عنوان: آشنایی با AOP Interceptors
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۹ ۱۷:۱
آدرس: www.dotnettips.info
گروه‌ها: C#, Design patterns, AOP, Architecture

در حین استفاده از Interceptors، کار مداخله و تحت نظر قرار دادن قسمت‌های مختلف کدها، توسط کامپوننت‌های خارجی صورت خواهد گرفت. این کامپوننت‌های خارجی، به صورت پویا، تزئین کننده‌هایی را جهت محصور سازی قسمت‌های مختلف کدهای شما تولید می‌کنند. این‌ها، بسته به توانایی‌هایی که دارند، در زمان اجرا و یا حتی در زمان کامپایل نیز قابل تنظیم می‌باشند.

ابزارهایی جهت تولید AOP Interceptors

متداول‌ترین کامپوننت‌های خارجی که جهت تولید AOP Interceptors مورد استفاده قرار می‌گیرند، همان IOC Containers معروف هستند مانند MS Unity، Ninject، StructureMap و غیره.
سایر ابزارهای تولید AOP Interceptors، از روش تولید Dynamic proxies بهره می‌گیرند. به این ترتیب مزین کننده‌هایی پویا، در زمان اجرا، کدهای شما را محصور خواهند کرد. (نمونه‌ای از آن‌را شاید در حین کار با ORMهای مختلف دیده باشید).

نگاهی به فرآیند Interception

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 وجود 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.
ج) سپس، کد فراخوان، وهله دریافتی را مورد پردازش قرار داده و شروع به استفاده از متدها و خواص آن خواهد نمود.

اکنون با اضافه کردن Interception به این پروسه، چند مرحله دیگر نیز در این بین به آن اضافه خواهند شد:
الف) در اینجا نیز در ابتدا کد فراخوان، درخواست وهله‌ای را بر اساس اینترفیسی خاص به IOC Container ارائه می‌دهد.
ب) IOC Container نیز سعی در وهله سازی درخواست رسیده بر اساس تنظیمات اولیه خود می‌کند.
ج) اما در این حالت IOC Container تشخیص می‌دهد، نوعی که باید بازگشت دهد، علاوه بر وهله سازی، نیاز به مزین سازی توسط Aspects و پیاده سازی Interceptors را نیز دارد. بنابراین نوع مورد انتظار را در صورت وجود، به یک Dynamic Proxy، بجای بازگشت مستقیم به فراخوان ارائه می‌دهد.
د) در ادامه Dynamic Proxy، نوع مورد انتظار را توسط Interceptors محصور کرده و به فراخوان بازگشت می‌دهد.
ه) اکنون فراخوان، در حین استفاده از امکانات شیء وهله سازی شده، به صورت خودکار مراحل مختلف اجرای یک Aspect را که در قسمت قبل بررسی شدند، سبب خواهد شد.

نحوه ایجاد Interceptors

برای ایجاد یک Interceptor دو مرحله باید انجام شود:
الف) پیاده سازی یک اینترفیس
ب) اتصال آن به کدهای اصلی برنامه

در ادامه قصد داریم از یک IOC Container معروف به نام StructureMap در یک برنامه کنسول استفاده کنیم. برای دریافت آن نیاز است دستور پاورشل ذیل را در کنسول نوگت ویژوال استودیو فراخوانی کنید:

```
PM> Install-Package structuremap
```

پس از آن یک برنامه کنسول جدید را ایجاد کنید. (هدف از استفاده از این نوع پروژه خاص، توضیح جزئیات یک فناوری، بدون

درگیر شدن با لایه UI است)
البته باید دقت داشت که برای استفاده از StructureMap نیاز است به خواص پروژه مراجعه و سپس حالت Client profile را به Full profile تغییر داد تا برنامه قابل کامپایل باشد.

```
using System;
using StructureMap;

namespace AOP00
{
    public interface IMyType
    {
        void DoSomething(string data, int i);
    }

    public class MyType : IMyType
    {
        public void DoSomething(string data, int i)
        {
            Console.WriteLine("DoSomething({0}, {1});", data, i);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            ObjectFactory.Initialize(x =>
            {
                x.For<IMyType>().Use<MyType>();
            });

            var myType = ObjectFactory.GetInstance<IMyType>();
            myType.DoSomething("Test", 1);
        }
    }
}
```

اکنون کدهای این برنامه را به نحو فوق تغییر دهید.
در اینجا یک اینترفیس نمونه و پیاده سازی آن‌را ملاحظه می‌کنید. همچنین نحوه آغاز تنظیمات StructureMap و نحوه دریافت یک وهله متناظر با IMyType نیز بیان شده‌اند.
نکته‌ی مهمی که در اینجا باید به آن دقت داشت، وضعیت شیء myType حین فراخوانی متد myType.DoSomething است. شیء myType در اینجا، دقیقا یک وهله‌ی متداول از کلاس myType است و هیچگونه دخل و تصرفی در نحوه اجرای آن صورت نگرفته است.
خوب! تا اینجای کار را احتمالا پیشتر نیز دیده بودید. در ادامه قصد داریم یک Interceptor را طراحی و مراحل چهارگانه اجرای یک Aspect را در اینجا بررسی کنیم.

