عنوان: **بایدها و نبایدهای استفاده از IoC Containers**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۲۵ ۲۳:۳۳
آدرس: www.dotnettips.info
گروه‌ها: Design patterns, Dependency Injection, IoC

طوری با IoC Containers کار کنید که انگار وجود خارجی ندارند

تفاوت پایه‌ای که بین یک فریم ورک IoC و سایر فریم ورک‌ها وجود دارد، در معکوس شدن مسئولیت‌ها است. در اینجا لایه‌های مختلف برنامه شما نیستند که فریم ورک IoC را فراخوانی می‌کنند؛ بلکه این فریم ورک IoC است که از جزئیات ارتباطات و وابستگی‌های سیستم شما آگاه است و نهایتا کار کنترل وهله سازی اشیاء مختلف را عهده دار خواهد شد. طول عمر آن‌ها را تنظیم کرده یا حتی در بعضی از موارد مانند برنامه نویسی جنبه‌گرا یا AOP، نسبت به تزئین این اشیاء یا دخالت در مراحل مختلف فراخوانی متدهای آن‌ها نیز نقش خواهد داشت. نکته‌ی مهم در اینجا، نا آگاهی برنامه از حضور آن‌ها است.
بنابراین در پروژه شما اگر ماژول‌ها و لایه‌های مختلفی حضور دارند، تنها برنامه اصلی است که باید ارجاعی را به فریم ورک IoC داشته باشد و نه سایر لایه‌های سیستم. علت حضور آن در ریشه سیستم نیز تنها باید به اصطلاحا bootstrapping و اعمال تنظیمات مرتبط با آن خلاصه شود.
به عبارتی استفاده صحیح از یک فریم ورک IoC نباید به شکل الگوی Service Locator باشد؛ حالتی که در تمام قسمت‌های برنامه مدام مشاهده می‌کنید resolver.Resolve، resolver.Resolve و الی آخر. باید از این نوع استفاده از فریم ورک‌های IoC تا حد ممکن حذر شود و کدهای برنامه نباید وابستگی مستقیم ثانویه‌ای را به نام خود فریم ورک IoC پیدا کنند.

```
var container = BootstrapContainer();

var finder = container.Resolve<IDuplicateFinder>();
var processor = container.Resolve<IArgumentsParser>();

Execute( args, processor, finder );

container.Dispose();
```

نمونه‌ای از نحوه صحیح استفاده از یک IoC Container را مشاهده می‌کنید. تنها در سه نقطه است که یک IoC container باید حضور پیدا کند:
الف) در آغاز برنامه برای اعمال تنظیمات اولیه و bootstrapping
ب) پیش از اجرای عملی جهت وهله سازی وابستگی‌های مورد نیاز
ج) پس از اجرای عمل مورد نظر جهت آزاد سازی منابع

نکته مهم اینجا است که در حین اجرای فرآیند، این فرآیند باید تا حد ممکن از حضور IoC container بی‌خبر باشد و کار تشکیل اشیاء باید خارج از منطق تجاری برنامه انجام شود: IoC container خود را صدا نزنید؛ او شما را صدا خواهد زد.
عنوان شد تا «حد ممکن». این تا حد ممکن به چه معنایی است؟ اگر کار وهله سازی اشیاء را می‌توانید تحت کنترل قرار دهید، مثلا آیا می‌توانید در نحوه وهله سازی کنترلرها در ASP.NET MVC دخل و تصرف کرده و در زمان وهله سازی، اینکار را به یک IoC Container واگذار کنید؟ اگر بلی، دیگر به هیچ عنوانی نباید داخل کلاس‌های فراخوانی شده و تزریق شده به کنترلرهای برنامه اثری از IoC Container شما مشاهده شود. زیرا این فریم ورک‌ها اینقدر توانمند هستند که بتوانند تا چندین لایه از سیستم را واکاوی کرده و وابستگی‌های لازم را وهله سازی کنند.
اگر خیر (نمی‌توانید کار وهله سازی اشیاء را مستقیما تحت کنترل قرار دهید)؛ مانند تهیه یک Role Provider سفارشی در ASP.NET MVC که کار وهله سازی این Role Provider راسا توسط موتور ASP.NET انجام می‌شود و در این بین امکان دخل و تصرفی هم در آن ممکن نیست، آنگاه مجاز است داخل این کلاس ویژه از متدهای container.Resolve استفاده کرد؛ چون چاره‌ی دیگری وجود ندارد و IoC Container نیست که کار وهله سازی ابتدایی آن‌را عهده دار شده است. باید دقت داشت به این حالت خاص دیگر تزریق وابستگی‌ها گفته نمی‌شود؛ بلکه نام الگوی آن Service locator است. در Service locator یک کامپوننت خودش به دنبال وابستگی‌های مورد نیازش می‌گردد. در حالت تزریق وابستگی‌ها، یک کامپوننت وابستگی‌های مورد نیاز را درخواست می‌کند.

