عنوان: **استفاده از الگوی Adapter در تزریق وابستگی‌ها**
نویسنده: محمد رضا منشادی
تاریخ: ۱۳۹۲/۰۸/۱۹ ۱۸:۰
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns, Dependency Injection, Ioc, Dependency Inversion

در بعضی از مواقع ممکن است که در هنگام استفاده از اصل تزریق وابستگی‌ها، با یک مشکل روبرو شویم و آن این است که اگر از کلاسی استفاده می‌کنیم که به سورس آن دسترسی نداریم، نمی‌توانیم برای آن یک Interface تهیه کنیم و اصل (Depend on abstractions, not on concretions) از بین می‌رود، حال چه باید کرد.
برای اینکه موضوع تزریق وابستگی‌ها (DI) به صورت کامل در قسمتهای دیگر سایت توضیح داده شده است، دوباره آن را برای شما بازگو نمی‌کنیم .
لطفا به کدهای ذیل توجه کنید:

کد بدون تزریق وابستگی ها

به سازنده کلاس ProductService و تهیه یک نمونه جدید از وابستگی مورد نیاز آن دقت نمائید:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web;

namespace ASPPatterns.Chap2.Service
{
    public class Product
    {
    }

    public class ProductRepository
    {
        public IList<Product> GetAllProductsIn(int categoryId)
        {
            IList<Product> products = new List<Product>();
            // Database operation to populate products …
            return products;
        }
    }

    public class ProductService
    {
        private ProductRepository _productRepository;
        public ProductService()
        {
            _productRepository = new ProductRepository();
        }

        public IList<Product> GetAllProductsIn(int categoryId)
        {
            IList<Product> products;
            string storageKey = string.Format("products_in_category_id_{0}", categoryId);
            products = (List<Product>)HttpContext.Current.Cache.Get(storageKey);
            if (products == null)
            {
                products = _productRepository.GetAllProductsIn(categoryId);
                HttpContext.Current.Cache.Insert(storageKey, products);
            }
            return products;
        }
    }
}
```

همان کد با تزریق وابستگی

```
using System;
using System.Collections.Generic;
```

```
namespace ASPPatterns.Chap2.Service
{
    public interface IProductRepository
    {
        IList<Product> GetAllProductsIn(int categoryId);
    }

    public class ProductRepository : IProductRepository
    {
        public IList<Product> GetAllProductsIn(int categoryId)
        {
            IList<Product> products = new List<Product>();
            // Database operation to populate products …
            return products;
        }
    }

    public class ProductService
    {
        private IProductRepository _productRepository;
        public ProductService(IProductRepository  productRepository)
        {
            _productRepository = productRepository;
        }

        public IList<Product> GetAllProductsIn(int categoryId)
        {
            //…
        }
    }
}
```

همانطور که ملاحظه می‌کنید به علت دسترسی به سورس، به راحتی برای استفاده از کلاس ProductRepository در کلاس ProductService، از تزریق وابستگی‌ها استفاده کرده‌ایم.

اما از این جهت که شما دسترسی به سورس Http context class را ندارید، نمی‌توانید به سادگی یک Interface را برای آن ایجاد کنید و سپس یک تزریق وابستگی را مانند کلاس ProductRepository برای آن تهیه نمائید.

خوشبختانه این مشکل قبلا حل شده است و الگویی که به ما جهت پیاده سازی آن کمک کند، وجود دارد و آن الگوی آداپتر (Adapter Pattern) می‌باشد.

این الگو عمدتا برای ایجاد یک Interface از یک کلاس به صورت یک Interface سازگار و قابل استفاده می‌باشد. بنابراین می‌توانیم این الگو را برای تبدیل HTTP Context caching API به یک API سازگار و قابل استفاده به کار ببریم.

در ادامه می‌توان Interface سازگار جدید را در داخل productservice که از اصل تزریق وابستگی‌ها (DI) استفاده می‌کند تزریق کنیم.

یک اینترفیس جدید را با نام ICacheStorage به صورت ذیل ایجاد می‌کنیم:

```
public interface ICacheStorage
{
    void Remove(string key);
    void Store(string key, object data);
    T Retrieve<T>(string key);
}
```

حالا که شما یک اینترفیس جدید دارید، می‌توانید کلاس produceservic را به شکل ذیل به روز رسانی کنید تا از این اینترفیس، به جای HTTP Context استفاده کند.