در ادامه نیاز خواهیم داشت تا یک Dynamic proxy را نیز مورد استفاده قرار دهیم؛ از این جهت که StructureMap تنها دارای Interceptorهای وهله سازی اطلاعات است و نه Method Interceptor. برای دسترسی به Method Interceptors نیاز به یک Dynamic proxy نیز می‌باشد. در اینجا از Castle.Core استفاده خواهیم کرد:

```
PM> Install-Package Castle.Core
```

برای دریافت آن تنها کافی است دستور پاور شل فوق را در خط فرمان کنسول پاورشل نوگت در VS.NET اجرا کنید.
سپس کلاس ذیل را به پروژه جاری اضافه کنید:

```
using System;
using Castle.DynamicProxy;

namespace AOP00
{
    public class LoggingInterceptor : IInterceptor
    {
        public void Intercept(IInvocation invocation)
        {
```

```
            try
            {
                Console.WriteLine("Logging On Start.");

                invocation.Proceed(); //فراخوانی متد اصلی در اینجا صورت می‌گیرد

                Console.WriteLine("Logging On Success.");
            }
            catch (Exception ex)
            {
                Console.WriteLine("Logging On Error.");
                throw;
            }
            finally
            {
                Console.WriteLine("Logging On Exit.");
            }
        }
    }
}
```

در کلاس فوق کار Method Interception توسط امکانات Castle.Core انجام شده است. این کلاس باید اینترفیس IInterceptor را پیاده سازی کند. در این متد سطر invocation.Proceed دقیقا معادل فراخوانی متد مورد نظر است. مراحل چهارگانه شروع، پایان، خطا و موفقیت نیز توسط try/catch/finally پیاده سازی شده‌اند.

اکنون برای معرفی این کلاس به برنامه کافی است سطرهای ذیل را اندکی ویرایش کنیم:

```
static void Main(string[] args)
        {
            ObjectFactory.Initialize(x =>
            {
                var dynamicProxy = new ProxyGenerator();
                x.For<IMyType>().Use<MyType>();
                x.For<IMyType>().EnrichAllWith(myTypeInterface =>
dynamicProxy.CreateInterfaceProxyWithTarget(myTypeInterface, new LoggingInterceptor()));
            });

            var myType = ObjectFactory.GetInstance<IMyType>();
            myType.DoSomething("Test", 1);
        }
```

در اینجا تنها سطر EnrichAllWith آن جدید است. ابتدا یک پروکسی پویا تولید شده است. سپس این پروکسی پویا کار دخالت و تحت نظر قرار دادن اجرای متدهای اینترفیس IMyType را عهده دار خواهد شد.
برای مثال اکنون با فراخوانی متد myType.DoSomething، ابتدا کنترل برنامه به پروکسی پویای تشکیل شده توسط Castle.Core منتقل می‌شود. در اینجا هنوز هم متد DoSomething فراخوانی نشده است. ابتدا وارد بدنه متد public void Intercept خواهیم شد. سپس سطر invocation.Proceed، فراخوانی واقعی متد DoSomething اصلی را انجام می‌دهد. در ادامه باز هم فرصت داریم تا مراحل موفقیت، خطا یا خروج را لاگ کنیم.
تنها زمانیکه کار متد public void Intercept به پایان می‌رسد، سطر پس از فراخوانی متد myType.DoSomething اجرا خواهد شد.
در این حالت اگر برنامه را اجرا کنیم، چنین خروجی را نمایش می‌دهد:

```
Logging On Start.
DoSomething(Test, 1);
Logging On Success.
Logging On Exit.
```

بنابراین در اینجا نحوه دخالت و تحت نظر قرار دادن اجرای متدهای یک کلاس عمومی خاص را ملاحظه می‌کنید. برای اینکه کنترل کامل را در دست بگیریم، کلاس پروکسی پویا وارد عمل شده و اینجا است که این کلاس پروکسی تصمیم می‌گیرد چه زمانی باید فراخوانی واقعی متد مورد نظر انجام شود.
برای اینکه فراخوانی قسمت On Error را نیز ملاحظه کنید، یک استثنای عمدی را در متد DoSomething قرار داده و مجددا برنامه را اجرا کنید.

