عنوان: **عمومی سازی الگوریتم‌ها با استفاده از Reflection**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۳/۰۳ ۱۰:۲۵
آدرس: www.dotnettips.info
گروه‌ها: MVC, Reflection

در این مقاله قصد داریم عملیات Reflection را بیشتر در انجام ساده‌تر عملیات ببینیم. عملیاتی که به همراه کار اضافه، تکراری و خسته کننده است و با استفاده از Reflection این کارها حذف شده و تعداد خطوط هم پایین می‌آید. حتی گاها ممکن است موجب استفاده‌ی مجدد از کدها شود که همگی این عوامل موجب بالا رفتن امتیاز Refactoring می‌شوند.
در مثال‌های زیر مجموعه‌ای از Reflectionهای ساده و کاملا کاربردی است که من با آن‌ها روبرو شده ام.

کوتاه سازی کدهای نمایش یک View در ASP.NET MVC با Reflection

یکی از قسمت‌هایی که مرتبا با آن سر و کار دارید، نمایش اطلاعات است. حتی یک جدول را هم که می‌خواهید بسازید، باید ستون‌های آن جدول را یک به یک معرفی کنید. ولی در عمل، یک Reflection ساده این کار به یک تابع چند خطی و سپس برای ترسیم هر ستون جدول از دو خط استفاده خواهید کرد ولی مزیتی که دارد این است که این تابع برای تمامی جدول‌ها کاربردی عمومی پیدا می‌کند. برای نمونه دوست داشتم برای بخش مدیر، قسمت پروفایلی را ایجاد کنم و در آن اطلاعاتی چون نام، نام خانوادگی، تاریخ تولد، تاریخ ایجاد و خیلی از اطلاعات دیگر را نمایش دهم. به جای اینکه بیایم برای هر قسمت یک خط partial ایجاد کنم، با استفاده از reflection و یک حلقه، تمامی اطلاعات را به آن پارشال پاس می‌کنم. مزیت این روش این است که اگر بخواهم در یک جای دیگر، اطلاعات یک محصول یا یک فاکتور را هم نمایش دهم، باز هم همین تابع برایم کاربرد خواهد داشت:

تصویر زیر را که برگرفته از یک قالب Bootstrap است، ملاحظه کنید. اصلا علاقه ندارم که برای یک به یک آن‌ها، یک سطر جدید را تعریف کنم و به View بگویم این پراپرتی را نشان بده؛ دوباره مورد بعدی هم به همین صورت و دوباره و دوباره و ... . دوست دارم یک تابع عمومی، همه‌ی این کارها را خودکار انجام دهد.

First Name	: Jonathan	Last Name	: Smith
Country	: Australia	Birthday	: 13 July 1983
Occupation	: UI Designer	Email	: jsmith@flatlab.com
Mobile	: (12) 03 4567890	Phone	: 88 (02) 123456

ساختار اطلاعاتی تصویر فوق به شرح زیر است:

```
<div>
                            <div>
                                <div>
                                    <p><span>First Name </span>: Jonathan</p>
                                </div>
   </div>
                        </div>
```

که به دو فایل پارشال تقسیم شده است _Bio و _BioRow که محتویات هر پارشال هم به شرح زیر است:

_BioRow

```
@model System.Web.UI.WebControls.ListItem
```

```
<div>
    <p><span>@Model.Text </span>: @Model.Value</p>
</div>
```

در پارشال بالا ورودی از نوع listItem است که یک متن دارد و یک مقدار.(شاید به نظر شبیه حالت جفت کلید و مقدار باشد ولی در این کلاس خبری از کلید نیست).

پارشال پایینی هم دربرگیرنده‌ی پارشال بالاست که قرار است چندین و چند بار پارشال بالا در خودش نمایش دهد.
_Bio

```
@using System.Web.UI.WebControls
@using ZekrWepApp.Filters
@model ZekrModel.Admin

    <div>
        <h1>Bio Graph</h1>
        <div>

            @{
                ListItemCollection collection = GetCustomProperties.Get(Model,exclude:new
string[]{"Poems","Id"});
                foreach (var item in collection)
                {
                    Html.RenderPartial(MVC.Shared.Views._BioRow, item);
                }
            }

                    </div>
    </div>
```

پارشال بالا یک مدل از کلاس Admin را می‌پذیرد که قرار است اطلاعات شخصی مدیر را نمایش دهد. در ابتدا متدی از یک کلاس ایستا وجود دارد که کدهای Reflection درون آن قرار دارند که یک مجموعه از ListItemها را بر می‌گرداند و سپس با یک حلقه، پارشال _BioRow را صدا می‌زند.

