عنوان: **نگاشت دیتای XML به کمک AutoMapper**
نویسنده: محمد صاحب
تاریخ: ۱۳۹۱/۰۷/۱۴ ۱۳:۲۱
آدرس: www.dotnettips.info
برچسب‌ها: AutoMapper, LINQ to XML

صورت مساله که مشخصه قراره دیتای رو از منبع داده‌ی Xml به model مورد نظرمون نگاشت کنیم چیزی شبیه کاری که متد GetEntries انجام میده و تو این پست معرفی شده...

AutoMapper به صورت داخلی و با استفاده از قراردادها نمیتونه xml رو به object تبدیل کنه ولی این کار به کمک LINQ to XML قابل انجامه.

مثالی که برای این پست انتخاب شده سوژه‌ی داغ روزهای اخیره ؟!
مدل زیر رو در نظر داشته باشید

```
public class PreciousMetal
    {
        public string Name { get; set; }
        public float Price { get; set; }
        public DateTime UpdateTime { get; set; }
    }
```

قراره از یک وب سرویس اطلاعات مربوط به فلزات گرانبها رو دریافت و به مدل PreciousMetal نگاشت کنیم.ساختار اطلاعات دریافتی ما به شکل زیره

```
<pricelist currency="usd">
  <price timestamp="1349347920" per="ozt" commodity="gold">1788.70</price>
  <price timestamp="1349347860" per="ozt" commodity="palladium">665.50</price>
  <price timestamp="1349347920" per="ozt" commodity="platinum">1701.25</price>
  <price timestamp="1349347920" per="ozt" commodity="silver">34.91</price>
</pricelist>
```

برای نگاشت‌های معمولی کار سختی نداریم و از MapFrom استفاده میکنیم مثلا برای قیمت

```
Mapper.CreateMap<XElement, PreciousMetal>().ForMember(des => des.Price,
                                                       op =>
                                                       op.MapFrom(src => src.Value));
```

ولی برای زمان دریافت قیمت با توجه به متفاوت بودن زمان دریافتی مثلا در اینجا Unix time از Custom value resolvers استفاده میکنیم

```
public class UnixTimestampResolver : ValueResolver<XElement, DateTime>
    {
        protected override DateTime ResolveCore(XElement source)
        {
            var origin = new DateTime(1970, 1, 1, 0, 0, 0, 0);
            return origin.AddSeconds(Convert.ToDouble(source.Attribute("timestamp").Value));
        }
    }
```

همچنیا میخواهیم از معادل فارسی نام فلزات گرانبها استفاده کنیم

```
public class EnglishPMetalToFarsiResolver : ValueResolver<XElement, string>
    {
        readonly Dictionary<string, string> _pMetaldic = new Dictionary<string, string>
                                                         {
                                                             {"gold", "طلا"},
                                                             {"palladium", "پالادیوم"},
                                                             {"platinum", "پلاتین "},
                                                             {"silver", "نقره"}
```

```
                                                    };
        protected override string ResolveCore(XElement source)
        {
            string pMetalFarsi;
            return _pMetaldic.TryGetValue(source.Attribute("commodity").Value, out pMetalFarsi) ?
pMetalFarsi : string.Empty;
        }
    }
```

نکته: از سری قبلی آشنایی با AutoMapper همیشه بین انتخاب Custom Value Formatters و Custom value resolvers مشکل داشتم مثلا همین قسمت بنظر خودم Custom Value Formatters مناسبتر میاد بعد کمی وقت گذاشتن مشخص شد گویا یه جورایی Custom Value Formatters اضافه س و اشتباه تو طراحی بوده.

