عنوان: **مروری بر کاربردهای Action و Func - قسمت دوم**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۸ ۱۸:۱۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, Refactoring

در قسمت قبل از Func و Actionها برای ساده سازی طراحی‌های مبتنی بر اینترفیس‌هایی با یک متد استفاده کردیم. این مورد خصوصا در حالت‌هایی که قصد داریم به کاربر اجازه‌ی فرمول نویسی بر روی اطلاعات موجود را بدهیم، بسیار مفید است.

مثال دوم) به استفاده کننده از API کتابخانه خود، اجازه فرمول نویسی بدهید

برای نمونه مثال ساده زیر را درنظر بگیرید که در آن قرار است یک سری عدد که از منبع داده‌ای دریافت شده‌اند، بر روی صفحه نمایش داده شوند:

```
public static void PrintNumbers()
{
    var numbers = new[] { 1,2,3,5,7,90 }; // from a data source
    foreach(var item in numbers)
    {
        Console.WriteLine(item);
    }
}
```

قصد داریم به برنامه نویس استفاده کننده از کتابخانه گزارش‌سازی خود، این اجازه را بدهیم که پیش از نمایش نهایی اطلاعات، بتواند توسط فرمولی که مشخص می‌کند، فرمت اعداد نمایش داده شده را تعیین کند.
روال کار اکثر ابزارهای گزارش‌سازی موجود، ارائه یک زبان اسکریپتی جدید برای حل این نوع مسایل است. اما با استفاده از Func و ... روش‌های Code first (بجای روش‌های Wizard first)، خیلی از این رنج و دردها را می‌توان ساده‌تر و بدون نیاز به اختراع و یا آموزش زبان جدیدی حل کرد:

```
public static void PrintNumbers(Func<int,string> formula)
{
    var numbers = new[] { 1,2,3,5,7,90 };  // from a data source
    foreach(var item in numbers)
    {
        var data = formula(item);
        Console.WriteLine(data);
    }
}
```

اینبار با استفاده از Func، امکان فرمول نویسی را به کاربر استفاده کننده از API ساده گزارش ساز فرضی خود داده‌ایم. Func تعریف شده در اینجا یک عدد int را در اختیار استفاده کننده قرار می‌دهد. در این بین، برنامه نویس می‌تواند هر نوع تغییر یا هر نوع فرمولی را که مایل است بر روی این عدد به کمک دستور زبان جاری مورد استفاده، اعمال کند و در آخر تنها باید نتیجه این عملیات را به صورت یک string بازگشت دهد. برای مثال:

```
PrintNumbers(number => string.Format("{0:n0}",number));
```

البته سطر فوق ساده شده فراخوانی زیر است:

```
PrintNumbers((number) =>{ return string.Format("{0:n0}",number); });
```

به این ترتیب اعداد نهایی با جدا کننده سه رقمی نمایش داده خواهند شد.
از این نوع طراحی، در ابزارها و کتابخانه‌های جدید گزارش سازی مخصوص ASP.NET MVC زیاد مشاهده می‌شوند.

مثال سوم) حذف کدهای تکراری برنامه

فرض کنید قصد دارید در برنامه وب خود مباحث caching را پیاده سازی کنید:

```
using System;
using System.Web;
using System.Web.Caching;
using System.Collections.Generic;

namespace WebToolkit
{
    public static class CacheManager
    {
        public static void CacheInsert(this HttpContextBase httpContext, string key, object data, int
durationMinutes)
        {
            if (data == null) return;
            httpContext.Cache.Add(
                key,
                data,
                null,
                DateTime.Now.AddMinutes(durationMinutes),
                TimeSpan.Zero,
                CacheItemPriority.AboveNormal,
                null);
        }
    }
}
```

در هر قسمتی از برنامه که قصد داشته باشیم اطلاعاتی را در کش ذخیره کنیم، الگوی تکراری زیر باید طی شود:

```
var item = httpContext.Cache[key];
if (item == null)
{
    item = ReadDataFromDataSource();
    if (item == null)
          return null;

    CacheInsert(httpContext, key, item, durationMinutes);
}
```

ابتدا باید وضعیت کش جاری بررسی شود؛ اگر اطلاعاتی در آن موجود نبود، ابتدا از منبع داده‌ای مورد نظر خوانده شده و سپس در کش درج شود.
می‌توان در این الگوی تکراری، خواندن اطلاعات را از منبع داده، به یک Func واگذار کرد و به این صورت کدهای ما به نحو زیر بازسازی خواهند شد:

```
using System;
using System.Web;
using System.Web.Caching;
using System.Collections.Generic;

namespace WebToolkit
{
    public static class CacheManager
    {
        public static void CacheInsert(this HttpContextBase httpContext, string key, object data, int
durationMinutes)
        {
            if (data == null) return;
            httpContext.Cache.Add(
                key,
                data,
                null,
                DateTime.Now.AddMinutes(durationMinutes),
                TimeSpan.Zero,
                CacheItemPriority.AboveNormal,
                null);
        }

        public static T CacheRead<T>(this HttpContextBase httpContext, string key, int durationMinutes,
Func<T> ifNullRetrievalMethod)
        {
            var item = httpContext.Cache[key];
            if (item == null)
            {
```

```
                item = ifNullRetrievalMethod();
                if (item == null)
                    return default(T);

                CacheInsert(httpContext, key, item, durationMinutes);
            }
            return (T)item;
        }
    }
}
```

و استفاده از آن نیز به نحو زیر خواهد بود:

```
var user = HttpContext.CacheRead(
                            "Key1",
                            15,
                            () => _usersService.FindUser(userId));
```

پارامتر سوم متد CacheRead به صورت خودکار تنها زمانیکه اطلاعات کش متناظری با کلید Key1 وجود نداشته باشند، اجرا شده و نتیجه در کش ثبت می‌گردد. در اینجا دیگر از if و else و کدهای تکراری بررسی وضعیت کش خبری نیست.

نظرات خوانندگان

نویسنده: علی علیار
تاریخ: ۱۳۹۱/۰۶/۱۵ ۲۰:۲

سلام میشه یه توضیحی درباره کد زیر بدید؟

```
public static T CacheRead<T>(this HttpContextBase httpContext, string key, int durationMinutes, Func<T>
ifNullRetrievalMethod)
        {
            var item = httpContext.Cache[key];
            if (item == null)
            {
                item = ifNullRetrievalMethod();
                if (item == null)
                    return default(T);

                CacheInsert(httpContext, key, item, durationMinutes);
            }
            return (T)item;
        }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۵ ۲۰:۱۳

در متن توضیح دادم «... الگوی تکراری زیر باید طی شود ...».
برای خواندن اطلاعات از کش سیستم، این الگوی تکراری در هرجایی از برنامه باید انجام شود:
الف) ابتدا باید به شیء Cache مراجعه شود. شاید بر اساس یک key مفروض، اطلاعاتی در آن موجود باشد.
ب) اگر نبود (قطعه if تعریف شده)، باید به یک منبع داده مشخص، مراجعه و اطلاعات دریافت شود. سپس این اطلاعات در کش برای دفعات مراجعه بعدی ثبت گردد.
ج) اطلاعات نهایی بازگشت داده شود.

در اینجا قسمت مراجعه به منبع داده، توسط Func به استفاده کننده از متد CacheRead واگذار شده است. به این صورت ما فقط می‌دونیم که یک تابع در اختیار این متد قرار خواهد گرفت که در زمان مناسب می‌شود آن را فراخوانی کرد.
مثلا در مثالی که در انتهای بحث است یک نمونه از کاربرد آن‌را مشاهده می‌کنید.