یک مثال:

```
public class ExampleClass
{
    private readonly IService _service;

    public ExampleClass()
    {
        _service = Container.Resolve<IService>();
    }

    public void DoSomething(int id)
    {
        _service.DoSomething(id);
    }
}
```

کاری که در اینجا انجام شده است نمونه اشتباهی از استفاده از یک IoC Container می‌باشد. به صرف اینکه مشغول به استفاده از یک IoC Container هستیم به این معنا نیست که واقعا الگوی معکوس سازی وابستگی‌ها را درست درک کرده‌ایم. در اینجا الگوی Service locator مورد استفاده است و نه الگوی تزریق وابستگی‌ها. به عبارتی در مثال فوق، کلاس ExampleClass وابسته است به یک وابستگی جدیدی به نام Container، علاوه بر وابستگی IService ایی که به او قرار است خدماتی را ارائه دهد.
نمونه اصلاح شده کلاس فوق، تزریق وابستگی‌ها در سازنده کلاس به نحو زیر است:

```
public class ExampleClass
{
    private IService _service;

    public ExampleClass(IService service)
    {
        _service = service;
    }

    public void DoSomething(int id)
    {
        _service.DoSomething(id);
    }
}
```

در اینجا این کلاس است که وابستگی‌های خود را درخواست می‌کند و نه اینکه خودش به دنبال آن‌ها بگردد.

نمونه دیگری از کلاسی که خودش به دنبال یافتن و وهله سازی وابستگی‌های مورد نیازش است مثال زیر می‌باشد:

```
public class Search
{
   IDinner _dinner;
   public Search(): this(new Dinner())
   { }

   public Search(IDinner dinner)
   {
      _dinner = dinner;
   }
}
```

به این کار poor man's dependency injection هم گفته می‌شود؛ اولین سازنده از طریق یک default constructor سعی کرده است وابستگی‌های کلاس را، خودش تامین کند. باز هم کلاس می‌داند که به چه وابستگی خاصی نیاز دارد و عملا معکوس سازی وابستگی‌ها رخ نداده است. همچنین استفاده از این حالت زمانیکه کلاس Dinner خودش وابستگی به کلاس‌های دیگر داشته باشد، بسیار به هم ریخته و مشکل خواهد بود. مزیت استفاده از IoC Containers وهله سازی یک large object graph کامل است. به علاوه توسط IoC Containers مدیریت طول عمر اشیاء را نیز می‌توان تحت نظر قرار داد. برای مثال می‌توان به یک IoC Container گفت تنها یک وهله از DbContext را در طول یک درخواست ایجاد و آن‌را در اختیار لایه‌های مختلف برنامه قرار بده؛ چون نیاز داریم کاری که در طی یک درخواست انجام می‌شود، در داخل یک تراکنش انجام شده و همچنین بی‌جهت به ازای هر new DbConetxt جدید، یکبار اتصالی به بانک اطلاعاتی باز و بسته نشود (سرعت بیشتر، سربار کمتر).

نظرات خوانندگان

نویسنده: مهدی فرهانی
تاریخ: ۱۳۹۲/۰۱/۲۶ ۱:۱۴

اگر ترکیبی از Service Locator و poor man's dependency injection استفاده شود چه ایراداتی دارد ؟
مثلاً این کد

```
protected readonly IUnitOfWork UnitOfWork;

        protected BaseOperation():this(ObjectFactory.GetInstance<IUnitOfWork>())
        {

        }
        protected BaseOperation(IUnitOfWork uow)
        {
            UnitOfWork = uow;
        }
```

به غیر از این که کلاس مورد نظر به Container وابسته هست آیا ایراد دیگری هم هست یا خیر ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۲۶ ۱:۲۰

- این بهتر است. مدیریت طول عمر اشیاء (مثلا ایجاد یک وهله در طی یک درخواست) و همچنین وهله سازی object graph در چند سطح به صورت خودکار توسط Service Locator هم انجام می‌شود.
- ولی در کل اگر امکان وهله سازی کلاس BaseOperation توسط IoC Container به صورت مستقل وجود دارد (چیزی مثل استفاده از DefaultControllerFactory در ASP.NET MVC) بهتر است اجازه بدید خود IoC Container کار تزریق وابستگی‌ها را به صورت خودکار انجام دهد و کلاس‌ها اطلاعی از وجود آن نداشته باشند.