و اما نحوه استفاده

```
static void Main(string[] args)
        {
            //تعریف نگاشت‌ها Mapper.CreateMap<XElement, PreciousMetal>().ForMember(des => des.Name,
                                                                    op =>
op.ResolveUsing<EnglishPMetalToFarsiResolver>())
                .ForMember(des => des.Price,
                           op =>
                           op.MapFrom(src => src.Value))

                .ForMember(des => des.UpdateTime, op => op.ResolveUsing<UnixTimestampResolver>());

            Mapper.AssertConfigurationIsValid();

            //دریافت قیمت‌ها از منبع داده
            var doc = XDocument.Load("http://www.xmlcharts.com/cache/precious-metals.xml");
            var priceData = doc.Descendants("pricelist").Take(1).Elements("price");

            //فراخوانی نگاشت
            var preciousMetals = Mapper.Map<IEnumerable<XElement>, IList<PreciousMetal>>(priceData);


            foreach (var preciousMetal in preciousMetals)
            {
                Console.WriteLine(preciousMetal.Name + " " + preciousMetal.Price + " " +
preciousMetal.UpdateTime.ToShortDateString());
            }

            Console.ReadLine();

        }
```

عنوان: **تهیه خروجی XML از یک بانک اطلاعاتی، توسط EF Code first**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۱۰ ۱۱:۳۵
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework, xml, LINQ to XML

نگارش کامل SQL Server امکان تهیه خروجی XML از یک بانک اطلاعاتی را دارد . اما اگر بخواهیم از سایر بانک‌های اطلاعاتی که چنین توابع توکاری ندارند، استفاده کنیم چطور؟ برای تهیه خروجی XML توسط Entity framework و مستقل از نوع بانک اطلاعاتی در حال استفاده، حداقل دو روش وجود دارد:

الف) استفاده از امکانات Serialization توکار دات نت

```
using System.IO;
using System.Xml;
using System.Xml.Serialization;

namespace DNTViewer.Common.Toolkit
{
    public static class Serializer
    {
        public static string Serialize<T>(T type)
        {
            var serializer = new XmlSerializer(type.GetType());
            using (var stream = new MemoryStream())
            {
                serializer.Serialize(stream, type);
                stream.Seek(0, SeekOrigin.Begin);
                using (var reader = new StreamReader(stream))
                {
                    return reader.ReadToEnd();
                }
            }
        }
    }
}
```

در اینجا برای نمونه، لیستی از اشیاء مدنظر خود را تهیه کرده و به متد Serialize فوق ارسال کنید. نتیجه کار، تهیه معادل XML آن است.
امکانات سفارشی سازی محدودی نیز برای XmlSerializer درنظر گرفته شده است؛ برای نمونه قرار دادن ویژگی‌هایی مانند XmlIgnore بالای خواصی که نیازی به حضور آن‌ها در خروجی نهایی XML نمی‌باشد.

ب) استفاده از امکانات LINQ to XML دات نت

روش فوق بدون مشکل کار می‌کند، اما اگر بخواهیم قسمت Reflection خودکار ثانویه آن‌را (برای نمونه جهت استخراج مقادیر از لیست دریافتی) حذف کنیم، می‌توان از LINQ to XML استفاده کرد که قابلیت سفارشی سازی بیشتری را نیز در اختیار ما قرار می‌دهد (کاری که در سایت جاری برای تهیه خروجی XML از بانک اطلاعاتی آن انجام می‌شود).

```
private string createXmlFile(string dir)
        {
            var xLinq = new XElement("ArrayOfPost",
                        _blogPosts
                            .AsNoTracking()
                            .Include(x => x.Comments)
                            .Include(x => x.User)
                            .Include(x => x.Tags)
                            .OrderBy(x => x.Id)
                            .ToList()
                            .Select(x => new XElement("Post", postXElement(x)))
                            );

            var xmlFile = Path.Combine(dir, "dot-net-tips-database.xml");
            xLinq.Save(xmlFile);
```

```
            return xmlFile;
        }

        private static XElement[] postXElement(BlogPost x)
        {
            return new XElement[]
            {
                new XElement("Id", x.Id),
                new XElement("Title", x.Title),
                new XElement("Body", x.Body),
                new XElement("CreatedOn", x.CreatedOn),
                tagElement(x),
                new XElement("User",
                                new XElement("Id", x.UserId.Value),
                                new XElement("FriendlyName", x.User.FriendlyName))
            }.Where(item => item != null).ToArray();
        }

        private static XElement tagElement(BlogPost x)
        {
            var tags = x.Tags.Any() ?
                            x.Tags.Select(y =>
                                        new XElement("Tag",
                                                new XElement("Id", y.Id),
                                                new XElement("Name", y.Name)))
                                    .ToArray() : null;
            if (tags == null)
                return null;

            return new XElement("Tags", tags);
        }
```

