

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 در نظر 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 بر 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 OrderHandler را 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: ۱۳:۳۴ ۱۳۹۲/۱۰/۱۷

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 به StructureMap معرفی کرده‌ام.

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

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

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

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