نظرات خوانندگان

نویسنده: علی ملکی
تاریخ: ۱۳۹۲/۱۰/۰۹ ۱۱:۷

با سلام
در صورتی که بخواهیم یک Interceptor فقط برای لایه سرویس داشته باشیم چطور میتونیم هنگام رجیستر کردن یکباره (بدون نوشتن تک تک تایپ ها) این اینترسپتور(EnrichAllWith) رو اضافه کنیم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۰۹ ۱۴:۳۶

باید از قابلیت scan در StructureMap استفاده کنید:

```
ObjectFactory.Initialize(x =>
            {
                var dynamicProxy = new ProxyGenerator();
                x.Scan(scanner =>
                    {
                        scanner.AssemblyContainingType<IMyType>(); // نحوه یافتن اسمبلی لایه سرویس

                        // Connect `IName` interface to 'Name' class automatically
                        scanner.WithDefaultConventions()
                               .OnAddedPluginTypes(plugin => plugin.EnrichWith(target =>

dynamicProxy.CreateInterfaceProxyWithTargetInterface(target.GetType().GetInterfaces().First(),
                                                                    target.GetType().GetInterfaces(),
target,
                                                                    new LoggingInterceptor())
                                                    ));
                    });
            });
```

- در این حالت AssemblyContainingType مشخص می‌کند که کدام اسمبلی باید اسکن شود.
- WithDefaultConventions یعنی هرجایی IName داشتیم را به صورت خودکار به Name متصل کن. (روال پیش فرض سیم کشی اینترفیس‌ها و کلاس‌ها برای وهله سازی)
- OnAddedPluginTypes یک Callback هست که زمان انجام اولیه تنظیمات به ازای هر type یافت شده فراخوانی می‌شود. در اینجا می‌شود با استفاده از EnrichWith و ProxyGenerator کار اتصال کلاس Interceptor را انجام داد.

نویسنده: رضا شش
تاریخ: ۱۳۹۲/۱۰/۱۱ ۱۶:۲۹

برای اینکه استثای عمدی تولید کنم من از مثال ساده زیر استفاده کردم اما استثنا رخ نمی‌دهد. دلیل آن چیست؟
با تشکر Test_ExceptionAspect.zip

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۱۱ ۱۹:۶

قسمت EnrichAllWith را حذف کنید. بعد برنامه را اجرا کنید. باز هم اجرا می‌شود و استثنایی صادر نمی‌شود. چرا؟ چون اجرای کد آن معادل است با:

```
double d = 0;
Console.WriteLine(1 / d); // compiles, runs, results in: Infinity
```

مقدار infinity برای نوع double تعریف شده اما برای نوع int خیر؛ اینطوری طراحی شده .

نویسنده: رضا شش
تاریخ: ۱۳۹۲/۱۰/۱۵ ۱۰:۳

برای حالتهایی مثل حالتی که قرار است بر روی چند صد هزار رکورد ، محاسباتی صورت گیرد و نتیجه در دیتابیس ذخیره شود اگر بخواهیم یکسری کارها مثل لاگ و استثنا و ... را به درون اینترسپتر بکشانیم و از پروکسی استفاده کنیم آیا کارایی را پایین نمی‌آورد؟
در همین حالت اگر انتیتی‌های متفاوتی داشته باشیم و همزمان از انتیتی‌های مختلف نیاز به وهله سازی باشد چطور؟
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۱۵ ۱۰:۸

وجود یک Interceptor تداخلی در روند کاری جزئیات متد شما ندارد. جائیکه فراخوانی متد invocation.Proceed انجام می‌شود، روند انجام آن مستقل است از وجود Interceptor و فقط پیش و پس از آن یا استثنای حاصل تحت نظر قرار می‌گیرند.

نویسنده: رضا شش
تاریخ: ۱۳۹۲/۱۰/۱۵ ۱۴:۲۳

منظورم این بود که برای حالتی که از امکاناتی مثل Castle.Core استفاده می‌کنیم یک پروکسی از کلاس اصلی ما تولید می‌کند و با توجه به اینترسپتر در زمان اجرا این کلاس تزیین شده را اجرا می‌کند. مگر برای هر وهله از کلاس اصلی ما این اتفاق رخ نمی‌دهد؟ اگر چنین است پس پروکسی‌های زیادی با توجه به کلاس اصلی و اینترسپتورهای مختلفی که تعریف کرده ایم ایجاد می‌شود.
می‌خواستم ببینم این عملیات کارایی را پایین نمی‌آورد؟.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۱۵ ۱۴:۴۱

