo.tblNews
GO
EXECUTE sp_rename N'dbo.Tmp_tblNews', N'tblNews', 'OBJECT'
GO
ALTER TABLE dbo.tblNews ADD CONSTRAINT
PK_tblNews PRIMARY KEY CLUSTERED
(
tblNewsId
) WITH( STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
ON) ON [PRIMARY]

GO
ALTER TABLE dbo.tblNews ADD CONSTRAINT
FK_tblNews_tblCategory FOREIGN KEY
(
tblCategoryId
) REFERENCES dbo.tblCategory
```

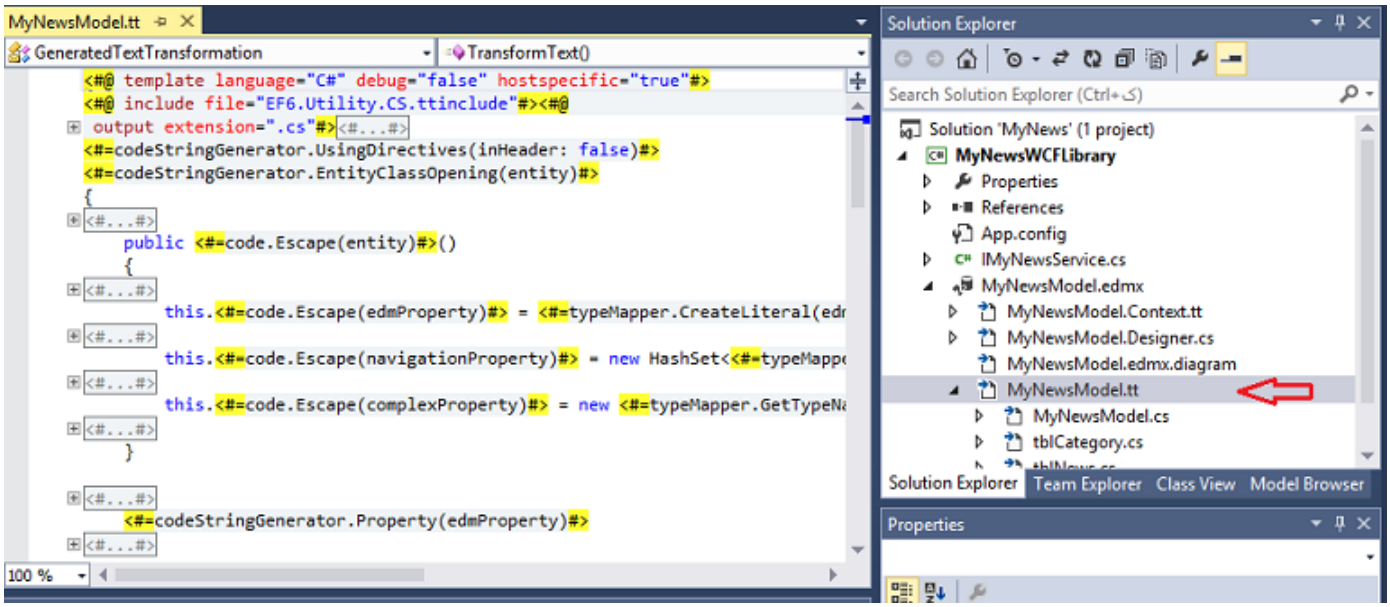
```
(
tblCategoryId
) ON UPDATE  NO ACTION
 ON DELETE  NO ACTION

