عنوان: **Lazy loading در تزریق وابستگی‌ها به کمک StructureMap**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۲۷ ۰:۷
آدرس: www.dotnettips.info
گروه‌ها: Design patterns, Dependency Injection, IoC

پیشنیاز این بحث، مطلب «استفاده از StructureMap به عنوان یک IoC Container» می‌باشد که پیشتر در این سری مطالعه کردید (در حد نحوه نصب StructureMap و آشنایی با تنظیمات اولیه آن)

ابتدا ساختار بحث جاری را به نحو زیر درنظر بگیرید:

```
namespace DI04.Services
{
    public interface IAccounting
    {
        void CreateInvoice(int orderId, int count);
    }
}

namespace DI04.Services
{
    public interface ISales
    {
        bool ShippingAllowed(int orderId);
    }
}

namespace DI04.Services
{
    public interface IOrderHandler
    {
        void Handle(int orderId, int count);
    }
}

using System;

namespace DI04.Services
{
    public class Accounting : IAccounting
    {
        public Accounting()
        {
            Console.WriteLine("Accounting ctor.");
        }

        public void CreateInvoice(int orderId, int count)
        {
            // ...
        }
    }
}

using System;

namespace DI04.Services
{
    public class Sales : ISales
    {
        public Sales()
        {
            Console.WriteLine("Sales ctor.");
        }

        public bool ShippingAllowed(int orderId)
        {
            // فقط جهت آزمایش سیستم
            return false;
        }
    }
}

using System;

namespace DI04.Services
{
```

```
    public class OrderHandler : IOrderHandler
    {
        private readonly IAccounting _accounting;
        private readonly ISales _sales;
        public OrderHandler(IAccounting accounting, ISales sales)
        {
            Console.WriteLine("OrderHandler ctor.");
            _accounting = accounting;
            _sales = sales;
        }

        public void Handle(int orderId, int count)
        {
            if (_sales.ShippingAllowed(orderId))
            {
                _accounting.CreateInvoice(orderId, count);
            }
        }
    }
}
```

در اینجا کار مدیریت سفارشات در کلاس OrderHandler انجام می‌شود. این کلاس دارای دو وابستگی تزریق شده در سازنده کلاس می‌باشد.

در متد Handle، اگر مجوز کار توسط متد ShippingAllowed صادر شد، آنگاه کار نهایی توسط متد CreateInvoice باید صورت گیرد. با توجه به اینکه تزریق وابستگی‌ها در سازنده کلاس صورت می‌گیرد، نیاز است پیش از وهله سازی کلاس OrderHandler، هر دو وابستگی آن وهله سازی و تزریق شوند. در حالیکه در مثال جاری هیچگاه به وهله‌ای از نوع IAccounting نیاز نخواهد شد؛ زیرا متد ShippingAllowed در این مثال، فقط false بر می‌گرداند.

و از این نمونه‌ها زیاد هستند. کلاس‌هایی با تعداد متدهای بالا و تعداد وابستگی‌های قابل توجه که ممکن است در طول عمر شیء وهله سازی شده این کلاس، تنها به یکی از این وابستگی‌ها نیاز شود و نه به تمام آن‌ها.

راه حلی برای این مساله در دات نت 4 با معرفی کلاس Lazy ارائه شده است؛ به این نحو که اگر برای مثال در اینجا accounting را Lazy تعریف کنیم، تنها زمانی وهله سازی خواهد شد که به آن نیاز باشد و نه پیش از آن.

```
private readonly Lazy<IAccounting> _accounting;
```

سؤال: Lazy loading تزریق وابستگی‌ها را چگونه می‌توان توسط StructureMap فعال ساخت؟

ابتدا تعاریف کلاس OrderHandlerرا به نحو زیر بازنویسی می‌کنیم:

```
using System;

namespace DI04.Services
{
    public class OrderHandlerLazy : IOrderHandler
    {
        private readonly Lazy<IAccounting> _accounting;
        private readonly Lazy<ISales> _sales;
        public OrderHandlerLazy(Lazy<IAccounting> accounting, Lazy<ISales> sales)
        {
            Console.WriteLine("OrderHandlerLazy ctor.");
            _accounting = accounting;
            _sales = sales;
        }

        public void Handle(int orderId, int count)
        {
            if (_sales.Value.ShippingAllowed(orderId))
            {
                _accounting.Value.CreateInvoice(orderId, count);
            }
        }
    }
}
```