```
public class ProductService
{
    private IProductRepository _productRepository;
    private ICacheStorage _cacheStorage;
    public ProductService(IProductRepository  productRepository,
    ICacheStorage cacheStorage)
    {
        _productRepository = productRepository;
        _cacheStorage = cacheStorage;
```

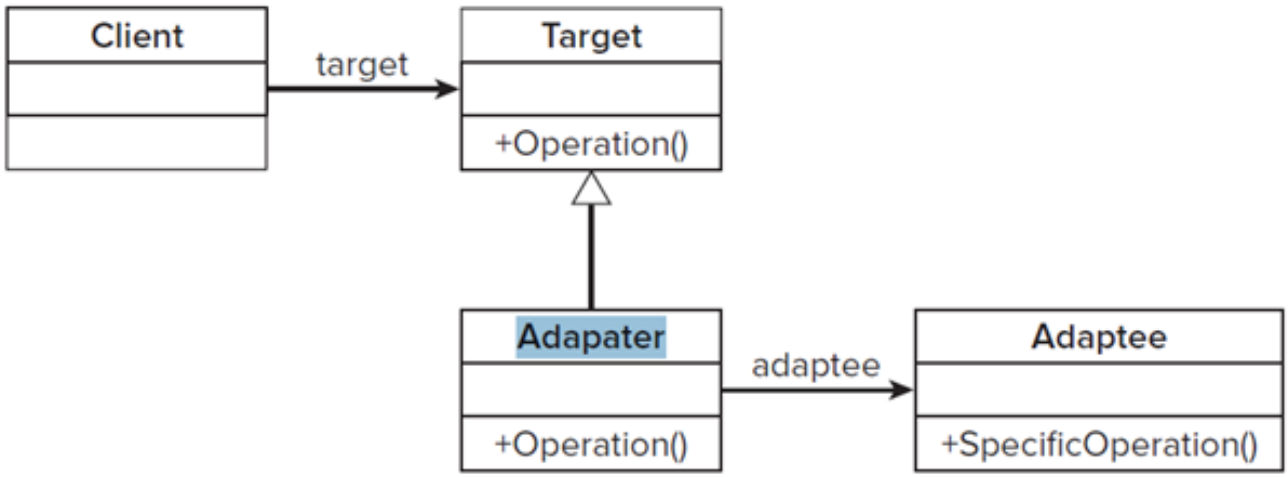
```
    }