GO
COMMIT
```

پس از آن مدل Entity Framework را باز کنید و در جایی از صفحه راست‌کلیک کرده و از منوی بازشده گزینه Update Model from Database را انتخاب کنید. سپس در پنجره بازشده، چون هیچ جدول، نما یا روالی به پایگاه داده‌ها نیفزوده ایم؛ دگمه‌ی Finish را کلیک کنید. دوباره کلاس tblNews را بازکنید. متوجه خواهید شد که همه‌ی DataContractها و DataMemberها را حذف شده است. ممکن است بگویید می‌توانستیم کلاس یا مدل را تغییر دهیم و به وسیله‌ی Generate Database from Model به‌هنگام کنیم. ولی در نظر بگیرید که نیاز به ایجاد چندین جدول دیگر داریم و مدلی با ده‌ها Entity دارید. در این صورت همه‌ی تغییراتی که در کلاس داده ایم زدوده خواهد شد.

در بخش پسین، درباره‌ی این‌که چه کنیم که عبارت‌هایی که به کلاس‌ها می‌افزاییم حذف نشود؛ خواهم نوشت.

عنوان: **ایجاد سرویس چندلایه‌ی WCF با Entity Framework در قالب پروژه - 3**
نویسنده: حامد قنادی
تاریخ: ۱۳۹۲/۱۰/۲۶ ۱۴:۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, LINQ, WCF, Entity Framework 6

پیش از ادامه‌ی نوشتار بهتر است توضیحاتی درباره‌ی قالب‌های T4 داده شود. این قالب‌های مصنوعی حاوی کدهایی که است که هدف آن صرفه‌جویی در نوشتن کد توسط برنامه نویس است. مثلاً در MVC شما یکبار قالبی برای صفحه Index خود تهیه می‌کنید که برای نمونه بجای ساخت جدول ساده، از گرید Kendo استفاده کند و همچنین دارای دکمه ویرایش و جزئیات باشد. از این پس هر بار که نیاز به ساخت یک نمای نوع لیست برای یک ActionResult داشته باشید فرم ساز MVC از قالب شما استفاده خواهد کرد.
روشن است که خود Visual Studio نیز از T4 در ساخت بسیاری از فرم‌ها و کلاس‌ها بهره می‌برد.
خبر خوب این‌که برای ساخت کلاس‌های هر موجودیت در Entity Framework نیز از قالب‌های T4 استفاده می‌شود و این‌که این قالب‌ها در دسترس توسعه‌دهندگان برای ویرایش یا افزودن است.
افزونه‌ی Tangible را دریافت کنید و سپس نصب کنید. این افزونه ظاهر نامفهوم قالب‌های T4 را ساده و روشن می‌کند.
ما نیاز داریم که خود Visual Studio زحمت این سه کار را بکشد:
1- بالای هر کلاس موجودیت عبارت using System.Runtime.Serialization; را بنویسید.
2- صفت [DataContract] را پیش از تعریف کلاس بیفزاید.
3- صفت [DataMember] را پیش از تعریف هر ویژگی بیفزاید.
همانند شکل زیر روی فایل MyNewsModel.tt دوکلیک کنید تا محتوای آن در سمت چپ نشان داده شود. این محتوا باید ظاهری همانند شکل پیدا کرده باشد:



کد زیر را در محتوای فایل جست‌وجو کنید:

```
public string Property(EdmProperty edmProperty)
    {
        return string.Format(
            CultureInfo.InvariantCulture,
            "{0} {1} {2} {{ {3}get; {4}set; }}",
            Accessibility.ForProperty(edmProperty),
            _typeMapper.GetTypeName(edmProperty.TypeUsage),
            _code.Escape(edmProperty),
            _code.SpaceAfter(Accessibility.ForGetter(edmProperty)),
            _code.SpaceAfter(Accessibility.ForSetter(edmProperty)));
```

```
    }
```

متن آن‌را به این صورت تغییر دهید:

```
public string Property(EdmProperty edmProperty)
    {
        return string.Format(
            CultureInfo.InvariantCulture,
"[DataMember]" + Environment.NewLine +
            "{0} {1} {2} {{ {3}get; {4}set; }}",
            Accessibility.ForProperty(edmProperty),
            _typeMapper.GetTypeName(edmProperty.TypeUsage),
            _code.Escape(edmProperty),
            _code.SpaceAfter(Accessibility.ForGetter(edmProperty)),
            _code.SpaceAfter(Accessibility.ForSetter(edmProperty)));
    }
```

بار دیگر به دنبال این کد بگردید:

```
public string EntityClassOpening(EntityType entity)
    {
        return string.Format(
            CultureInfo.InvariantCulture,
            "{0} {1}partial class {2}{3}",
            Accessibility.ForType(entity),
            _code.SpaceAfter(_code.AbstractOption(entity)),
            _code.Escape(entity),
            _code.StringBefore(" : ", _typeMapper.GetTypeName(entity.BaseType)));
    }
```

این کد را نیز به این صورت تغییر دهید:

```
public string EntityClassOpening(EntityType entity)
    {
        return string.Format(
            CultureInfo.InvariantCulture,
"[DataContract]" + Environment.NewLine +
            "{0} {1}partial class {2}{3}",
            Accessibility.ForType(entity),
            _code.SpaceAfter(_code.AbstractOption(entity)),
            _code.Escape(entity),
            _code.StringBefore(" : ", _typeMapper.GetTypeName(entity.BaseType)));
    }
```

برای واپسین تغییر به دنبال کد زیر بگردید:

```
public string UsingDirectives(bool inHeader, bool includeCollections = true)
    {
        return inHeader == string.IsNullOrEmpty(_code.VsNamespaceSuggestion())
            ? string.Format(
                CultureInfo.InvariantCulture,
                "{0}using System;{1}" +
                "{2}",
                inHeader ? Environment.NewLine : "",
                includeCollections ? (Environment.NewLine + "using System.Collections.Generic;") : "",
                inHeader ? "" : Environment.NewLine)
            : "";
    }
```

سپس کد زیر را جاگزین آن کنید:

```
public string UsingDirectives(bool inHeader, bool includeCollections = true)
    {
        return inHeader == string.IsNullOrEmpty(_code.VsNamespaceSuggestion())
            ? string.Format(
                CultureInfo.InvariantCulture,
"using System.Runtime.Serialization;" + Environment.NewLine +
```

```
                "{0}using System;{1}" +
                "{2}",
                inHeader ? Environment.NewLine : "",
                includeCollections ? (Environment.NewLine + "using System.Collections.Generic;") : "",
                inHeader ? "" : Environment.NewLine)
            : "";
    }
```

فایل MyNewsModel.tt را ذخیره کنید و از آن خارج شوید. بار دیگر هر کدام از کلاس‌های tblNews و tblCategory را باز کنید. خواهید دید که به صورت خودکار تغییرات مد نظر ما به آن افزوده شده است. از این پس بدون هیچ دلواپسی بابت حذف صفت‌ها، می‌توانید هرچند بار که خواستید مدل خود را به‌هنگام کنید.

در بخش پسین دوباره به WCF بازخواهیم گشت و به تعریف روال‌های مورد نیاز خواهیم پرداخت.

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۰/۲۶ ۱۴:۸

با تشکر از شما. روش دیگری برای حل مساله استفاده از AOP است:

استفاده از IL Code Weaving برای تولید ویژگی‌های تکراری مورد نیاز در WCF

نویسنده: حمید
تاریخ: ۱۳۹۲/۱۰/۲۷ ۴:۱

