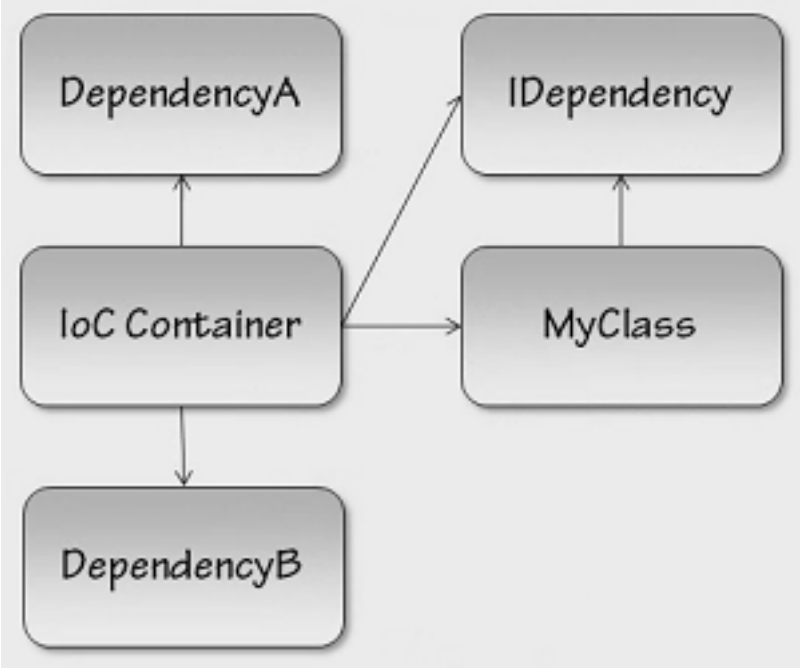
عنوان: **آشنایی با نحوه ایجاد یک IoC Container**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۲۵ ۲۳:۲۱
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns, Dependency Injection, IoC

قبل از اینکه وارد بحث استفاده از کتابخانه‌های بسیار غنی IoC Container موجود شویم، بهتر است یک نمونه ساده آن‌ها را طراحی کنیم تا بهتر بتوان با عملکرد و ساختار درونی آن‌ها آشنا شد.

IoC Container چیست؟

IoC Container، فریم ورکی است برای انجام تزریق وابستگی‌ها. در این فریم ورک امکان تنظیم اولیه وابستگی‌های سیستم وجود دارد. برای مثال زمانیکه برنامه از یک IoC Container، نوع اینترفیس خاصی را درخواست می‌کند، این فریم ورک با توجه به تنظیمات اولیه‌اش، کلاسی مشخص را بازگشت خواهد داد.
IoC Containerهای قدیمی‌تر، برای انجام تنظیمات اولیه خود از فایل‌های کانفیگ استفاده می‌کردند. نمونه‌های جدیدتر آن‌ها از روش‌های Fluent interfaces برای مشخص سازی تنظیمات خود بهره می‌برند.

زمانیکه از یک IOC Container در کدهای خود استفاده می‌کنید، مراحلی چند رخ خواهند داد:
الف) کد فراخوان، از IOC Container، یک شیء مشخص را درخواست می‌کند. عموما اینکار با درخواست یک اینترفیس صورت می‌گیرد؛ هرچند محدودیتی نیز نداشته و امکان درخواست یک کلاس از نوعی مشخص نیز وجود دارد.
ب) در ادامه IOC Container به لیست اشیاء قابل ارائه توسط خود نگاه کرده و در صورت وجود، وهله سازی شیء درخواست شده را انجام و نهایتا شیء مطلوب را بازگشت خواهد داد.
در این بین زنجیره‌ی وابستگی‌های مورد نیاز نیز وهله سازی خواهند شد. برای مثال اگر وابستگی اول به وابستگی دوم برای وهله سازی نیاز دارد، کار وهله سازی وابستگی‌های وابستگی دوم نیز به صورت خودکار انجام خواهند شد. (این موردی است که بسیاری از تازه واردان به این بحث تا یکبار آن‌را امتحان نکنند باور نخواهند کرد!)
ج) سپس کد فراخوان وهله دریافتی را مورد پردازش قرار داده و سپس شروع به استفاده از متدها و خواص آن خواهد نمود.



