عنوان: **معرفی Aspect oriented programming**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۹ ۱۰:۳۵
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns, AOP, C#, Architecture

AOP یا Aspect oriented programming چیست؟

AOP یکی از فناوری‌های مرتبط با توسعه نرم افزار محسوب می‌شود که توسط آن می‌توان اعمال مشترک و متداول موجود در برنامه را در یک یا چند ماژول مختلف قرار داد (که به آن‌ها Aspects نیز گفته می‌شود) و سپس آن‌ها را به مکان‌های مختلفی در برنامه متصل ساخت. عموما Aspects، قابلیت‌هایی را که قسمت عمده‌ای از برنامه را تحت پوشش قرار می‌دهند، کپسوله می‌کنند. اصطلاحا به این نوع قابلیت‌های مشترک، تکراری و پراکنده مورد نیاز در قسمت‌های مختلف برنامه، Cross cutting concerns نیز گفته می‌شود؛ مانند اعمال ثبت وقایع سیستم، امنیت، مدیریت تراکنش‌ها و امثال آن. با قرار دادن این نیازها در Aspects مجزا، می‌توان برنامه‌ای را تشکیل داد که از کدهای تکراری عاری است.

مثالی از کدهای تکراری پراکنده در برنامه

به برنامه ذیل و قسمت‌های مختلف ثبت وقایع آن دقت کنید:

```
using System;

namespace AOP00
{
    class Program
    {
        static void Main(string[] args)
        {
            Log.Debug("Program has started.");
            //.....
            try
            {

            }
            catch (Exception ex)
            {
                Log.Error(ex);
                throw;
            }
            finally
            {
                //.....
                Log.Debug("Program has ended.");
            }
        }
    }
}
```

همانطور که ملاحظه می‌کنید، حجم بالایی از کدهای تکراری ثبت وقایع، تنها در قسمت کوچکی از برنامه تدارک دیده شده‌اند. این مساله نقض اصل DRY یا Don't repeat yourself است. کاری که برای رفع این مشکل قرار است انجام دهیم، استفاده از AOP و کپسوله سازی اعمال تکراری و سپس اتصال آن به قسمت‌های مختلف برنامه است.

معرفی Aspects و مزایای استفاده از آن‌ها

همانطور که عنوان شد اولین گام در AOP، کپسوله سازی کدهای تکراری است که اصطلاحا یک Aspect را تشکیل می‌دهند. بنابراین هر Aspect صرفا یک محصور کننده قابلیتی خاص و تکراری در برنامه است. این Aspect باید اصل SRP یا Single responsibility principle (تک مسئولیتی) را رعایت کند. برای اتصال یک Aspect به قطعه‌های مختلف کدهای برنامه از الگوی طراحی تزئین کننده یا Decorator pattern استفاده می‌شود. به این ترتیب که این Aspect خاص قرار است قسمتی از کدهای برنامه را تزئین کند. همچنین در این حالت، open closed principle نیز بهتر رعایت خواهد گردید. از این جهت که کدهای تکراری برنامه، به Aspects منتقل شده‌اند و دیگر نیازی نیست برای تغییر آن‌ها، کدهای قسمت‌های مختلف را تغییر داد (کدهای برنامه باز خواهند بود برای

توسعه و بسته برای تغییر). بنابراین با استفاده از Aspects، به یک طراحی شیءگرای بهتر نیز دست خواهیم یافت.

