عنوان: **آشنایی با AOP IL Weaving**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۲۱ ۹:۲۶
آدرس: www.dotnettips.info
گروه‌ها: C#, Design patterns, AOP, Architecture

IL Weaving در AOP به معنای اتصال Aspects تعریف شده، پس از کامپایل برنامه به فایل‌های باینری نهایی است. اینکار با ویرایش اسمبلی‌ها در سطح IL یا کد میانی صورت می‌گیرد. بنابراین در این حالت دیگر یک محصور کننده و پروکسی، در این بین جهت مزین سازی اشیاء، در زمان اجرای برنامه تشکیل نمی‌شود. بلکه فراخوانی Aspects به معنای فراخوانی واقعی قطعه کدهایی است که به اسمبلی‌های برنامه پس از کامپایل آن‌ها تزریق شده‌اند.
در دنیای دات نت، ابزارهای چندی امکان انجام IL Weaving را فراهم ساخته‌اند که تعدادی از آن‌ها به قرار ذیل هستند:

- PostSharp
- LOOM.NET
- Wicca
- و ...

در بین این‌ها، PostSharp معروفترین فریم ورک AOP بوده و در ادامه از آن استفاده خواهیم کرد.

پیشنیاز ادامه بحث

ابتدا یک پروژه کنسول جدید را آغاز کرده و سپس در خط فرمان پاور شل نوگت در VS.NET دستور ذیل را اجرا کنید:

```
PM> Install-Package PostSharp
```

به این ترتیب ارجاعی به PostSharp به پروژه جاری اضافه خواهد شد. البته حجم آن نسبتا بالا است؛ نزدیک به 20 مگ به همراه ابزارهای تزریق کد همراه با آن. مجوز استفاده از آن نیز تجاری و مدت دار است.

مراحل ایجاد یک Aspect برای پروسه IL Code Weaving

ابتدا یک کلاس پایه مشتق شده از کلاسی ویژه موجود در یکی از فریم ورک‌های AOP باید تعریف شود. مرحله بعد، کار اتصال این Aspect می‌باشد که توسط پردازشگر ثانویه IL Code Weaving انجام می‌شود.
در ادامه قصد داریم همان مثال LoggingInterceptor قسمت دوم این سری را با استفاده از IL Code Weaving پیاده سازی کنیم.

```
using System;

namespace AOP03
{
    public class MyType
    {
        public void DoSomething(string data, int i)
        {
            Console.WriteLine("DoSomething({0}, {1});", data, i);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            new MyType().DoSomething("Test", 1);
        }
    }
}
```

کدهای برنامه همانند قبل است. اما اینبار بجای استفاده از Interceptors، با ارث بری از کلاس OnMethodBoundaryAspect کتابخانه PostSharp شروع خواهیم کرد:

```
using System;
using PostSharp.Aspects;

namespace AOP03
{
    [Serializable]
    public class LoggingAspect : OnMethodBoundaryAspect
    {
        public override void OnEntry(MethodExecutionArgs args)
        {
            Console.WriteLine("On Entry");
        }

        public override void OnExit(MethodExecutionArgs args)
        {
            Console.WriteLine("On Exit");
        }

        public override void OnSuccess(MethodExecutionArgs args)
        {
            Console.WriteLine("On Success");
        }

        public override void OnException(MethodExecutionArgs args)
        {
            Console.WriteLine("On Exception");
        }
    }
}
```

نیاز است این کلاس توسط ویژگی Serializable مزین شود تا توسط PostSharp قابل استفاده گردد. همانطور که ملاحظه می‌کنید، مراحل مختلف اجرای یک Aspcet در اینجا با override متدهای کلاس پایه OnMethodBoundaryAspect پیاده سازی شده‌اند. این مراحل را پیشتر در زمان استفاده از Interceptors توسط try/finally/catch بررسی کرده بودیم.

اکنون اگر برنامه را اجرا کنیم، اتفاق خاصی رخ نداده و همان خروجی معمول متد DoSomething در کنسول نمایش داده خواهد شد. بنابراین در مرحله بعد نیاز است تا این Aspect را به کدهای برنامه متصل کنیم.

کلاس OnMethodBoundaryAspect در کتابخانه PostSharp، از کلاس MulticastAttribute مشتق می‌شود. بنابراین LoggingAspect ایی را که ایجاد کرده‌ایم نیز می‌توان به صورت یک ویژگی به متدهای مورد نظر خود افزود:

```
public class MyType
    {
        [LoggingAspect]
        public void DoSomething(string data, int i)
        {
            Console.WriteLine("DoSomething({0}, {1});", data, i);
        }
    }
```

اکنون اگر برنامه را اجرا کنیم، با خروجی زیر مواجه خواهیم شد:

```
On Entry
DoSomething(Test, 1);
On Success
On Exit
```

برای اینکه بتوان عملیات رخ داده را بهتر توضیح داد می‌تواند از یک دی‌کامپایلر مانند برنامه معروف Reflector استفاده کرد:

```
public void DoSomething(string data, int i)
{
    <>z__Aspects.a0.OnEntry(null);
    try
    {
        Console.WriteLine("DoSomething({0}, {1});", data, i);
        <>z__Aspects.a0.OnSuccess(null);
    }
    catch (Exception)
    {
        <>z__Aspects.a0.OnException(null);
```

```
        throw;
    }
    finally
    {
        <>z__Aspects.a0.OnExit(null);
    }
}
```

این کدی است که به صورت پویا توسط PostSharp به اسمبلی نهایی فایل اجرایی برنامه تزریق شده است.

خوب! این یک روش اتصال Aspects به برنامه است. اما اگر همانند Interceptors بخواهیم Aspect تعریف شده را سراسری اعمال کنیم چکار باید کرد (بدون نیاز به قرار دادن ویژگی بر روی تک تک متدها)؟
برای اینکار ابتدا نیاز است میدان عملکرد Aspect تعریف شده را توسط ویژگی MulticastAttributeUsage محدود کنیم تا برای مثال به خواص اعمال نشوند:

```
[Serializable]
[MulticastAttributeUsage(MulticastTargets.Method, TargetMemberAttributes =
MulticastAttributes.Instance)]
public class LoggingAspect : OnMethodBoundaryAspect
```

سپس فایل AssemblyInfo.cs استاندارد پروژه را گشوده و سطر زیر را به آن اضافه کنید:

```
[assembly: LoggingAspect(AttributeTargetTypes = "AOP03.*")]
```

توسط AttributeTargetTypes می‌توان اعمال این Aspect را به یک فضای نام خاص نیز محدود کرد.

مزیت روش IL Code Weaving نسبت به Interceptors، کارآیی و سرعت بالاتر است. از این جهت که کدهایی که قرار است اجرا شوند، پیشتر در اسمبلی برنامه قرار گرفته‌اند و نیازی نیست تا در زمان اجرا، کدی به برنامه به صورت پویا تزریق گردد.

نظرات خوانندگان

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۳/۱۰/۰۹ ۱۸:۲۷

با سلام.
چگونه می‌توان به IUnitOfWork درون Aspectهای تعریف شده بوسیله PostSharp ،دسترسی یافت؟
(تمامی Aspectها درون Dll دیگر قرار دارند).

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۰۹ ۱۹:۵

« Importing Dependencies from the Target Object »