خلاصه‌ای از نحوه تبدیل اطلاعات لیستی از مطالب را به معادل XML آن در کدهای فوق مشاهده می‌کنید. یک سری نکات ریز نیز باید در اینجا رعایت شوند:

1) کار با یک new XElement که دارای متد Save با فرمت XML نیز هست، شروع می‌شود. مقدار آن‌را مساوی یک کوئری از بانک اطلاعاتی قرار می‌دهیم. این کوئری چون قرار است تنها اطلاعاتی را از بانک اطلاعاتی دریافت کند و نیازی به تغییر در آن‌ها نیست، با استفاده از متد AsNoTracking ، حالت فقط خواندنی پیدا کرده است.

2) اطلاعاتی را که نیاز است در فایل نهایی XML وجود داشته باشند، تنها کافی است در قسمت Select این کوئری با فرمت new XElementهای تو در تو قرار دهیم. به این ترتیب قسمت Relection خودکار XmlSerializer روش مطرح شده در ابتدای بحث دیگر وجود نداشته و عملیات نهایی بسیار سریعتر خواهد بود.

3) چون در این حالت، کار انجام شده دستی است، باید نام‌های گره‌های صحیحی را انتخاب کنیم تا اگر قرار است توسط همان XmlSerializer مجددا کار serializer.Deserialize صورت گیرد، عملیات با شکست مواجه نشود. بهترین کار برای کم شدن سعی و خطاها، تهیه یک لیست اطلاعات آزمایشی و سپس ارسال آن به روش ابتدای بحث است. سپس می‌توان با بررسی خروجی آن مثلا دریافت که روش serializer.Deserialize به صورت پیش فرض به دنبال ریشه‌ای به نام ArrayOfPost برای دریافت لیستی از مطالب می‌گردد و نه Posts یا هر نام دیگری.

4) در کوئری LINQ to Entites نوشته شده، پیش از Select، یک ToList قرار دارد. متاسفانه EF اجازه استفاده مستقیم از Select هایی از نوع XElement را نمی‌دهد و باید ابتدا اطلاعات را تبدیل به LINQ to Objects کرد.

5) در حین تهیه XElementها اگر قرار است عنصری نال باشد، باید آن‌را در خروجی نهایی ذکر نکرد. به این ترتیب serializer.Deserialize بدون نیاز به تنظیمات اضافه‌تری بدون مشکل کار خواهد کرد. در غیراینصورت باید وارد مباحثی مانند تعریف یک فضای نام جدید برای خروجی XML به نام XSI رفت و سپس به کمک ویژگی‌ها، xsi:nil را به true مقدار دهی کرد. اما همانطور که در متد postXElement ملاحظه می‌کنید، برای وارد نشدن به مبحث فضای نام xsi، مواردی که null بوده‌اند، اصلا در آرایه نهایی ظاهر نمی‌شوند و نهایتا در خروجی، حضور نخواهند داشت. به این ترتیب متد ذیل، بدون مشکل و بدون نیاز به تنظیمات اضافه‌تری قادر است فایل XML نهایی را تبدیل به معادل اشیاء دات نتی آن کند.

```
using System.IO;
using System.Xml;
using System.Xml.Serialization;

namespace DNTViewer.Common.Toolkit
{
    public static class Serializer
```

```
    {
        public static T DeserializePath<T>(string xmlAddress)
        {
            using (var xmlReader = new XmlTextReader(xmlAddress))
                {
                    var serializer = new XmlSerializer(typeof(T));
                    return (T)serializer.Deserialize(xmlReader);
                }
        }
    }
}
```