در اینجا سازنده‌های کلاس، به صورت Lazy معرفی شده‌اند. دسترسی به فیلدهای sales و accounting نیز اندکی تغییر کرده‌اند و اینبار از طریق خاصیت Value آن‌ها باید انجام شود.
مرحله نهایی هم اندکی تغییر در نحوه معرفی تنظیمات اولیه StructureMap است:

```
using System;
using DI04.Services;
using StructureMap;

namespace DI04
{
    class Program
    {
        static void Main(string[] args)
        {
            // تنظیمات اولیه برنامه که فقط یکبار باید در طول عمر برنامه انجام شود
            ObjectFactory.Initialize(x =>
            {
                x.For<IOrderHandler>().Use<OrderHandlerLazy>();

                // Lazy loading
                x.For<Lazy<IAccounting>>().Use(c => new Lazy<IAccounting>(c.GetInstance<Accounting>));
                x.For<Lazy<ISales>>().Use(c => new Lazy<ISales>(c.GetInstance<Sales>));
            });

            var orderHandler = ObjectFactory.GetInstance<IOrderHandler>();
            orderHandler.Handle(orderId: 1, count: 10);
        }
    }
}
```

به این ترتیب زمانیکه برنامه به sales.Value می‌رسد آنگاه نیاز به وهله سازی شیء متناظر با آن‌را خواهد داشت که در اینجا از طریق متد GetInstance به آن ارسال خواهد گردید.

خروجی برنامه در این حالت:

```
OrderHandlerLazy ctor.
Sales ctor.
```

همانطور که مشاهده می‌کنید، هرچند کلاس OrderHandlerLazy دارای دو وابستگی تعریف شده در سازنده کلاس است، اما تنها وابستگی Sales آن زمانیکه به آن نیاز شده، وهله سازی گردیده است و خبری از وهله سازی کلاس Accounting نیست (چون مطابق تعاریف کلاس‌های برنامه هیچگاه به مسیر accounting.Value نخواهیم رسید؛ بنابراین نیازی هم به وهله سازی آن نخواهد بود).

دریافت مثال این قسمت

DI04.zip

نظرات خوانندگان

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۱۰/۱۷ ۱۳:۳۴

سلام
یک سوال
من در یک برنامه MVC
چند کلاس دارم که در سازنده‌های آن کلاس‌های دیگر به صورت lazy تزریق میشود.
حال زمانی که کلاس مورد نظر فراخوانی می‌شود با خطای 202 به منزله عدم وجود سازنده پیش فرض مواجه می‌شوم در حالی که تمامی کلاسها را به صورت lazy یه StructureMap معرفی کرده ام.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۱۷ ۱۴:۱۲

خطای 202 به معنای ناقص بودن سیم‌کشی‌های آغازین برنامه شما است. نمونه‌اش در بحث مرتبط با MVC مطرح شده‌است .

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۱۰/۱۸ ۱۲:۲۸

سلام
سیم کشی‌های من درست بود
پارامترهارو در یک کلاس کپسوله کردم و به کنترلر پاس دادم درست شد. ظاهرا خطای غیر منطقی هست چون هیچ چیزی تغییر نکرد

نویسنده: عباسپور
تاریخ: ۱۳۹۳/۱۱/۱۴ ۱۱:۵۹

چگونه می‌توان تمام کلاس‌های لایه سرویس را بصورت Lazy معرفی کرد تا مجبور نشویم تک تک آنها را در موقع شروع برنامه Initialize کنیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۱/۱۴ ۱۲:۳۹

در مثال فوق، تنظیمات دستی را حذف کنید و آن‌را تبدیل کنید به :

```
x.Scan(scanner =>
{
    scanner.AssemblyContainingType<IOrderHandler>();

    // connects `IAccounting` to `Accounting` and `ISales` to `Sales` automatically.
    scanner.WithDefaultConventions();
});
```

مثال کامل

نویسنده: ح مرادف
تاریخ: ۱۳۹۳/۱۱/۱۶ ۱۹:۵۳

با سلام،
بنده در لایه سرویس در کد زیر به مشکل خوردم :

```
public class JobSubCategoryService : IJobSubCategoryService
```

```
{
       private readonly Lazy<IUnitOfWork> _uow;
        private readonly Lazy<IDbSet<JobSubCategory>> _jobSubCategories;
        public JobSubCategoryService(Lazy<IUnitOfWork> uow)
        {
            _uow = uow;
            _jobSubCategories = uow.Value.Set<JobSubCategory>();  // i have problem here
        }

    // some methods here ...
}
```