هرچند که به نکته خوبی، اشاره کردین اما این کار از اساس غلط است چون شما دارید کلاسهای لایه داده خود را expose می‌کنید. سرویس‌ها بادید DTOها را به بیرون EXPOSE کنند و تبدیل کلاسهای لایه BUSINESS به dtoها از طریق ابزاری مثل AUTOMAPPER انجام می‌شود. متشکرم

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۰/۲۷ ۹:۲۸

بایدی وجود ندارد در این حالت و بهتر است که اینگونه باشد یا حتی مخلوطی از این دو در عمل:

Pros and Cons of Data Transfer Objects

In large projects with so many entities, DTOs add a remarkable level of (extra) complexity and work to do. In short, a pure, 100% DTO solution is often just a 100 percent painful solution

عنوان: **ایجاد سرویس چندلایه‌ی WCF با Entity Framework در قالب پروژه - 4**
نویسنده: حامد قنادی
تاریخ: ۱۳۹۲/۱۰/۲۷ ۱۱:۴۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, LINQ, WCF

برای ادامه‌ی کار به لایه‌ی Interface بازمی‌گردیم. کلیه‌ی متدهایی که به آن نیاز داریم، نخست در این لایه تعریف می‌شود. در این‌جا نیز از قراردادهایی برای تعریف کلاس و روال‌های آن بهره می‌بریم که در ادامه به آن می‌پردازیم. پیش از آن باید بررسی کنیم، برای استفاده از این دو موجودیت، به چه متدهایی نیاز داریم. من گمان می‌کنم موارد زیر برای کار ما کافی باشد:

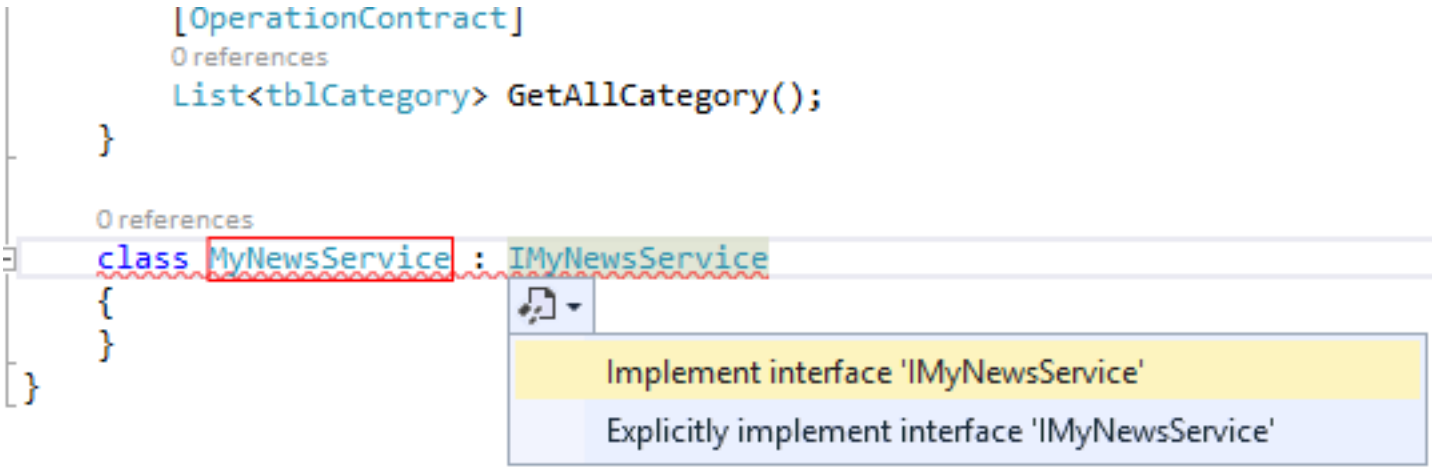
-1 نمایش کلیه‌ی رکوردهای جدول خبر
-2 انتخاب رکوردی از جدول خبر با پارامتر ورودی شناسه‌ی جدول خبر
-3 درج یک رکورد جدید در جدول خبر
-4 ویرایش یک رکورد از جدول خبر
-5 حذف یک رکورد از جدول خبر
-6 افزودن یک دسته
-7 حذف یک دسته
-8 نمایش دسته‌ها

هم‌اکنون به صورت زیر آن‌ها را تعریف کنید:

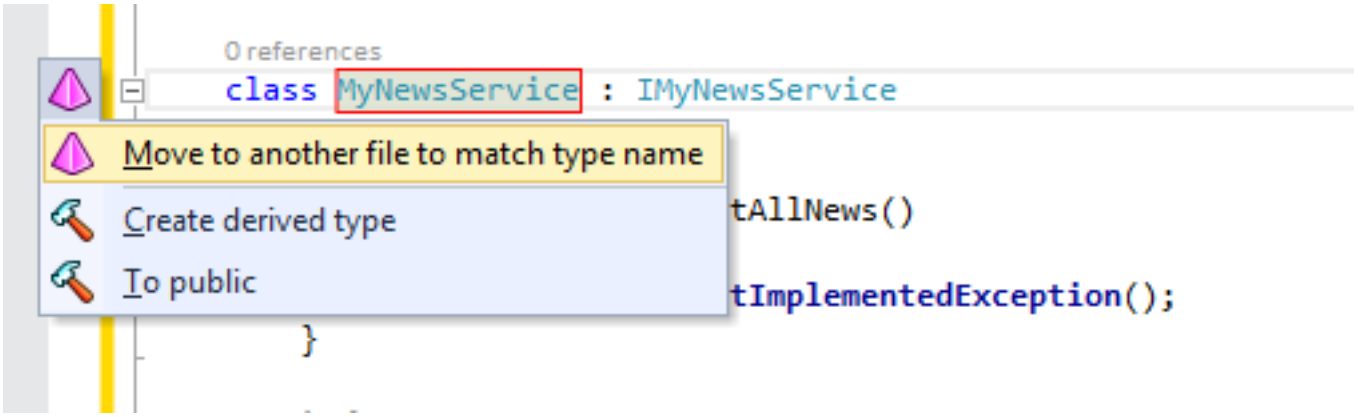
```
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel;
using System.Text;
using System.Threading.Tasks;

namespace MyNewsWCFLibrary
{
    [ServiceContract]
    interface IMyNewsService
    {
        [OperationContract]
        List<tblNews> GetAllNews();

        [OperationContract]
        tblNews GetNews(int tblNewsId);

        [OperationContract]
        int AddNews(tblNews News);

        [OperationContract]
        bool EditNews(tblNews News);