عنوان: **ثبت لینک‌های غیرتکراری**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۱۰ ۲۰:۱۵
آدرس: www.dotnettips.info
گروه‌ها: xml, LINQ to XML, OPML

ثبت لینک‌های مختلف در یک سیستم (مثلا قسمت به اشتراک گذاری لینک‌ها) در ابتدای کار شاید ساده به نظر برسد؛ خوب، هر صفحه‌ای که یک آدرس منحصربفرد بیشتر ندارد. ما هش این لینک را محاسبه می‌کنیم و بعد روی این هش، یک کلید منحصربفرد را تعریف خواهیم کرد تا دیگر رکوردی تکراری ثبت نشود. همچنین چون این هش نیز طول کوتاهی دارد، جستجوی آن بسیار سریع خواهد بود. واقعیت این است که خیر! این روش ناکارآمدترین حالت پردازش لینک‌های مختلف است.

برای مثال لینک‌های http://www.site.com و http://www.site.com/index.htm دو هش متفاوت را تولید می‌کنند اما در عمل یکی هستند. نمونه‌ی دیگر، لینک‌های http://www.site.com/index.htm و http://www.site.com/index.htm#section1 هستند که فقط اصطلاحا در یک fragment با هم تفاوت دارند و از این دست لینک‌هایی که باید در حین ثبت یکی درنظر گرفته شوند، زیاد هستند و اگر علاقمند به مرور آن‌ها هستید، می‌توانید به صفحه‌ی URL Normalization در ویکی‌پدیا مراجعه کنید.

اگر نکات این صفحه را تبدیل به یک کلاس کمکی کنیم، به کلاس ذیل خواهیم رسید:

```
using System;
using System.Web;

namespace OPMLCleaner
{
    public static class UrlNormalization
    {
        public static bool AreTheSameUrls(this string url1, string url2)
        {
            url1 = url1.NormalizeUrl();
            url2 = url2.NormalizeUrl();
            return url1.Equals(url2);
        }

        public static bool AreTheSameUrls(this Uri uri1, Uri uri2)
        {
            var url1 = uri1.NormalizeUrl();
            var url2 = uri2.NormalizeUrl();
            return url1.Equals(url2);
        }

        public static string[] DefaultDirectoryIndexes = new[]
            {
                "default.asp",
                "default.aspx",
                "index.htm",
                "index.html",
                "index.php"
            };

        public static string NormalizeUrl(this Uri uri)
        {
            var url = urlToLower(uri);
            url = limitProtocols(url);
            url = removeDefaultDirectoryIndexes(url);
            url = removeTheFragment(url);
            url = removeDuplicateSlashes(url);
            url = addWww(url);
            url = removeFeedburnerPart(url);
            return removeTrailingSlashAndEmptyQuery(url);
        }

        public static string NormalizeUrl(this string url)
        {
            return NormalizeUrl(new Uri(url));
        }

        private static string removeFeedburnerPart(string url)
        {
            var idx = url.IndexOf("utm_source=", StringComparison.Ordinal);
            return idx == -1 ? url : url.Substring(0, idx - 1);
        }

        private static string addWww(string url)
        {
            if (new Uri(url).Host.Split('.').Length == 2 && !url.Contains("://www."))
```

```
            {
                return url.Replace("://", "://www.");
            }
            return url;
        }

        private static string removeDuplicateSlashes(string url)
        {
            var path = new Uri(url).AbsolutePath;
            return path.Contains("//") ? url.Replace(path, path.Replace("//", "/")) : url;
        }

        private static string limitProtocols(string url)
        {
            return new Uri(url).Scheme == "https" ? url.Replace("https://", "http://") : url;
        }

        private static string removeTheFragment(string url)
        {
            var fragment = new Uri(url).Fragment;
            return string.IsNullOrWhiteSpace(fragment) ? url : url.Replace(fragment, string.Empty);
        }

        private static string urlToLower(Uri uri)
        {
            return HttpUtility.UrlDecode(uri.AbsoluteUri.ToLowerInvariant());
        }

        private static string removeTrailingSlashAndEmptyQuery(string url)
        {
            return url
                    .TrimEnd(new[] { '?' })
                    .TrimEnd(new[] { '/' });
        }

        private static string removeDefaultDirectoryIndexes(string url)
        {
            foreach (var index in DefaultDirectoryIndexes)
            {
                if (url.EndsWith(index))
                {
                    url = url.TrimEnd(index.ToCharArray());
                    break;
                }
            }
            return url;
        }
    }
}
```