در تصویر فوق محل قرارگیری یک IOC Container را مشاهده می‌کنید. یک IOC Container در مورد تمام وابستگی‌های مورد نیاز،

اطلاعات لازم را دارد. همچنین این فریم ورک در مورد کلاسی که قرار است از وابستگی‌های سیستم استفاده نماید نیز مطلع است؛ به این ترتیب می‌تواند به صورت خودکار در زمان وهله سازی آن، نوع‌های وابستگی‌های مورد نیاز آن‌را در اختیارش قرار دهد.
برای مثال در اینجا MyClass، وابستگی مشخص شده در سازنده خود را به نام IDependency از IOC Container درخواست می‌کند. سپس این IOC Container بر اساس تنظیمات اولیه خود، یکی از وابستگی‌های A یا B را بازگشت خواهد داد.

آغاز به کار ساخت یک IOC Container نمونه

در ابتدا کدهای آغازین مثال بحث جاری را در نظر بگیرید:

```
using System;

namespace DI01
{
    public interface ICreditCard
    {
        string Charge();
    }

    public class Visa : ICreditCard
    {
        public string Charge()
        {
            return "Charging with the Visa!";
        }
    }

    public class MasterCard : ICreditCard
    {
        public string Charge()
        {
            return "Swiping the MasterCard!";
        }
    }

    public class Shopper
    {
        private readonly ICreditCard creditCard;

        public Shopper(ICreditCard creditCard)
        {
            this.creditCard = creditCard;
        }

        public void Charge()
        {
            var chargeMessage = creditCard.Charge();
            Console.WriteLine(chargeMessage);
        }
    }
}
```

در اینجا وابستگی‌های کلاس خریدار از طریق سازنده آن که متداول‌ترین روش تزریق وابستگی‌ها است، در اختیار آن قرار خواهد گرفت. یک اینترفیس کردیت کارت تعریف شده‌است به همراه دو پیاده سازی نمونه آن مانند مسترکارت و ویزا کارت. ساده‌ترین نوع فراخوانی آن نیز می‌تواند مانند کدهای ذیل باشد (تزریق وابستگی‌های دستی):

```
var shopper = new Shopper(new Visa());
shopper.Charge();
```

در ادامه قصد داریم این فراخوانی‌ها را اندکی هوشمندتر کنیم تا بتوان بر اساس تنظیمات برنامه، کار تزریق وابستگی‌ها صورت گیرد و به سادگی بتوان اینترفیس‌های متفاوتی را در اینجا درخواست و مورد استفاده قرار داد. اینجا است که به اولین IoC Container خود خواهیم رسید:

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

```
namespace DI01
{
    public class Resolver
    {
        //کار ذخیره سازی و نگاشت از یک نوع به نوعی دیگر در اینجا توسط این دیکشنری انجام خواهد شد
        private Dictionary<Type, Type> dependencyMap = new Dictionary<Type, Type>();

        /// <summary>
        /// یک نوع خاص از آن درخواست شده و سپس بر اساس تنظیمات برنامه، کار وهله سازی
        /// نمونه معادل آن صورت خواهد گرفت
        /// </summary>
        public T Resolve<T>()
        {
            return (T)Resolve(typeof(T));
        }

        private object Resolve(Type typeToResolve)
        {
            Type resolvedType;

            // ابتدا بررسی می‌شود که آیا در تنظیمات برنامه نگاشت متناظری برای نوع درخواستی وجود دارد؟
            if (!dependencyMap.TryGetValue(typeToResolve, out resolvedType))
            {
                //اگر خیر، کار متوقف خواهد شد
                throw new Exception(string.Format("Could not resolve type {0}",
typeToResolve.FullName));
            }

            var firstConstructor = resolvedType.GetConstructors().First();
            var constructorParameters = firstConstructor.GetParameters();
            // در ادامه اگر این نوع، دارای سازنده‌ی بدون پارامتری است
            // بلافاصله وهله سازی خواهد شد
            if (!constructorParameters.Any())
                return Activator.CreateInstance(resolvedType);


            var parameters = new List<object>();
            foreach (var parameterToResolve in constructorParameters)
            {
                // در اینجا یک فراخوانی بازگشتی صورت گرفته است برای وهله سازی
                // خودکار پارامترهای مختلف سازنده یک کلاس
                parameters.Add(Resolve(parameterToResolve.ParameterType));
            }
            return firstConstructor.Invoke(parameters.ToArray());
        }

        public void Register<TFrom, TTo>()
        {
            dependencyMap.Add(typeof(TFrom), typeof(TTo));
        }
    }
}
```