سوال بنده اینه که آیا لازمه خود کانتکست رو هم بصورت Lazy نماییم ؟
سوال دوم بنده این است که آیا در حالت Lazy روشی برای خودکار کردن معرفی کلاس‌ها و اینترفیس‌ها به استراکچر مپ وجود دارد (شما در پایان مقاله جاری بصورت دستی معرفی نموده اید ...)؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۱/۱۶ ۲۰:۳

- آیا وابستگی تزریق شده، در تمام متدهای آن کلاس استفاده می‌شود؟ اگر بله، خیر؛ نیازی نیست. اگر خیر، «بهتر است» به صورت lazy تعریف شود.
- بله. کمی بالاتر پاسخ دادم. مثال کامل آن برای اجرا و بررسی بیشتر.

نویسنده: ح مرداف
تاریخ: ۱۳۹۳/۱۱/۱۶ ۲۱:۳۳

اینطور که شما می‌فرمایید ، می‌توان نتیجه گرفت که کدهای این بخش فرقی با حالت غیر Lazy ندارد و روال مثل گذشته است و تنها تفاوت در کلاس‌های سرویس می‌باشد.
(البته طبق فایل معرفی شده در گیت هاب ، گویا در بخش ابتدایی کلاس SmObjectFactory تغییراتی داریم)

سوالی که پیش میاد اینه که اگر نیاز باشه در یک کلاس خود کلاس کانتکس رو Lazy کنیم ، آیا کدنویسی بصورت زیر درون کلاس سرویس درست است :

```
private readonly Lazy<IUnitOfWork> _uow;
private readonly IDbSet<JobCategory> _jobCategories;
public JobCategoryService(Lazy<IUnitOfWork> uow)
{
     _uow = uow;
     _jobCategories = _uow.Value.Set<JobCategory>();
}
```

یا اینکه کد زیر را باید در متدی که مورد نیاز است بنویسیم ؟

```
_jobCategories = _uow.Value.Set<JobCategory>();
```

طبق فرمایشات شما به نظرم روش اول نادرست باشه ؛ درسته ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۱/۱۶ ۲۲:۵۰

- بله. StructureMap 3.x بدون مشکل با سازنده‌های Lazy کار می‌کند و نیازی به تنظیمات اضافه‌تری ندارد.
- اگر uow در تمام متدهای کلاس جاری استفاده می‌شود، نیازی نیست Lazy تعریف شود. اگر خیر، روش دومی که نوشتید، در صورت نیاز سبب وهله سازی آن خواهد شد. مورد تنظیم شده در سازنده، عملا تفاوتی با حالت معمولی ندارد؛ چون بلافاصله

سبب وهله سازی آن می‌شود (اولین تماس با خاصیت Value، آن‌را وهله سازی می‌کند).

نویسنده: امین کاشانی
تاریخ: ۱۳۹۳/۱۱/۱۶ ۲۳:۴۹

در حالت lazy

```
public interface IUnitOfWork
{
    Lazy<IDbSet<TEntity>> LazySet<TEntity>() where TEntity : class;
    int SaveChanges();
}
```

در کلاس context که از کلاس IUnitOfWork ارث بری کرده پیاده سازی متد

```
Lazy <IDbSet<TEntity>> LazySet<TEntity>()  where  TEntity :  class
```

ایراد می‌دهد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۱/۱۶ ۲۳:۵۹

- بله. چون در تعریف قبلی آن، متد Set در کلاس پایه DbContext از قبل موجود بود و پیاده سازی شده بود. به همین جهت نیازی به پیاده سازی مجدد آن نبود. بدیهی است هر تعریف جدید دیگری را که اضافه کنید، خودتان هم باید مطابق معمول روال کار با اینترفیس‌ها، پیاده سازی آن‌را به کلاس Context خودتان اضافه کنید.
- ضمنا در اینجا Lazy تعریف کردن یک Set غیرضروری است. این Set فقط به یک جدول از بانک اطلاعاتی اشاره می‌کند و جزئی از کوئری LINQ نوشته شده خواهد بود. اگر قرار است چیزی را Lazy تعریف کنید، uow Lazy<IUnitOfWork> در سازنده‌ی یک کلاس خواهد بود. کل شیء و نه یک خاصیت از آن. زمانیکه Uow وهله سازی می‌شود، تمام Setهای آن در دسترس هستند و Lazy تعریف کردن آن‌ها در اینجا فایده‌ای ندارد.
- همچنین EF برای Setها مباحث Lazy loading خاص خودش را دارد و از این بحث جدا است.