عنوان: **مراحل Refactoring یک قطعه کد برای اعمال تزریق وابستگی‌ها**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۳۰ ۱۲:۱۰
آدرس: www.dotnettips.info
گروه‌ها: Design patterns, Dependency Injection, IoC

برای رسیدن به الگوی معکوس سازی وابستگی‌ها عموما مراحل زیر طی می‌شوند:

الف) در متدهای لایه جاری خود واژه‌های کلیدی new و همچنین کلیه فراخوانی‌های استاتیک را بیابید.
ب) وهله سازی این‌ها را به یک سطح بالاتر (نقطه آغازین برنامه) منتقل کنید. اینکار باید بر اساس اتکای به Abstraction و برای مثال استفاده از اینترفیس‌ها صورت گیرد.
ج) اینکار را آنقدر تکرار کنید تا دیگر در کدهای لایه جاری خود واژه کلیدی new یا فراخوانی متدهای استاتیک مشاهده نشود.
د) در آخر وهله سازی object graphهای مورد نیاز را به یک IoC Container محول کنید.

یک مثال: ابتدا بررسی یک قطعه کد متداول

```
using System.Net;
using System.Text;
using System.Text.RegularExpressions;
using System.Web.Mvc;

namespace DI06.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            string result = string.Empty;
            using (var client = new WebClient { Encoding = Encoding.UTF8 })
            {
                result = client.DownloadString("http://www.dotnettips.info/");
            }
            var match = new Regex(@"(?s)<title>(.+?)</title>", RegexOptions.IgnoreCase).Match(result);
            var title = match.Groups[1].Value.Trim();

            ViewBag.PageTitle = title;
            return View();
        }
    }
}
```

فرض کنید یک برنامه ASP.NET MVC را به نحو فوق تهیه کرده‌ایم. در کدهای کنترلر آن قصد داریم محتویات Html یک صفحه خاص را دریافت و سپس عنوان آن‌را استخراج کرده و نمایش دهیم.
مشکلات کد فوق:
الف) قرار گرفتن منطق تجاری پیاده سازی کدها مستقیما داخل کدهای یک اکشن متد؛ این مساله در دراز مدت به تکرار شدید کدها منجر خواهد شد که نهایتا قابلیت نگهداری آن‌را کاهش می‌دهند.
ب) در این کد حداقل دو بار واژه کلیدی new ذکر شده است. مورد اول یا new WebClient، از همه مهم‌تر است؛ از این جهت که نوشتن آزمون واحد را برای این کنترلر بسیار مشکل می‌کند. آزمون‌های واحد باید سریع و بدون نیاز به منابع خارجی، قابل اجرا باشند. تعویض آن هم مطابق کدهای تدارک دیده شده کار ساده‌ای نیست.

بهبود کیفیت قطعه کد متداول فوق با استفاده از الگوی معکوس سازی وابستگی‌ها

در اصل معکوس سازی وابستگی‌ها عنوان کردیم لایه بالایی سیستم نباید مستقیما به لایه‌های زیرین در حال استفاده از آن، وابسته باشد. این وابستگی باید معکوس شده و همچنین بر اساس Abstraction یا برای مثال استفاده از اینترفیس‌ها صورت گیرد.
به همین منظور یک پروژه دیگر را از نوع Class library، مثلا به نام DI06.Services به Solution جاری اضافه می‌کنیم.

```
namespace DI06.Services
{
```

```
    public interface IWebClientServices
    {
        string FetchUrl(string url);
        string GetWebPageTitle(string url);
    }
}

using System.Net;
using System.Text;
using System.Text.RegularExpressions;

namespace DI06.Services
{
    public class WebClientServices : IWebClientServices
    {
        public string FetchUrl(string url)
        {
            using (var client = new WebClient { Encoding = Encoding.UTF8 })
            {
                return client.DownloadString(url);
            }
        }

        public string GetWebPageTitle(string url)
        {
            var html = FetchUrl(url);
            var match = new Regex(@"(?s)<title>(.+?)</title>", RegexOptions.IgnoreCase).Match(html);
            return match.Groups[1].Value.Trim();
        }
    }
}
```

در این لایه، سرویس WebClient ایی را تدارک دیده‌ایم. این سرویس می‌تواند محتوای Html یک آدرس را برگرداند و یا عنوان آن آدرس خاص را استخراج نماید.