کد درون این کلاس ایستا را بررسی می‌کنیم؛ این کلاس دو متد دارد یکی عمومی و دیگری خصوصی است:

```
public class GetCustomProperties
    {
        private static PropertyInfo[] getObjectsInfos(object obj,string[] inclue,string[] exclude )
        {
            var list = obj.GetType().GetProperties();

            PropertyInfo[] outputPropertyInfos = null;

            if (inclue != null)
            {
                return list.Where(propertyInfo => inclue.Contains(propertyInfo.Name)).ToArray();
            }
            if (exclude != null)
            {
                return list.Where(propertyInfo => !exclude.Contains(propertyInfo.Name)).ToArray();
            }
            return list;
        }
    }
```

کد بالا که یک کد خصوصی است، سه پارامتر را می‌پذیرد. اولی مدل یا کلاسی است که به آن پاس کرده‌ایم. دو پارامتر بعدی اختیاری است و در کد پارشال بالا Exclude را تعریف کرده ایم و تنهای یکی از دو پارامتر بالا هم باید مورد استفاده قرار بگیرند و Include ارجحیت دارد. وظیفه‌ی این دو پارمتر این است که آرایه ای از رشته‌ها را دریافت می‌کنند که نام پراپرتی‌ها در آن‌ها ذکر شده است. آرایه Include می‌گوید که فقط این پراپرتی‌ها را برگردان ولی اگر دوست دارید همه‌ی پارامترها را نمایش دهید و تنها یکی یا چندتا از آن‌ها را حذف کنید، از آرایه Exclude استفاده کنید. در صورتی که این دو آرایه خالی باشند، همه‌ی پراپرتی‌ها

بازگشت داده می‌شوند و در صورتی که یکی از آن‌ها وارد شده باشد، طبق دستورات Linq بالا بررسی می‌کند که (Include) آیا اسامی مشترکی بین آن‌ها وجود دارد یا خیر؟ اگر وجود دارد آن را در لیست قرار داده و بر می‌گرداند و در حالت Exclude این مقایسه به صورت برعکس انجام می‌گیرد و باید لیستی برگردانده شود که اسامی، نکته مشترکی نداشته باشند.

متد عمومی که در این کلاس قرار دارد به شرح زیر است:

```
public static ListItemCollection Get(object obj,string[] inclue=null,string[] exclude=null)
        {
            var propertyInfos = getObjectsInfos(obj, inclue, exclude);
            if (propertyInfos == null) throw new ArgumentNullException("propertyInfos is null");

            var collection = new ListItemCollection();

            foreach (PropertyInfo propertyInfo in propertyInfos)
            {


                string name = propertyInfo.Name;

                foreach (Attribute attribute in propertyInfo.GetCustomAttributes(true))
                {
                    DisplayAttribute displayAttribute = attribute as DisplayAttribute;

                    if (displayAttribute != null)
                    {
                        name = displayAttribute.Name;
                        break;
                    }
                }

                string value = "";
                object objvalue = propertyInfo.GetValue(obj);
                if (objvalue != null) value = objvalue.ToString();

                collection.Add(new ListItem(name,value));
            }
            return collection;
        }
```

این متد سه پارامتر را از کاربر دریافت و به سمت متد خصوصی ارسال می‌کند. موقعی‌که پراپرتی‌ها بازگشت داده می‌شوند، دو قسمت آن مهم است؛ یکی عنوان پراپرتی و دیگری مقدار پراپرتی. از آن جا که نام پراپرتی‌ها طبق سلیقه‌ی برنامه نویس و با حروف انگلیسی نوشته می‌شوند، در صورتی که برنامه نویس از متادیتای Display در مدل بهره برده باشد، به جای نام پراپرتی مقداری را که به متادیتای Display داده‌ایم، بر می‌گردانیم.

کد بالا پراپرتی‌ها را دریافت و یک به یک متادیتاهای آن را بررسی کرده و در صورتی که از متادیتای Display استفاده کرده باشند، مقدار آن را جایگزین نام پراپرتی خواهد کرد. در مورد مقدار هم از آنجا که اگر پراپرتی با Null پر شده باشد، تبدیل به رشته‌ای با پیام خطای روبرو خواهد شد. در نتیجه بهتر است یک شرط احتیاط هم روی آن پیاده شود. در آخر هم از متن و مقدار، یک آیتم ساخته و درون Collection اضافه می‌کنیم و بعد از اینکه همه پراپرتی‌ها بررسی شدند، Collection را بر می‌گردانیم.

```
[Display(Name = "نام کاربری")]
public string UserName { get; set; }
```

