عنوان: **استفاده از AOP Interceptors برای حذف کدهای تکراری کش کردن اطلاعات در لایه سرویس برنامه**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۲۰ ۱۹:۳۶
آدرس: www.dotnettips.info
گروه‌ها: C#, Design patterns, AOP, Architecture

اکثر برنامه‌های ما دارای قابلیت‌هایی هستند که با موضوعاتی مانند امنیت، کش کردن اطلاعات، مدیریت استثناها، ثبت وقایع و غیره گره خورده‌اند. به هر یک از این موضوعات یک Aspect یا cross-cutting concern نیز گفته می‌شود.
در این قسمت قصد داریم اطلاعات بازگشتی از لایه سرویس برنامه را کش کنیم؛ اما نمی‌خواهیم مدام کدهای مرتبط با کش کردن اطلاعات را در مکان‌های مختلف لایه سرویس پراکنده کنیم. می‌خواهیم یک ویژگی یا Attribute سفارشی را تهیه کرده (مثلا به نام CacheMethod) و به متد یا متدهایی خاص اعمال کنیم. سپس برنامه، در زمان اجرا، بر اساس این ویژگی‌ها، خروجی‌های متدهای تزئین شده با ویژگی CacheMethod را کش کند.
در اینجا نیز از ترکیب StructureMap و DynamicProxy پروژه Castle، برای رسیدن به این مقصود استفاده خواهیم کرد. به کمک StructureMap می‌توان در زمان وهله سازی کلاس‌ها، آن‌ها را به کمک متدی به نام EnrichWith توسط یک محصور کننده دلخواه، مزین یا غنی سازی کرد. این مزین کننده را جهت دخالت در فراخوانی‌های متدها، یک DynamicProxy درنظر می‌گیریم. با پیاده سازی اینترفیس IInterceptor کتابخانه DynamicProxy مورد استفاده و تحت کنترل قرار دادن نحوه و زمان فراخوانی متدهای لایه سرویس، یکی از کارهایی را که می‌توان انجام داد، کش کردن نتایج است که در ادامه به جزئیات آن خواهیم پرداخت.

پیشنیازها

ابتدا یک برنامه جدید کنسول را آغاز کنید. تنظیمات آن‌را از حالت Client profile به Full تغییر دهید.
سپس همانند قسمت‌های قبل، ارجاعات لازم را به StructureMap و Castle.Core نیز اضافه نمائید:

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

همچنین ارجاعی را به اسمبلی استاندارد System.Web.dll نیز اضافه نمائید.
از این جهت که از HttpRuntime.Cache قصد داریم استفاده کنیم. HttpRuntime.Cache در برنامه‌های کنسول نیز کار می‌کند. در این حالت از حافظه سیستم استفاده خواهد کرد و در پروژه‌های وب از کش IIS بهره می‌برد.

ویژگی CacheMethod مورد استفاده

```
using System;

namespace AOP02.Core
{
    [AttributeUsage(AttributeTargets.Method)]
    public class CacheMethodAttribute : Attribute
    {
        public CacheMethodAttribute()
        {
            // مقدار پیش فرض
            SecondsToCache = 10;
        }

        public double SecondsToCache { get; set; }
    }
}
```

همانطور که عنوان شد، قصد داریم متدهای مورد نظر را توسط یک ویژگی سفارشی، مزین سازیم تا تنها این موارد توسط AOP Interceptor مورد استفاده پردازش شوند.
در ویژگی CacheMethod، خاصیت SecondsToCache بیانگر مدت زمان کش شدن نتیجه متد خواهد بود.

ساختار لایه سرویس برنامه

```
using System;
using System.Threading;
using AOP02.Core;

namespace AOP02.Services
{
    public interface IMyService
    {
        string GetLongRunningResult(string input);
    }

    public class MyService : IMyService
    {
        [CacheMethod(SecondsToCache = 60)]
        public string GetLongRunningResult(string input)
        {
            Thread.Sleep(5000); // simulate a long running process
            return string.Format("Result of '{0}' returned at {1}", input, DateTime.Now);
        }
    }
}
```

اینترفیس IMyService و پیاده سازی نمونه آن‌را در اینجا مشاهده می‌کنید. از این لایه در برنامه استفاده شده و قصد داریم نتیجه بازگشت داده شده توسط متدی زمانبر را در اینجا توسط AOP Interceptors کش کنیم.