هنوز کار معکوس سازی وابستگی‌ها رخ نداده است. صرفا اندکی تمیزکاری و انتقال پیاده سازی منطق تجاری به یک سری کلاس‌هایی با قابلیت استفاده مجدد صورت گرفته است. به این ترتیب اگر باگی در این کدها وجود داشته باشد و همچنین از آن در چندین نقطه برنامه استفاده شده باشد، اصلاح این کلاس مرکزی، به یکباره تمامی قسمت‌های مختلف برنامه را تحت تاثیر مثبت قرار داده و از تکرار کدها و فراموشی احتمالی بهبود قسمت‌های مشابه جلوگیری می‌کند.

کار معکوس سازی وابستگی‌ها در یک لایه بالاتر صورت خواهد گرفت:

```
using System.Web.Mvc;
using DI06.Services;

namespace DI06.Controllers
{
    public class HomeController : Controller
    {
        readonly IWebClientServices _webClientServices;
        public HomeController(IWebClientServices webClientServices)
        {
            _webClientServices = webClientServices;
        }

        public ActionResult Index()
        {
            ViewBag.PageTitle = _webClientServices.GetWebPageTitle("http://www.dotnettips.info/");
            return View();
        }
    }
}
```

اینبار کنترلر Home را توسط تزریق وابستگی‌ها در سازنده کنترلر، بازنویسی کرده‌ایم. کد نهایی بسیار تمیزتر از حالت قبل است. دیگر پیاده سازی متد GetWebPageTitle مستقیما داخل یک اکشن متد قرار نگرفته است. همچنین این کنترلر اصلا نمی‌داند که قرار است از کدام پیاده سازی اینترفیس IWebClientServices استفاده کند. اگر در تنظیمات اولیه IoC Container مورد استفاده، کلاس WebClientServices ذکر شده باشد، از آن استفاده خواهد کرد؛ یا اگر حتی کلاس **Fake** WebClientServices نیز معرفی گردد (جهت انجام آزمون غیر وابسته به new WebClient)، باز هم بدون کوچکترین تغییری در کدهای آن قابل استفاده خواهد بود.

در مورد نحوه تنظیمات اولیه یک IoC Container و یا پیشنیازهای ASP.NET MVC جهت آماده شدن برای تزریق خودکار وابستگی‌ها در سازنده کنترلرها، پیشتر مطالبی را در این سری مطالعه کرده‌اید؛ در اینجا نیز اصول مورد استفاده یکی است و تفاوتی نمی‌کند.

نظرات خوانندگان

نویسنده: اکبر بابامرادی
تاریخ: ۱۳۹۲/۰۴/۱۸ ۱۶:۲۶

لطفا مثالی جهت درک مفاهیم معکوس سازی، در کنترلرهایی که با دیتابیس ارتباط دارند و یک مدل به View ارسال می‌کنند، بزنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۱۸ ۱۶:۲۹

- اصول آن تفاوتی نمی‌کند. مراحل تشکیل لایه سرویس، برپایی IoC و همچنین تزریق وابستگی‌ها در سازنده کنترلرها یکی است.
+ مراجعه کنید به قسمت 12 سری EF Code first . یک مثال در این مورد (کار با دیتابیس در MVC، وب فرم‌ها و برنامه‌های ویندوزی به همراه تزریق وابستگی‌ها) وجود دارد و همچنین قابل دریافت است.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۱/۳۰ ۲۳:۵۶

یک سری نکته‌ی دیگر را نیز جهت refactoring کدها به تزریق وابستگی‌ها، در اینجا می‌توانید مطالعه کنید.