- سؤال شما این بود که در کلاس اصلی من، در متدی داخل آن، با چند صدهزار رکورد کار انجام می‌شود. پاسخ این است که اصلا این پروکسی ایجاد شده ربطی به داخل متد شما ندارد. کاری به وهله سازی‌های انجام شده داخل آن نیز ندارد.
invocation.Proceed یعنی این متد رو اجرا کن؛ نه اینکه هر وهله‌ای که داخل آن متد قرار می‌گیرد را نیز با پروکسی مزین کن.
- تمام ORMها برای پیاده سازی مباحث Lazy loading یک شیء پروکسی را از شیء اصلی شما ایجاد می‌کنند. نمونه‌اش را شاید با EF Code first با نام‌های خودکاری مانند ClassName_00394CF1F92740F13E3 دیده باشید؛ NHibernate هم یک زمانی از همین Castle.Core برای تدارک پروکسی‌های اشیاء استفاده می‌کرد. سربار آن در حین ایجاد چندین هزار وهله از یک شیء، در حد همان کار با ORMهایی است که هر روزه از آن‌ها استفاده می‌کنید (اگر می‌خواهید یک حسی از این قضیه داشته باشید).

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۱۳ ۱۸:۴۱

به روز رسانی
اگر از StructureMap نگارش 3 استفاده کنید، کلیه متدهای XYZ **Enrich** به XYZ **Decorate** تبدیل شده‌اند.

نویسنده: داستان
تاریخ: ۱۳۹۳/۰۱/۳۰ ۱۲:۳۵

ممنون بابت این دوره زیبا؛
ایا روشی توکار برای بازگشت مقدار، از یک Interceptor به متد اجراشونده هست؟
بعنوان مثال رشته ای که Log می‌شود رو بعنوان مقدار بازگشتی در متد اجرا شونده دریافت کنیم؟
باتشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۳۰ ۱۳:۴

- یکی از اهداف مهم AOP این است که به صورت لایه‌ای نامریی عمل کند و هر زمان که نیاز باشد، بتوان بدون کوچکترین تغییری در کدهای اصلی برنامه، کل منطق آن‌را حذف، یا با نمونه‌ای دیگر جایگزین کرد. بنابراین دریافت یک مقدار از Interceptor داخل متدی در برنامه، نقض کننده‌ی فلسفه‌ی وجودی این عملیات است.
- اما توسط پارامتر IInvocation و مقداری Reflection، دسترسی کاملی به اطلاعات متد فراخوان هست و در اینجا می‌توان در صورت نیاز، پارامتر و مقداری را نیز به آن ارسال کرد.
- در ASP.NET MVC، مفهوم فیلترها دقیقا پیاده سازی کننده‌ی Interceptorهای AOP هستند. در اینجا نیز مستقیما اطلاعاتی به فراخوان، در صورت نیاز بازگشت داده نمی‌شود. اما Context جاری در اختیار Interceptor و فیلتر هست. به این ترتیب Interceptor فرصت خواهد داشت به این Context مشترک، اطلاعاتی را اضافه کند یا تغییر دهد. مثلا به لیست خطاهای آن یک خطای اعتبارسنجی جدید را اضافه کند.

نویسنده: داستان
تاریخ: ۱۳۹۳/۰۲/۱۴ ۱۲:۲۸

بله،ممنون
جناب نصیری میشه یک مثال در ASP.NET MVC بزنید؟
مثلا پیاده سازی LoggerInterceptor برای اکشن هایی ک یک ActionResult برمیگردانند و درصورت بروز استثناء پیغامی مرتبط ب کاربر نمایش داده شود...
پیاپیش ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۱۴ ۱۲:۳۴

پیشنیازهای آن در سایت مطرح شده‌اند. ابتدا نیاز دارید تا تزریق وابستگی‌ها را در ASP.MVC پیاده سازی کنید . پس از اینکه کنترل وهله سازی یک کنترلر تماما در اختیار IoC Container قرار گرفت، سایر مباحث آن با مطلب جاری تفاوتی نمی‌کند و یکی است.
این یک راه حل است. راه دیگر آن استفاده از امکانات توکار خود ASP.NET MVC است و استفاده از فیلترهای آن که در حقیقت نوعی Interceptor توکار و یکپارچه هستند.

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۳/۱۰/۰۸ ۱۵:۱۰

با سلام.
امکان دارد نحوه معرفی کلاس‌های AOP را برای تمام Typeها و نه فقط برای یک Type خاص ، برای structuremap ورژن 3 راهنمایی کنید؟ سپاس.