تدارک یک CacheInterceptor

```
using System;
using System.Web;
using Castle.DynamicProxy;

namespace AOP02.Core
{
    public class CacheInterceptor : IInterceptor
    {
        private static object lockObject = new object();

        public void Intercept(IInvocation invocation)
        {
            cacheMethod(invocation);
        }

        private static void cacheMethod(IInvocation invocation)
        {
            var cacheMethodAttribute = getCacheMethodAttribute(invocation);
            if (cacheMethodAttribute == null)
            {
                // متد جاری توسط ویژگی کش شدن مزین نشده است
                // بنابراین آن‌را اجرا کرده و کار را خاتمه می‌دهیم
                invocation.Proceed();
                return;
            }

            // دراینجا مدت زمان کش شدن متد از ویژگی کش دریافت می‌شود
            var cacheDuration = ((CacheMethodAttribute)cacheMethodAttribute).SecondsToCache;

            // برای ذخیره سازی اطلاعات در کش نیاز است یک کلید منحصربفرد را
            //  بر اساس نام متد و پارامترهای ارسالی به آن تهیه کنیم
            var cacheKey = getCacheKey(invocation);

            var cache = HttpRuntime.Cache;
            var cachedResult = cache.Get(cacheKey);


            if (cachedResult != null)
            {
                // اگر نتیجه بر اساس کلید تشکیل شده در کش موجود بود
                // همان را بازگشت می‌دهیم
                invocation.ReturnValue = cachedResult;
            }
            else
            {
```

```
                lock (lockObject)
                {
                    // در غیر اینصورت ابتدا متد را اجرا کرده
                    invocation.Proceed();
                    if (invocation.ReturnValue == null)
                        return;

                    // سپس نتیجه آن‌را کش می‌کنیم
                    cache.Insert(key: cacheKey,
                                 value: invocation.ReturnValue,
                                 dependencies: null,
                                 absoluteExpiration: DateTime.Now.AddSeconds(cacheDuration),
                                 slidingExpiration: TimeSpan.Zero);
                }
            }
        }

        private static Attribute getCacheMethodAttribute(IInvocation invocation)
        {
            var methodInfo = invocation.MethodInvocationTarget;
            if (methodInfo == null)
            {
                methodInfo = invocation.Method;
            }
            return Attribute.GetCustomAttribute(methodInfo, typeof(CacheMethodAttribute), true);
        }

        private static string getCacheKey(IInvocation invocation)
        {
            var cacheKey = invocation.Method.Name;

            foreach (var argument in invocation.Arguments)
            {
                cacheKey += ":" + argument;
            }

            // todo: بهتر است هش این کلید طولانی بازگشت داده شود
            // کار کردن با هش سریعتر خواهد بود
            return cacheKey;
        }
    }
}
```

کدهای CacheInterceptor مورد استفاده را در بالا مشاهده می‌کنید.
توضیحات ریز قسمت‌های مختلف آن به صورت کامنت، جهت درک بهتر عملیات، ذکر شده‌اند.

اتصال Interceptor به سیستم

خوب! تا اینجای کار صرفا تعاریف اولیه تدارک دیده شده‌اند. در ادامه نیاز است تا DI و DynamicProxy را از وجود آن‌ها مطلع کنیم.

```
using System;
using AOP02.Core;
using AOP02.Services;
using Castle.DynamicProxy;
using StructureMap;

namespace AOP02
{
    class Program
    {
        static void Main(string[] args)
        {
            ObjectFactory.Initialize(x =>
            {
                var dynamicProxy = new ProxyGenerator();
                x.For<IMyService>()
                 .EnrichAllWith(myTypeInterface =>
                        dynamicProxy.CreateInterfaceProxyWithTarget(myTypeInterface, new
CacheInterceptor()))
                 .Use<MyService>();
            });

            var myService = ObjectFactory.GetInstance<IMyService>();
```

```
            Console.WriteLine(myService.GetLongRunningResult("Test"));
            Console.WriteLine(myService.GetLongRunningResult("Test"));
        }
    }
}
```