در اینجا کدهای کلاس Resolver یا همان IoC Container ابتدایی بحث را مشاهده می‌کنید. توضیحات قسمت‌های مختلف آن به صورت کامنت ارائه شده‌اند.

```
var resolver = new Resolver();
//تنظیمات اولیه
resolver.Register<Shopper, Shopper>();
resolver.Register<ICreditCard, Visa>();
//تزریق وابستگی‌ها و وهله سازی
var shopper = resolver.Resolve<Shopper>();
shopper.Charge();
```

در ادامه نحوه استفاده از IoC Container ایجاد شده را مشاهده می‌کنید.
ابتدا کار تعاریف نگاشت‌های اولیه انجام می‌شود. در این صورت زمانیکه متد Resolve فراخوانی می‌گردد، نوع درخواستی آن به همراه سازنده دارای آرگومانی از نوع ICreditCard وهله سازی شده و بازگشت داده خواهد شد. سپس با در دست داشتن یک وهله آماده، متد Charge آن‌را فراخوانی خواهیم کرد.

بررسی نحوه استفاده از Microsoft Unity به عنوان یک IoC Container

Unity چیست؟

Unity یک فریم ورک IoC Container تهیه شده توسط مایکروسافت می‌باشد که آن‌را به عنوان جزئی از Enterprise Library خود قرار داده است. بنابراین برای دریافت آن یا می‌توان کل مجموعه Enterprise Library را دریافت کرد و یا به صورت مجزا به عنوان یک بسته نیوگت نیز قابل تهیه است.
برای این منظور در خط فرمان پاورشل نیوگت در VS.NET دستور ذیل را اجرا کنید:

```
PM> Install-Package Unity
```

پیاده سازی مثال خریدار توسط Unity

همان مثال قسمت قبل را درنظر بگیرید. قصد داریم اینبار بجای IoC Container دست سازی که تهیه شد، پیاده سازی آن‌را به کمک MS Unity انجام دهیم.

```
using Microsoft.Practices.Unity;

namespace DI02
{
    class Program
    {
        static void Main(string[] args)
        {
            var container = new UnityContainer();

            container.RegisterType<ICreditCard, MasterCard>();

            var shopper = container.Resolve<Shopper>();
            shopper.Charge();
        }
    }
}
```

همانطور که ملاحظه می‌کنید، API آن بسیار شبیه به کلاس دست سازی است که در قسمت قبل تهیه کردیم.
مطابق کدهای فوق، ابتدا تنظیمات IoC Container انجام شده است. به آن اعلام کرده‌ایم که در صورت نیاز به ICreditCard، نوع MasterCard را یافته و وهله سازی کن. با این تفاوت که Unity هوشمندتر بوده و سطر مربوط به ثبت کلاس Shoper ایی را که در قسمت قبل انجام دادیم، در اینجا حذف شده است.
سپس به این IoC Container اعلام کرده‌ایم که نیاز به یک وهله از کلاس خریدار داریم. در اینجا Unity کار وهله سازی‌های خودکار وابستگی‌ها و تزریق آن‌ها را در سازنده کلاس خریدار انجام داده و نهایتا یک وهله قابل استفاده را در اختیار ادامه برنامه قرار خواهد داد.

یک نکته:
به صورت پیش فرض کار تزریق وابستگی‌ها در سازنده کلاس‌ها به صورت خودکار انجام می‌شود. اگر نیاز به Setter injection و مقدار دهی خواص کلاس وجود داشت می‌توان به نحو ذیل عمل کرد:

```
container.RegisterType<ICreditCard, MasterCard>(new InjectionProperty("propertyName", 5));
```

نام خاصیت و مقدار مورد نظر به عنوان پارامتر متد RegisterType باید تعریف شوند.

مدیریت طول عمر اشیاء در Unity

توسط یک IoC Container می‌توان یک وهله معمولی از شیءایی را درخواست کرد و یا حتی طول عمر این وهله را به صورت Singleton معرفی نمود (یک وهله در طول عمر کل برنامه). در Unity اگر تنظیم خاصی اعمال نشود، هربار که متد Resolve

فراخوانی می‌گردد، یک وهله جدید را در اختیار ما قرار خواهد داد. اما اگر پارامتر متد RegisterType را با وهله‌ای از ContainerControlledLifetimeManager مقدار دهی کنیم:

```
container.RegisterType<ICreditCard, MasterCard>(new ContainerControlledLifetimeManager());
```

از این پس با هربار فراخوانی متد Resolve، در صورت نیاز به وابستگی از نوع ICreditCard، تنها یک وهله مشترک از MasterCard ارائه خواهد شد.
حالت پیش فرض مورد استفاده، بدون ذکر پارامتر متد RegisterType، مقدار TransientLifetimeManager می‌باشد.

