عنوان: **الگوی نماینده (پروکسی) Proxy Pattern**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۵/۰۱/۲۴ ۱:۳۰
آدرس: www.dotnettips.info
گروه‌ها: Design patterns

همه کاربران کامپیوتر در ایران به خوبی با کلمه پروکسی آشنا هستند. پروکسی به معنی نماینده یا واسط است و پروکسی واسطی است بین ما و شیء اصلی. پروکسی در شبکه به این معنی است که سیستم شما به یک سیستم واسط متصل شده است که از طریق پروکسی محدودیت‌های دسترسی برای آن تعریف شود. در اینجا هم پروکسی در واقع به همین منظور استفاده می‌شود. در تعدادی از کامنت‌های سایت خوانده بودم دوستان در مورد اصول SOLID و Refactoring بحث می‌کردند که آیا انجام عمل اعتبارسنجی در خود اصل متد کار درستی است یا خیر. در واقع خودم هم نمی‌دانم که این حرکت چقدر به این اصول پایبند است یا خیر، ولی می‌دانم که الگوی پروکسی کل سؤالات بالا را حذف می‌کند و با این الگو دیگر این سؤال اهمیتی ندارد. به عنوان مثال فرض کنید که شما یک برنامه ساده کار با فایل را دارید. ولی اگر بخواهید اعتبارسنجی‌هایی را برای آن تعریف کنید، بهتر است اینکار را به یک پروکسی بسپارید تا شیء گرایی بهتری را داشته باشید.

برای شروع ابتدا یک اینترفیس تعریف می‌کنیم:

```
public interface IFilesService
    {
        DirectoryInfo GetDirectoryInfo(string directoryPath);
        void DeleteFile(string fileName);
        void WritePersonInFile(string fileName,string name, string lastName, byte age);
    }
```

این اینترفیس شامل سه متد نام نویسی، حذف فایل و دریافت اطلاعات یک دایرکتوری است. بعد از آن دو عدد کلاس را از آن مشتق می‌کنیم:
کلاس اصلی:

```
class FilesServices:IFilesService
    {
        public DirectoryInfo GetDirectoryInfo(string directoryPath)
        {
            return new DirectoryInfo(directoryPath);
        }

        public void DeleteFile(string fileName)
        {
            File.Delete(fileName);
            Console.WriteLine("the file has been deleted");
        }

        public void WritePersonInFile(string fileName, string name, string lastName, byte age)
        {
            var text = $"my name is {name} {lastName} with {age} years old from dotnettips.info";
            File.WriteAllText(fileName,text);
        }
    }
```

که تنها وظیفه اجرای فرامین را دارد و دیگری کلاس پروکسی است که وظیف تامین اعتبارسنجی و آماده سازی پیش شرطها را دارد:

```
class FilesServicesProxy:IFilesService
    {
        private readonly IFilesService _filesService;

        public FilesServicesProxy()
        {
            _filesService=new FilesServices();
        }

        public DirectoryInfo GetDirectoryInfo(string directoryPath)
        {
```

```
            var existing = Directory.Exists(directoryPath);
            if (!existing)
                Directory.CreateDirectory(directoryPath);
            return _filesService.GetDirectoryInfo(directoryPath);
        }

        public void DeleteFile(string fileName)
        {
            if(!File.Exists(fileName))
                Console.WriteLine("Please enter a valid path");
            else
                _filesService.DeleteFile(fileName);
        }

        public void WritePersonInFile(string fileName, string name, string lastName, byte age)
        {
            if (!Directory.Exists(fileName.Remove(fileName.LastIndexOf("\\"))))
            {
                Console.WriteLine("File Path is not valid");
                return;
            }
            if (name.Trim().Length == 0)
            {
                Console.WriteLine("first name must enter");
                return;
            }

            if (lastName.Trim().Length == 0)
            {
                Console.WriteLine("last name must enter");
                return;
            }

            if (age<18)
            {
                Console.WriteLine("your age is illegal");
                return;
            }


            if (name.Trim().Length < 3)
            {
                Console.WriteLine("first name must be more than 2 letters");
                return;
            }

            if (lastName.Trim().Length <5)
            {
                Console.WriteLine("last name must be more than 4 letters");
                return;
            }

            _filesService.WritePersonInFile(fileName,name,lastName,age);
            Console.WriteLine("the file has been written");
        }
    }
```