در قسمت تنظیمات اولیه DI مورد استفاده، هر زمان که شیءایی از نوع IMyService درخواست شود، کلاس MyService وهله سازی شده و سپس توسط CacheInterceptor محصور می‌گردد. اکنون ادامه برنامه با این شیء محصور شده کار می‌کند.
حال اگر برنامه را اجرا کنید یک چنین خروجی قابل مشاهده خواهد بود:

```
Result of 'Test' returned at 2013/04/09 07:19:43
Result of 'Test' returned at 2013/04/09 07:19:43
```

همانطور که ملاحظه می‌کنید هر دو فراخوانی یک زمان را بازگشت داده‌اند که بیانگر کش شدن اطلاعات اولی و خوانده شدن اطلاعات فراخوانی دوم از کش می‌باشد (با توجه به یکی بودن پارامترهای هر دو فراخوانی).

از این پیاده سازی می‌شود به عنوان کش سطح دوم ORMها نیز استفاده کرد (صرفنظر از نوع ORM در حال استفاده).

دریافت مثال کامل این قسمت

AOP02.zip

نظرات خوانندگان

نویسنده: MehRad
تاریخ: ۱۳۹۲/۰۶/۱۹ ۱۸:۲۹

سلام
فرق این روش از کش کردن با کش سطح دوم که در این قسمت معرفی نمودین در چیست ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۶/۱۹ ۱۸:۳۷

- خلاصه‌ای از قسمت اول این دوره
«هر Aspect صرفا یک محصور کننده قابلیتی خاص و تکراری در برنامه است. 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.»
بنابراین فرق مهمش با روش کار با Expressions این است که شما در اینجا به یک Attribute جدید رسیدید که منطق پیاده سازی آن جایی در لابلای کدهای شما قرار نگرفته. هر زمان که نیازی به آن نبود، فقط کافی است که قسمت EnrichAllWith تنظیمات IoC Container یاده شده را حذف کرد. این روش یک دید دیگر طراحی شیءگرا است.
- از دیدگاه صرفا کاربردی:
الف) روش AOP یاد شده با هر نوع ORM ایی سازگار است. اصلا مهم نیست که الزاما EF باشد یا NH.
ب) چون درگیر بسیاری از جزئیات ریز تفسیر Expressions نشده، سریعتر است.

نویسنده: محسن موسوی
تاریخ: ۱۳۹۳/۰۹/۱۱ ۱۶:۵۸

به سازنده CacheInterceptor پارامتر UnitOfWork را پاس دادم.

```
var container = new Container();

            container.Configure(x =>
            {
x.For(typeof(IUnitOfWork)).Use(typeof(UnitOfWork)).SetLifecycleTo<HttpContextLifecycle>();
                x.Scan(scan =>
                {
                    scan.WithDefaultConventions();
                    scan.Assembly("Test.Services");
                });
                x.Policies.SetAllProperties(y =>
                {
                    y.WithAnyTypeFromNamespace("Test.Services.Interfaces");
                });
 var dynamicProxy = new ProxyGenerator();
                x.For<ITestService>()
                    .DecorateAllWith(myTypeInterface =>
                        dynamicProxy.CreateInterfaceProxyWithTarget(myTypeInterface,
                            container.GetInstance<CacheInterceptor>()));
            });
```

ولی uow به صورت HttpContextLifecycle نمیشه.مشکل این تعریف چیه؟
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۹/۱۱ ۱۷:۳۳

طول عمر Container ایی که تعاریف اولیه را دارد باید به صورت singleton تعریف شود. اگر قرار باشد هر جایی مجزا وهله سازی شود، Container های متفاوتی خواهید داشت با اشیاء متفاوتی. یک مثال در این مورد: DIO6.zip