نویسنده: مهدی فرهانی
تاریخ: ۱۳۹۲/۰۱/۲۶ ۱:۳۳

در اصل کلاس BaseOperation یک کلاس Abstract هست که بقیه Operationها از این کلاس ارثی بری میکنند.

```
public abstract class BaseOperation : IPartikanOperation
```

و هیچ وهله سازی مستقیمی از آن در برنامه صورت نمیگرد.

```
public class UserOperations : BaseOperation, IUserOperations
    {
        private readonly IUserService _userService;
        private readonly IMessageTemplateService _messageTemplateService;


        public UserOperations(IUserService userService, IMessageTemplateService messageTemplateService)
        {
            _userService = userService;
            _messageTemplateService = messageTemplateService;
        }
```

راه حلی که من استفاده کردم ، استفاده از پارمتر ورودی برای کلاس‌های فرزند هست

```
public UserOperations(IUnitOfWork uow,IUserService userService, IMessageTemplateService
messageTemplateService) : base(uow)
        {
            _userService = userService;
            _messageTemplateService = messageTemplateService;
        }
```

با توجه به اینکه هیچ وهله سازی از کلاس پایه صورت نمیگره ،آیا لزومی دارد که وابستگی به Container از کلاس پایه گرفته شود ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۲۶ ۹:۵۲

لطفا متن قسمت جاری را یکبار دیگر مطالعه بفرمائید. جواب صریحی را دریافت خواهید کرد.
(قسمت‌های وهله سازی خودکار وابستگی‌های کلاس‌های به هم وابسته (منظور از Object graph)؛ به علاوه امکان تعریف طول عمر یک شیء طوریکه هربار وهله سازی نشود (مثلا فقط در طول یک درخواست در تمام کلاس‌های وابسته به صورت یک وهله مشترک در دسترس باشد؛ مفید برای حالت استفاده از الگوی واحد کار). همچنین الگوی Service locator و فرق آن با تزریق وابستگی‌ها. مواردی که شاید یکی به نظر به رسند اما یکی نیستند)

نویسنده: رضا بزرگی
تاریخ: ۱۳۹۲/۰۲/۰۷ ۱:۲۶

لطفا در این مورد " *تفاوت پایه‌ای که بین یک فریم ورک IoC و سایر فریم ورک‌ها وجود دارد، در معکوس شدن مسئولیت‌ها است.* " بیشتر توضیح دهید.
- چه چیزی عامل برتری structuremap برای انتخاب شماست. و کلا چه تفاوت‌هایی با هم دارند. مثلا با ninject.
- آیا با توجه به ویژگی‌های جدید نسخه unity 3 که به تازگی منتشر شده و از طرفی بومی بودن اون، میشه گفت ارزش امتحان کردن داره یا خیر.
ممنونم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۰۷ ۸:۷

- هر دوره قسمت اختصاصی رو داره به نام « پرسش و پاسخ » برای طرح این نوع سؤالات خارج از موضوع مطلب جاری، اما مرتبط با عنوان دوره.
- در مورد معکوس شدن مسئولیت‌ها به تفصیل در سه قسمت اول این دوره مطلب نوشته شده است؛ پیش از شروع به کد نویسی.
- من StructureMap رو ترجیح می‌دم. خیلی‌ها هم همین نظر رو دارند:
IoC libraries compared
Which .NET Dependency Injection frameworks are worth looking into
- این مورد بیشتر سلیقه‌ای هست.

نویسنده: وحید م
تاریخ: ۱۳۹۲/۰۷/۱۹ ۲۳:۴۰

با سلام
بنده structuremap را از نوگت گرفتم پوشه ای برایم ایجاد شده که حاوی دوکلاس بود یکی IoC.cs و SmDependencyResolver.cs سوالی که داشتم آیا IOC همان servicelocator است.؟
آیا ObjectFactory.GetInstance همان کار servicelocator را انجام می‌دهد.؟
آیا var processor = container.Resolve<IArgumentsParser> هم همان کار servicelocator را انام می‌دهد

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۷/۱۹ ۲۳:۴۸

- خیر. به زبان ساده اگر وابستگی‌ها از طریق سازنده کلاس یا خواص آن در اختیار کلاس قرار گیرنده و در این بین ابزاری یا کتابخانه‌ای این تزریق را انجام دهد، به آن ابزار IoC Container می‌گویند. اگر در یک کلاس مستقیما از امکانات IoC Container

برای دریافت وابستگی‌ها استفاده شود، الگوی Service locator نام دارد و در این حالت خود IoC Container یک وابستگی در طراحی شما به حساب می‌آید.
- بله و خیر. بله اگر مستقیما داخل یک کلاس مثلا لایه سرویس یا یک کنترلر و امثال آن استفاده شود. اگر از آن در یک کلاس فکتوری مانند که کار وهله سازی مثلا کنترلرها و امثال آن را عهده دار است، استفاده شود دیگر الگوی Service locator نیست و تزریق وابستگی‌های استاندارد است.
- بله و خیر. مانند قبل.