        [OperationContract]
        bool DeleteNews(int tblNewsId);

        [OperationContract]
        int AddCategory(tblCategory News);

        [OperationContract]
        bool DeleteCategory(int tblCategoryId);

        [OperationContract]
        List<tblCategory> GetAllCategory();
    }
}
```

همان‌گونه که مشاهده می‌کنید از دو قرارداد جدید ServiceContract و OperationContract در فضای نام System.ServiceModel بهره برده ایم. ServiceContract صفتی است که بر روی Interface اعمال می‌شود و تعیین می‌کند که مشتری چه فعالیت‌هایی را روی سرویس می‌تواند انجام دهد و OperationContract تعیین می‌کند، چه متدهایی در اختیار قرار خواهند گرفت. برای ادامه‌ی کار نیاز است تا کلاس اجرا را ایجاد کنیم. برای این‌کار از ابزار Resharper بهره خواهم برد: روی نام interface همانند شکل کلیک کنید و سپس برابر با شکل عمل کنید:

کلاسی به نام MyNewsService با ارث‌بری از IMyNewsService ایجاد می‌شود. زیر حرف I از IMyNewsService یک خط دیده می‌شود که با کلیک روی آن برابر با شکل زیر عمل کنید:



ملاحظه خواهید کرد که کلیه‌ی متدها برابر با Interface ساخته خواهد شد. اکنون همانند شکل روی نشان هرم شکلی که هنگامی که روی نام کلاس کلیک می‌کنید، در سمت چپ نشان داده می‌شود کلیک کنید و گزینه Move to another file to match type name را انتخاب کنید:



به صورت خودکار محتوای این کلاس به یک فایل دیگر انتقال می‌یابد. اکنون هر کدام از متدها را به شکل دلخواه ویرایش می‌کنیم.
من کد کلاس را این‌گونه تغییر دادم:

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

namespace MyNewsWCFLibrary
{
    class MyNewsService : IMyNewsService
    {
        private dbMyNewsEntities dbMyNews = new dbMyNewsEntities();
        public List<tblNews> GetAllNews()
        {
            return dbMyNews.tblNews.Where(p => p.IsDeleted == false).ToList();
        }

        public tblNews GetNews(int tblNewsId)
        {
            return dbMyNews.tblNews.FirstOrDefault(p => p.tblNewsId == tblNewsId);
        }

        public int AddNews(tblNews News)
        {
            dbMyNews.tblNews.Add(News);
            dbMyNews.SaveChanges();
            return News.tblNewsId;
        }

        public bool EditNews(tblNews News)
        {
            try
            {
                dbMyNews.Entry(News).State = EntityState.Modified;
                dbMyNews.SaveChanges();
                return true;
            }
            catch (Exception exp)
            {
                return false;
            }
        }

        public bool DeleteNews(int tblNewsId)
        {
            try
            {
                tblNews News = dbMyNews.tblNews.FirstOrDefault(p => p.tblNewsId == tblNewsId);
                News.IsDeleted = true;
                dbMyNews.SaveChanges();
            return true;
            }
            catch (Exception exp)
            {
                return false;
            }
        }
```

```
        public int AddCategory(tblCategory Category)
        {
            dbMyNews.tblCategory.Add(Category);
            dbMyNews.SaveChanges();
            return Category.tblCategoryId;
        }

        public bool DeleteCategory(int tblCategoryId)
        {
            try
            {
                tblCategory Category = dbMyNews.tblCategory.FirstOrDefault(p => p.tblCategoryId ==
tblCategoryId);
                Category.IsDeleted = true;
                dbMyNews.SaveChanges();
                return true;
            }
            catch (Exception exp)
            {
                return false;
            }
        }

        public List<tblCategory> GetAllCategory()
        {
            return dbMyNews.tblCategory.Where(p => p.IsDeleted == false).ToList();
        }
    }
}
```

ولی شما ممکن است درباره‌ی حذف، دوست داشته باشید رکوردها از پایگاه داده حذف شوند و نه این‌که با یک فیلد بولی آن‌ها را مدیریت کنید. در این صورت کد شما می‌تواند این‌گونه نوشته شود:

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

namespace MyNewsWCFLibrary
{
    class MyNewsService : IMyNewsService
    {
        private dbMyNewsEntities dbMyNews = new dbMyNewsEntities();
        public List<tblNews> GetAllNews()
        {
            return dbMyNews.tblNews.ToList();
        }

        public tblNews GetNews(int tblNewsId)
        {
            return dbMyNews.tblNews.FirstOrDefault(p => p.tblNewsId == tblNewsId);
        }

        public int AddNews(tblNews News)
        {
            dbMyNews.tblNews.Add(News);
            dbMyNews.SaveChanges();
            return News.tblNewsId;
        }

        public bool EditNews(tblNews News)
        {
            try
            {
                dbMyNews.Entry(News).State = EntityState.Modified;
                dbMyNews.SaveChanges();
                return true;
            }
            catch (Exception exp)
            {
                return false;
            }
        }

        public bool DeleteNews(tblNews News)
        {
```

```
            try
            {
                dbMyNews.tblNews.Remove(News);
                dbMyNews.SaveChanges();
            return true;
            }
            catch (Exception exp)
            {
                return false;
            }
        }

        public int AddCategory(tblCategory Category)
        {
            dbMyNews.tblCategory.Add(Category);
            dbMyNews.SaveChanges();
            return Category.tblCategoryId;
        }

        public bool DeleteCategory(tblCategory Category)
        {
            try
            {
                dbMyNews.tblCategory.Remove(Category);
                dbMyNews.SaveChanges();
                return true;
            }
            catch (Exception exp)
            {
                return false;
            }
        }

        public List<tblCategory> GetAllCategory()
        {
            return dbMyNews.tblCategory.ToList();
        }
    }
}
```

البته باید در نظر داشته باشید که در صورت هر گونه تغییر در پارامترهای ورودی، لایه‌ی Interface نیز باید تغییر کند. گونه‌ی دیگر نوشتن متد حذف خبر می‌تواند به صورت زیر باشد:

```
public bool DeleteNews(int tblNewsId)
        {
            try
            {
                tblNews News = dbMyNews.tblNews.FirstOrDefault(p => p.tblNewsId == tblNewsId);
                dbMyNews.tblNews.Remove(News);
                dbMyNews.SaveChanges();
            return true;
            }
            catch (Exception exp)
            {
                return false;
            }
        }
```

در بخش 5 درباره‌ی تغییرات App.Config خواهم نوشت.

عنوان: **تصادفی کردن آیتمهای لیست با استفاده از Extension Method**
نویسنده: بهزاد دات نت
تاریخ: ۱۳۹۳/۰۲/۰۷ ۱۴:۱۵
آدرس: www.dotnettips.info
گروه‌ها: LINQ, WPF, Extension Method, C#.NET

شاید برای شما هم پیش آمده باشد که بخواهید در هر بار واکشی لیستی از اطلاعات، مثلا از دیتابیس، آیتمهای آن را بصورت تصادفی مرتب کنید.
من در پروژه اخیرم برای نمایش یک سری سوال مجبور بودم که در هر بار نمایش سوالات، لیست را به صورت رندوم مرتب کنم و به کاربر نمایش بدم. برای حصول این مهم، یک extension method به شکل زیر نوشتم:

```
public static class RandomExtentions
    {
        public static void Shuffle<T>(this IList<T> list)
        {
            Random rng = new Random();
            Thread.Sleep(100);
            int n = list.Count;
            while (n > 1)
            {
                n--;
                int k = rng.Next(n + 1);
                T value = list[k];
                list[k] = list[n];
                list[n] = value;
            }
        }
    }
```

در این تابع که اسمش را Shuffle گذاشتم، با دریافت یک لیست از نوع T، آیتم‌های درون لیست را به صورت تصادفی مرتب می‌کند.

مثال :

```
var x =new  List<int>();
x.Add(1);
x.Add(2);
x.Add(3);
x.Add(4);
x.Add(5);
x.Shuffle();
```

در این مثال لیست x که از نوع int میباشد پس از فراخوانی Shuffle به یک لیست نامرتب تبدیل میشود که نحوه چیدمان در هر بار فراخوانی، تصادفی خواهد بود.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۰۷ ۱۴:۱۶

اگر از EF استفاده می‌کنید، برای اینکار یک ستون Guid پویا را اضافه می‌کند. سپس بر اساس این ستون، مرتب سازی را انجام می‌دهد. اطلاعات بیشتر

نویسنده: بهزاد دات نت
تاریخ: ۱۳۹۳/۰۲/۰۷ ۱۴:۳۰

با سپاس از شما. در صورت استفاده ار EF روشی که شما فرمودین بهتر و کارامدتر هستش.