کد کامل کلاس:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Linq.Expressions;
using System.Reflection;
using System.Web;
using System.Web.Mvc.Html;
using System.Web.UI.WebControls;
using Links;

namespace ZekrWepApp.Filters
```

```
{

    public class GetCustomProperties
    {
        public static ListItemCollection Get(object obj,string[] inclue=null,string[] exclude=null)
        {
            var propertyInfos = getObjectsInfos(obj, inclue, exclude);
            if (propertyInfos == null) throw new ArgumentNullException("propertyInfos is null");

            var collection = new ListItemCollection();

            foreach (PropertyInfo propertyInfo in propertyInfos)
            {
                string name = propertyInfo.Name;
                foreach (Attribute attribute in propertyInfo.GetCustomAttributes(true))
                {
                    DisplayAttribute displayAttribute = attribute as DisplayAttribute;

                    if (displayAttribute != null)
                    {
                        name = displayAttribute.Name;
                        break;
                    }
                }

                string value = "";
                object objvalue = propertyInfo.GetValue(obj);
                if (objvalue != null) value = objvalue.ToString();

                collection.Add(new ListItem(name,value));
            }
            return collection;
        }
        private static PropertyInfo[] getObjectsInfos(object obj,string[] include,string[] exclude )
        {
            var list = obj.GetType().GetProperties();

            PropertyInfo[] outputPropertyInfos = null;

            if (include != null)
            {
                return list.Where(propertyInfo => include.Contains(propertyInfo.Name)).ToArray();
            }
            if (exclude != null)
            {
                return list.Where(propertyInfo => !exclude.Contains(propertyInfo.Name)).ToArray();
            }
            return list;
        }
    }
}
```

لیستی از پارامترها با Reflection

مورد بعدی که ساده‌تر بوده و از کد بالا مختصرتر هم هست، این است که قرار بود برای یک درگاه، یک سری اطلاعات را با متد Post ارسال کنم که نحوه‌ی ارسال اطلاعات به شکل زیر بود:

```
amount=1000&orderId=452&Pid=xxx&....
```

کد زیر را من جهت ساخت قالب‌های این چنینی استفاده می‌کنم:

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

namespace Utils
{
    public class QueryStringParametersList
    {
        private string Symbol = "&";
        private List<KeyValuePair<string, string>> list { get; set; }
```

```
        public QueryStringParametersList()
        {
            list = new List<KeyValuePair<string, string>>();
        }
        public QueryStringParametersList(string symbol)
        {
            Symbol = symbol;
           list = new List<KeyValuePair<string, string>>();
        }

        public int Size
        {
            get { return list.Count; }
        }
        public void Add(string key, string value)
        {
            list.Add(new KeyValuePair<string, string>(key, value));
        }

        public string GetQueryStringPostfix()
        {
            return string.Join(Symbol, list.Select(p => Uri.EscapeDataString(p.Key) + "=" +
Uri.EscapeDataString(p.Value)));
        }
    }
}
```

یک متغیر به نام symbol دارد و در صورتی در شرایط متفاوت، قصد چسپاندن چیزی را به یکدیگر با علامتی خاص داشته باشید، این تابع می‌تواند کاربرد داشته باشد. این متد از یک لیست کلید و مقدار استفاده کرده و پارامترهایی را که به آن پاس می‌شود، نگهداری و سپس توسط متد GetQueryStringPostfix آن‌ها را با یکدیگر الحاق کرده و در قالب یک رشته بر می‌گرداند.
کاربرد Reflection در اینجا این است که من باید دوبار به شکل زیر، دو نوع اطلاعات متفاوت را پست کنم. یکی موقع ارسال به درگاه و دیگری موقع بازگشت از درگاه.

```
QueryStringParametersList queryparamsList = new QueryStringParametersList();

            ueryparamsList.Add("consumer_key", requestPayment.Consumer_Key);
            queryparamsList.Add("amount", requestPayment.Amount.ToString());
            queryparamsList.Add("callback", requestPayment.Callback);
            queryparamsList.Add("description", requestPayment.Description);
            queryparamsList.Add("email", requestPayment.Email);
            queryparamsList.Add("mobile", requestPayment.Mobile);
            queryparamsList.Add("name", requestPayment.Name);
            queryparamsList.Add("irderid", requestPayment.OrderId.ToString());
```

ولی با استفاده از کد Reflection که در بالاتر عنوان شد، باید نام و مقدار پراپرتی را گرفته و در یک حلقه آن‌ها را اضافه کنیم، بدین شکل:

```
private QueryStringParametersList ReadParams(object obj)
        {
            PropertyInfo[] propertyInfos = obj.GetType().GetProperties();

            QueryStringParametersList queryparamsList = new QueryStringParametersList();
            for (int i = 0; i < propertyInfos.Count(); i++)
            {

queryparamsList.Add(propertyInfos[i].Name.ToLower(),propertyInfos[i].GetValue(obj).ToString() );
            }
            return queryparamsList;
        }
```

در کد بالا هر بار پراپرتی‌های کلاس را خوانده و نام و مقدار آن‌ها را گرفته و به کلاس QueryString اضافه می‌کنیم. پارامتر ورودی این متد به این خاطر object در نظر گرفته شده است که تا هر کلاسی را بتوانیم به آن پاس کنیم که خودم در همین کلاس درگاه، دو کلاس را به آن پاس کردم.