از این روش برای تمیز کردن و حذف فیدهای تکراری در فایل‌های OPML تهیه شده نیز می‌شود استفاده کرد. عموما فیدخوان‌های نه‌چندان با سابقه، نکات یاد شده در این مطلب را رعایت نمی‌کنند و به سادگی می‌شود در این سیستم‌ها، فیدهای تکراری زیادی را ثبت کرد.

برای مثال اگر یک فایل OPML چنین ساختار XML ایی را داشته باشد:

```
<?xml version="1.0" encoding="utf-8"?>
<opml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" version="1.0">
  <body>
    <outline text="آی تی ایرانی">
    <outline type="rss"
                          text="فید کلی آخرین نظرات، مطالب، اشتراک‌ها و پروژه‌های .NET Tips"
                          title="فید کلی آخرین نظرات، مطالب، اشتراک‌ها و پروژه‌های .NET Tips"
                          xmlUrl="http://www.dotnettips.info/Feed/LatestChanges"
                          htmlUrl="http://www.dotnettips.info/" />
    </outline>
  </body>
</opml>
```

هر outline آن‌را به کلاس زیر می‌توان نگاشت کرد:

```
using System.Xml.Serialization;
```

```
namespace OPMLCleaner
{
    [XmlType(TypeName="outline")]
    public class Opml
    {
        [XmlAttribute(AttributeName="text")]
        public string Text { get; set; }

        [XmlAttribute(AttributeName = "title")]
        public string Title { get; set; }

        [XmlAttribute(AttributeName = "type")]
        public string Type { get; set; }

        [XmlAttribute(AttributeName = "xmlUrl")]
        public string XmlUrl { get; set; }

        [XmlAttribute(AttributeName = "htmlUrl")]
        public string HtmlUrl { get; set; }
    }
}
```

برای اینکار فقط کافی است از LINQ to XML به نحو ذیل استفاده کنیم:

```
var document = XDocument.Load("it-92-03-01.opml");
var results = (from node in document.Descendants("outline")
                            where node.Attribute("htmlUrl") != null && node.Parent.Attribute("text") !=
null
                             && node.Parent.Attribute("text").Value == "آی تی ایرانی"
                            select new Opml
                            {
                                HtmlUrl = (string)node.Attribute("htmlUrl"),
                                Text = (string)node.Attribute("text"),
                                Title = (string)node.Attribute("title"),
                                Type = (string)node.Attribute("type"),
                                XmlUrl = (string)node.Attribute("xmlUrl")
                            }).ToList();
```

در این حالت لیست کلیه فیدهای یک گروه را چه تکراری و غیرتکراری، دریافت خواهیم کرد. برای حذف موارد تکراری نیاز است از متد Distinct استفاده شود. به همین جهت باید کلاس ذیل را نیز تدارک دید:

```
using System.Collections.Generic;

namespace OPMLCleaner
{
    public class OpmlCompare : EqualityComparer<Opml>
    {
        public override bool Equals(Opml x, Opml y)
        {
            return UrlNormalization.AreTheSameUrls(x.HtmlUrl, y.HtmlUrl);
        }

        public override int GetHashCode(Opml obj)
        {
            return obj.HtmlUrl.GetHashCode();
        }
    }
}
```

اکنون با کمک کلاس OpmlCompare فوق که از کلاس UrlNormalization برای تشخیص لینک‌های تکراری استفاده می‌کند، می‌توان به لیست بهتر و متعادل‌تری رسید:

```
var distinctResults = results.Distinct(new OpmlCompare()).ToList();
```