کلاس پروکسی، همان کلاسی است که شما باید صدا بزنید. وظیفه کلاس پروکسی هم این است که در زمان معین و صحیح، کلاس اصلی شما را بعد از اعتبارسنجی‌ها و انجام پیش شرط‌ها صدا بزند. همانطور که می‌بیند، ما در سازنده کلاس اصلی را در حافظه قرار می‌دهیم. سپس در هر متد، اعتبارسنجی‌های لازم را انجام می‌دهیم. مثلا در متد GetDirectoryInfo باید اطلاعات دایرکتوری بازگشت داده شود؛ ولی اصل عمل در واقع در کلاس اصلی است و اینجا فقط شرط گذاشته‌ایم اگر مسیر داده شده معتبر نبود، همان مسیر را ایجاد کن و سپس متد اصلی را صدا بزن. یا اگر فایل موجود است جهت حذف آن اقدام کن و ...
در نهایت در بدنه اصلی با تست چندین حالت مختلف، همه متدها را داریم:

```
static void Main(string[] args)
        {
            IFilesService filesService=new FilesServicesProxy();
            filesService.WritePersonInFile("c:\\myfakepath\\a.txt","ali","yeganeh",26);

            var directory = filesService.GetDirectoryInfo("d:\\myrightpath\\");
            var fileName = Path.Combine(directory.FullName, "dotnettips.txt");

            filesService.WritePersonInFile(fileName, "al", "yeganeh", 26);
            filesService.WritePersonInFile(fileName, "ali", "yeganeh", 12);
```

```
            filesService.WritePersonInFile(fileName, "ali", "yeganeh", 26);

            filesService.DeleteFile("c:\\myfakefile.txt");
            filesService.DeleteFile(fileName);
        }
```

و نتیجه خروجی:

```
File Path is not valid
first name must be more than 2 letters
your age is illegal
the file has been written
Please enter a valid path
the file has been deleted
```

نظرات خوانندگان

نویسنده: عثمان رحیمی
تاریخ: ۱۳۹۵/۰۱/۲۴ ۸:۳۸

تشکر از شما آقای یگانه مقدم بابت مطلب مفید و متن روان مقاله .
یک سوال ، نظر شما در رابطه با اینکه برای تمام سرویس‌ها از این الگو استفاده کنیم چیست (یعنی دیگر داخل خود متدهای سرویس اعتبار سنجی ... رو انجام ندیم) ؟ و یا اینکه چه وقت هایی و برای چه سرویس هایی بهتر است از این الگو استفاده کنیم ؟

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۵/۰۱/۲۴ ۸:۵۱

ممنون از شما
به نظرم برای سرویس هایی که اعتبارسنجی‌های زیاد و طولانی دارند بسیار مناسب است و میزان نگهداری کد را بالاتر میبرد. در مورد سرویس‌ها و موارد مشابه که عموما از تزریق وابستگی استفاده میکنیم کمی مشکل ایجاد میکند که باید این را هم در نظر گرفت که اگر قرار باشد پروکسی‌های زیادی داشته باشیم مثلا کل تزریق برای پروکسی صورت خواهد گرفت که کلاس اصلی که اصلی‌ترین هدف تزریق است در این بین کمی ایجاد مشکل میکند. یعنی باید یک الگوی دیگر جهت استفاده از تزریق وابستگی‌ها پیدا کرد یا با الگوهایی مثل service locator پیش برویم.

نویسنده: محسن خان
تاریخ: ۱۳۹۵/۰۱/۲۴ ۱۰:۵۷

یک سری از الگوها چند دهه قبل معرفی شدند، ولی این روزها شاید به این شکل خام استفاده نشوند. برای مثال اصل مباحث AOP بر پایه پروکسی‌ها هست و یا نمونه‌ی بهتری برای اعتبارسنجی شاید استفاده از Code contracts باشد.

نویسنده: احمد نواصری
تاریخ: ۱۳۹۵/۰۱/۲۵ ۹:۴۵

نام دیگر این الگو، الگوی غشا یا محصور کننده هست. زمانی که شما یک شی داشته باشید و نمیخواهید سرویس گیرندگان به صورت مستقیم به آن دسترسی داشته باشند، شی اصلی را با کلاس Proxy محصور میکنید و یا به اصطلاح دور اون رو یک غشا درست میکنید و از طریق این غشا به سرویس گیرندگان سرویس میدهید.