مراحل اجرای یک Aspect

هر Aspect برای تزئین یا اتصال به قسمت‌های مختلف برنامه، یک طول عمر کاری مشخص را طی می‌کند:

الف) مرحله OnStart

```
public User GetUserById(int id)
        {
            if (Cache.ExistsFor(id))
            {
                return Cache[id];
            }
            else
            {
                var user = LoadFromDb(id);
                Cache.AddFor("User", id, user);
                return user;
            }
        }
```

مرحله اول اجرای یک Aspect، در آغاز کار قطعه‌ای است که قرار است آن‌را مزین کند. بنابراین بلافاصله قبل از اجرای کدی، برای مثال در یک متد، قادر خواهیم بود تا قطعه کد موجود در Aspect ایی را فراخوانی و اجرا کنیم.
برای مثال در متد GetUserById، پیش از اینکه کار به مراجعه به بانک اطلاعاتی برسد، ابتدا وضعیت کش سیستم بررسی می‌شود. بنابراین در این مثال می‌توان قسمت بررسی کش را به یک Aspect مجزا منتقل ساخته و در صورتیکه اطلاعاتی موجود بود، بازگشت داده شود؛ در غیر اینصورت مجوز اجرای ادامه کدها صادر گردد.

ب) مرحله OnSuccess

مرحله OnSuccess زمانی اجرا می‌شود که اجرای یک متد بدون بروز استثنایی خاتمه یافته است.

ج) مرحله OnExit

مرحله OnExit همانند مرحله OnSuccess است؛ با این تفاوت که مرحله OnSuccess در صورت بروز استثنایی در کدها اجرا نخواهد شد اما مرحله OnExit همواره در پایان کار یک متد فراخوانی می‌گردد.

د) مرحله OnError

مرحله OnError در طول عمر یک Aspect، در زمان بروز استثنایی رخ می‌دهد. برای مثال به این ترتیب می‌توان قسمت ثبت وقایع بروز استثناهای سیستم را کلا به یک Aspect مشخص انتقال داده و حجم کدهای تکراری را به این ترتیب به شدت کاهش داد.

انواع مختلف AOP

تا اینجا شاید این سؤال برای شما پیش آمده باشد که خوب! جالب است! اما چطور می‌خواهید در مراحلی که یاد شد، دخالت کرده و قطعه کدی را تزریق کنید؟
در AOP دو روش متداول کلی برای انجام اعمال تزریق کد وجود دارند:

1) استفاده از Interceptors

به کمک Interceptors، فرآیند فراخوانی متدها و خواص یک کلاس، تحت کنترل و نظارت قرار خواهند گرفت. برای انجام این امر، عموما از IOC Containers استفاده می‌شود (Inversion of control). احتمالا تا کنون از این کتابخانه‌ها تنها برای تزریق وابستگی‌های برنامه خود کمک گرفته‌اید و از سایر توانمندی‌های آن‌ها آنچنان استفاده‌ای نکرده‌اید. در این حالت، زمانیکه یک IOC Container کار وهله سازی کلاس خاصی را انجام می‌دهد، در همین حین می‌تواند مراحل یاد شده شروع، پایان و خطای متدها یا فراخوانی‌های خواص را نیز تحت نظر قرار داده و به این ترتیب مصرف کننده امکان تزریق کدهایی را در این مکان‌ها خواهد یافت.
مزیت مهم استفاده از Interceptors، عدم نیاز به کامپایل و یا تغییر ثانویه اسمبلی‌های موجود برای تغییری در کدهای آن‌ها است (برای تزریق نواحی تحت کنترل قرار دادن اعمال) و تمام کارها به صورت خودکار در زمان اجرای برنامه مدیریت می‌گردند.

2) بهره گیری از فناوری IL Code Weaving

در فناوی IL Code Weaving، ابتدا برنامه و ماژول‌های آن به نحو متداولی کامپایل و تبدیل به dll یا exe خواهند شد. سپس این dllها و فایل‌های اجرایی به پردازشگر ثانویه یک فریم ورک AOP برای تغییر و تزریق کدها سپرده خواهند شد. برای مثال در این حالت، کدهای سطح پایین IL مرتبط با مراحل مختلف اجرای یک Aspect، تولید و به اسمبلی‌های نهایی برنامه تزریق می‌شوند. اکنون به dll یا فایل اجرایی جدیدی خواهیم رسید که علاوه بر کدهای اصلی برنامه، حاوی کدهای تزریق شده تمام Aspects تعریف شده نیز هستند.