نویسنده: ناظم
تاریخ: ۱۳۹۲/۱۱/۱۵ ۱۴:۱۳

با سلام؛ یعنی اگر من در یک برنامه mvc و در یک کنترلر، از IoC container مستقیما برای تولید اشیا استفاده کنم در واقع از تزریق وابستگی‌ها استفاده نمیکنم و دارم از الگوی Service locator استفاده میکنم؟

```
public partial class Test : Controller
    {
        private IUnitOfWork _uow;
        private IService _Service;

        public Test()
        {
            _uow = ObjectFactory.GetInstance<IUnitOfWork>();
            _userService = ObjectFactory.GetInstance<IService>();
        }

    // Other Methods
}
```

مثلا در کلاس Service که اصلا از IOC container مستقیم استفاده نشده، و هنگام ایجاد شی در کنترلر به صورت خودکار وابستگیش تامین میشود ، الگوی تزریق وابستگی‌ها درست پیاده شد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۱۵ ۱۴:۲۱

خیر. این روش service locator است و در MVC قابل بهبود است. یک مطلب کامل در مورد آن داریم:
« تزریق خودکار وابستگی‌ها در برنامه‌های ASP.NET MVC »

نویسنده: ناظم
تاریخ: ۱۳۹۲/۱۱/۱۵ ۱۴:۳۸

برای این که مطمئن بشم که آیا DI رو به صورت صحیح پیاده کردم یا نه آیا چک لیستی یا روشی برای این کار هست؟

برای مثال الان دارم فکر میکنم که چطور این کارو تو winForms میشه انجام داد، پس از انجام ، آیا کارم درست هست یا نه؟ روشی برای حصول اطمینان هست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۱۵ ۱۴:۴۱

مطالب و مفاهیم همین مقاله جاری و سطر اول آن یعنی «طوری با IoC Containers کار کنید که انگار وجود خارجی ندارند» کافی است.

نویسنده: حسابی
تاریخ: ۱۳۹۳/۱۰/۰۲ ۱۳:۱۶

برای مواردی که در پروژه پیش میاد و مجبور هستیم تا یک نمونه کلاس بدون پارامتر رو ایجاد کنیم چه راهکاری رو پیشنهاد

می‌کنید؟ چون عملاً اون کلاس نمی‌تونه وابستگی‌های خودش رو از طریق تابع سازنده اش اعلام کنه.

به عنوان مثال در کد زیر من می‌خوام یک Custom Route Constraint تعریف کنم:

```
routes.MapRoute(
   name: "PagesById",
   url: "Page/{id}",
   defaults: new { controller = "Route", action = "PageById", id = UrlParameter.Optional },
   constraints: new { id = new CustomPageByIdRoute() }
);
```

و اینجا نمی‌تونم پارامتری به کلاس CustomPageByIdRoute بدم . کلاس CustomPageByIdRoute به این صورت هست:

```
public class CustomPageByIdRoute : IRouteConstraint
    {
        private readonly IUnitOfWork _uow;
        private readonly IPage _page;

        public CustomPageByIdRoute(IUnitOfWork uow, IPage page)
        {
            _uow = uow;
            _page = page;
        }

        public bool Match(HttpContextBase httpContext, Route route, string parameterName,
RouteValueDictionary values, RouteDirection routeDirection)
        {
            return _page.FindBy(x => x.Id == Convert.ToInt32(values[parameterName]) && x.DeletedBy ==
0).FirstOrDefault() != null;
        }
    }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۰۲ ۱۳:۵۱

- همیشه بجای new MyClass () می‌توان نوشت ObjectFactory.GetInstance<MyClass <>>(). در این حالت به صورت خودکار تا n سطح، تمام وابستگی‌های MyClass به صورت خودکار وهله سازی می‌شوند.
- همچنین بحث مفصلی در مورد مسیریابی و تزریق وابستگی‌ها در ASP.NET MVC در اینجا: « Improving ASP.NET MVC Routing » Configuration

نویسنده: حسابی
تاریخ: ۱۳۹۳/۱۰/۰۳ ۱۴:۲۲

مرسی هم به خاطر پاسخ و هم به خاطر لینک، فقط باید حواسمون باشه که ObjectFactory در نسخه‌های بعدی structuremap کنار گذاشته می‌شه.