    public IList<Product> GetAllProductsIn(int categoryId)
    {
        IList<Product> products;
        string storageKey = string.Format("products_in_category_id_{0}", categoryId);
        products = _cacheStorage.Retrieve<List<Product>>(storageKey);
        if (products == null)
        {
            products = _productRepository.GetAllProductsIn(categoryId);
            _cacheStorage.Store(storageKey, products);
        }
        return products;
    }
}
```

مسئله ای که در اینجا وجود دارد این است که HTTP Context Cache API صریحا نمی‌تواند Interface ایی که ما ایجاد کرده‌ایم را اجرا کند.

پس چگونه الگوی Adapter می‌تواند به ما کمک کند تا از این مشکل خارج شویم؟

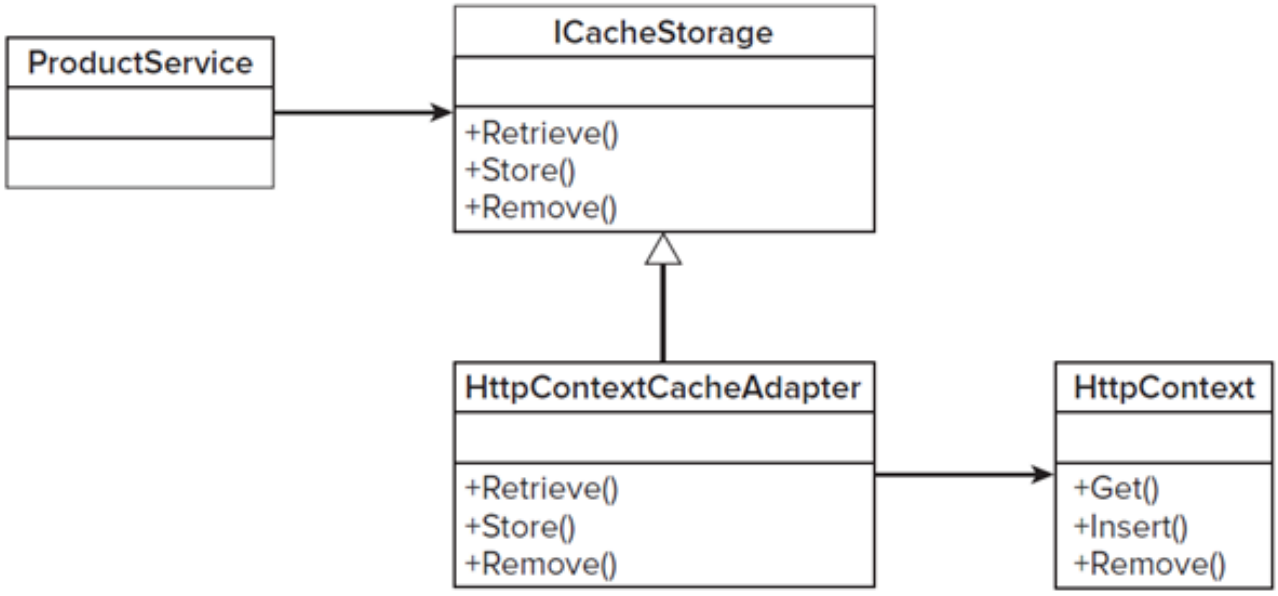
هدف این الگو به صورت ذیل در GOF مشخص شده است .«تبدیل Interface از یک کلاس به یک Interface مورد انتظار Client»

تصویر ذیل، مدل این الگو را به کمک UML نشان می‌دهد:



همانطور که در این تصویر ملاحظه می‌کنید، یک Client ارجاعی به یک Abstraction در تصویر (Target) دارد (ICacheStorage در کد نوشته شده). کلاس Adapter اجرای Target را بر عهده دارد و به سادگی متدهای Interface را نمایندگی می‌کند. در اینجا کلاس Adapter، یک نمونه از کلاس Adaptee را استفاده می‌کند و در هنگام اجرای قراردادهای Target، از این نمونه استفاده خواهد کرد.

اکنون کلاس‌های خود را در نمودار UML قرار می‌دهیم که به شکل ذیل آنها را ملاحظه می‌کنید.



در شکل ملاحظه می‌نمایید که یک کلاس جدید با نام HttpContextCacheAdapter مورد نیاز است. این کلاس یک کلاس روکش (محصور کننده یا Wrapper) برای متدهای HTTP Context cache است. برای اجرای الگوی Adapter کلاس HttpContextCacheAdapter را به شکل ذیل ایجاد می‌کنیم:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web;
namespace ASPPatterns.Chap2.Service
{
    public class HttpContextCacheAdapter : ICacheStorage
    {
        public void Remove(string key)
        {
            HttpContext.Current.Cache.Remove(key);
        }

        public void Store(string key, object data)
        {
            HttpContext.Current.Cache.Insert(key, data);
        }

        public T Retrieve<T>(string key)
        {
            T itemStored = (T)HttpContext.Current.Cache.Get(key);
            if (itemStored == null)
                itemStored = default(T);
            return itemStored;
        }
    }
}
```

حال به سادگی می‌توان یک caching solution دیگر را پیاده سازی کرد بدون اینکه در کلاس ProductService اثر یا تغییری ایجاد کند .

منبع : Professional Asp.Net Design Pattern