نظرات خوانندگان

نویسنده: مهدی فرهانی
تاریخ: ۱۳۹۲/۰۱/۲۶ ۱:۶

به نظر شما از بین فریم ورک‌ها موجود کدام یک بهتره ؟ مخصوصاً مقایسه ای بین Unity ، Ninject و StructerMap اگر داشته باشیم خیلی بهتره

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۲۶ ۱:۱۳

من StructureMap رو ترجیح می‌دم. خیلی‌ها هم همین نظر رو دارند:

IoC libraries compared

Which .NET Dependency Injection frameworks are worth looking into

نویسنده: سیروس
تاریخ: ۱۳۹۲/۰۱/۲۸ ۱۸:۲۸

با وجود اینکه ما خودمان می‌تونیم مانند کد زیر کار وهله سازی را انجام دهیم

```
var shopper = new Shopper(new Visa());
shopper.Charge();
```

چه لزومی به استفاده از IoC Container و کد

```
var resolver = new Resolver();
//تنظیمات اولیه
resolver.Register<Shopper, Shopper>();
resolver.Register<ICreditCard, Visa>();
//تزریق وابستگی‌ها و وهله سازی
var shopper = resolver.Resolve<Shopper>();
shopper.Charge();
```

وجود دارد، شاید بگید : " اگر وابستگی اول به وابستگی دوم برای وهله سازی نیاز دارد، کار وهله سازی وابستگی‌های وابستگی دوم نیز به صورت خودکار انجام خواهند شد. " میشه یه مثال ملموس‌تر بزنیم.
من یه خورده گیج شدم!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۲۸ ۲۰:۵۹

پاسخ به این سؤال نیاز به مطالعه قسمت‌های بعدی دارد.
- هدف از قسمت جاری آشنایی اولیه با مراحل ابتدایی کار با یک IoC Container است.
- در حالت اول هنوز شما هستید که مسئول وهله سازی‌های اولیه می‌باشید و کار وهله سازی را به لایه‌ای دیگر واگذار نکرده‌اید.
- در کد حالت اول نمی‌شود این وابستگی ارسالی به سازنده کلاس را به سادگی تعویض کرد. در حالیکه با استفاده از یک IoC container فقط کافی است تنظیمات اولیه نگاشت‌های آن‌را مشخص کنیم تا نوع کلاسی که باید در سازنده‌ها تزریق شوند مشخص شود. مزیت اینکار ساده‌تر شدن نوشتن آزمون‌های واحد و تهیه کلاس‌های Fake است؛ بدون نیازی به تغییری حتی در حد یک سطر در کدهای اصلی برنامه.
- در مورد وهله سازی خودکار چند سطح وابستگی‌ها، در قسمت‌های بعد تحت عنوان Object graph بیشتر بحث شده است و مثال زده شده. همیشه با یک کلاس ساده ویزا مانند مثال فوق سر و کار نداریم. عموما با سرویس‌هایی سر و کار داریم که خودشان نیز از سرویس‌های دیگری استفاده می‌کنند. برای مثال یک سرویس ارسال ایمیل از سرویس کاربران برای دریافت

ایمیل‌های کاربران کمک می‌گیرد. وهله سازی تمام این وابستگی‌ها را در چند سطح می‌شود با استفاده از IoC Containers خودکار کرد و به کدهای نهایی بسیار تمیزتری رسید.

- در حالت اول، طول عمر یک شیء را نمی‌شود مشخص کرد (یا حداقل نیاز به کد نویسی قابل توجهی دارد). برای مثال با استفاده از یک IoC Container می‌شود وهله ایجاد شده را Singleton کرد تا در سراسر برنامه یک وهله از آن استفاده شود یا حتی می‌شود در طول یک درخواست رسیده وب، یک وهله را در اختیار تمام کلاس‌های درگیر قرار داد. به این ترتیب به سربار کمتری در حالت‌های خاصی مانند وهله سازی ObjectContext یا DbContext در EF خواهیم رسید.

- زمان استفاده از IoC Containerها کارهای فراتری از تزریق وابستگی‌ها را هم می‌شود انجام داد. برای مثال فراخوانی‌های متدها را هم تحت نظر قرار داد (برنامه نویسی AOP یا جنبه گرا) و مثلا بدون نوشتن کد اضافه‌ای در برنامه، خروجی متدها را کش کرد. AOP یک سری بحث مفصل را در طی یک دوره جدا به همراه دارد.