عنوان: **آشنایی با Refactoring - قسمت 9**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۷/۲۵ ۲۲:۴۰:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: Refactoring

این قسمت از آشنایی با Refactoring به کاهش cyclomatic complexity اختصاص دارد و خلاصه آن این است: «استفاده از if های تو در تو بیش از سه سطح، مذموم است» به این علت که پیچیدگی کدهای نوشته شده را بالا برده و نگهداری آن‌ها را مشکل می‌کند. برای مثال به شبه کد زیر دقت کنید:

```
if
   if
     if
       if
         do something
       endif
     endif
   endif
 endif
```

که حاصل آن شبیه به نوک یک پیکان (Arrow head) شده است. یک مثال بهتر:

```
namespace Refactoring.Day9.RemoveArrowhead.Before
{
    public class Role
    {
        public string RoleName { set; get; }
        public string UserName { set; get; }
    }
}
```

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

namespace Refactoring.Day9.RemoveArrowhead.Before
{
    public class RoleRepository
    {
        private IList<Role> _rolesList = new List<Role>();

        public IEnumerable<Role> Roles { get { return _rolesList; } }

        public void AddRole(string username, string roleName)
        {
            if (!string.IsNullOrWhiteSpace(roleName))
            {
                if (!string.IsNullOrWhiteSpace(username))
                {
                    if (!IsInRole(username, roleName))
                    {
                        _rolesList.Add(new Role
                        {
                            UserName=username,
                            RoleName=roleName
                        });
                    }
                    else
                    {
                        throw new InvalidOperationException("User is already in this role.");
```

```
                    }
                }
                else
                {
                    throw new ArgumentNullException("username");
                }
            }
            else
            {
                throw new ArgumentNullException("roleName");
            }
        }

        public bool IsInRole(string username, string roleName)
        {
            return _rolesList.Any(x => x.RoleName == roleName && x.UserName == username);
        }
    }
}
```

متد AddRole فوق، نمونه‌ی بارز پیچیدگی بیش از حد حاصل از اعمال if های تو در تو است و ... بسیار متداول. برای حذف این نوک پیکان حاصل از if های تو در تو، از بالاترین سطح شروع کرده و شرط‌ها را برعکس می‌کنیم؛ با این هدف که هر چه سریعتر متد را ترک کرده و خاتمه دهیم:

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

namespace Refactoring.Day9.RemoveArrowhead.After
{
    public class RoleRepository
    {
        private IList<Role> _rolesList = new List<Role>();

        public IEnumerable<Role> Roles { get { return _rolesList; } }

        public void AddRole(string username, string roleName)
        {
            if (string.IsNullOrWhiteSpace(roleName))
                throw new ArgumentNullException("roleName");

            if (string.IsNullOrWhiteSpace(username))
                throw new ArgumentNullException("username");

            if (IsInRole(username, roleName))
                throw new InvalidOperationException("User is already in this role.");

            _rolesList.Add(new Role
            {
                UserName = username,
                RoleName = roleName
            });
        }

        public bool IsInRole(string username, string roleName)
        {
            return _rolesList.Any(x => x.RoleName == roleName && x.UserName == username);
        }
    }
}
```

اکنون پس از اعمال این Refactoring ، متد AddRole بسیار خواناتر شده و هدف اصلی آن که اضافه کردن یک شیء به لیست نقش‌ها است، واضح‌تر به نظر می‌رسد. به علاوه اینبار قسمت‌های مختلف متد AddRole، فقط یک کار را انجام می‌دهند و وابستگی‌های آن‌ها به یکدیگر نیز کاهش یافته است.

نظرات خوانندگان

نویسنده: Farhad Yazdan-Panah
تاریخ: ۲۳:۴۳:۳۹ ۱۳۹۰/۰۷/۲۵

نکته جالب تولید کد میانی کمتر و واضحتر نیز هست (به دلیل عمق کمتر درخت تصمیم).

نویسنده: M.Safdel
تاریخ: ۱۰:۴۷:۳۲ ۱۳۹۰/۰۷/۲۶

سری مطالب Refactoring عالی هستن و از شما ممنونم. امیدوارم که همچنان ادامه داشته باشن. احتمالا همه برنامه نویسها مثل خودم خیلی از این روشها را بصورت تجربی می دونن ولی اینکه این روشها در قالبهای خاص ارائه بشن خیلی جالبه

نویسنده: HIWA NAZARI
تاریخ: ۱۴:۵۳:۱۴ ۱۳۹۰/۰۷/۲۶

سلام می بخشید که سوالم رو اینجا میپرسم ولی چاره ای نیست من می خوام تعدادی کنترل رو رو بصورت runtime در Asp.net به صفحه اضافه کنم سپس مقادیرش رو هم بخونم ولی مشکل اینجا ست زمانی که من میخوام به صفحه ای دیگه برم وبرگردم کنترل ها ازدست میرند بهتر بگم میخوام چیزی شبیه به پروفایل facebook باشه

نویسنده: وحید نصیری
تاریخ: ۱۶:۵۶:۱۰ ۱۳۹۰/۰۷/۲۶

برای اینکه احتمالا ASP.NET Webforms page life cycle رو رعایت نکردید و الان ViewState صفحه چیزی از وجود کنترل‌های پویای شما نمی‌دونه. مثلا می‌تونید از DynamicControlsPlaceholder استفاده کنید. اگر جزئیات بیشتری نیاز داشتید این مطالب مفید هستند:

How To Perpetuate Dynamic Controls Between Page Views in ASP.NET

Dynamic Web Controls, Postbacks, and View State

Creating Dynamic Data Entry User Interfaces

(ASP.Net Dynamic Controls (Part 1

(ASP.Net Dynamic Controls (Part 2

(ASP.Net Dynamic Controls (Part 3

(ASP.Net Dynamic Controls (Part 4

نویسنده: شاهین کیاست
تاریخ: ۱۵:۳۳ ۱۳۹۱/۰۶/۰۳

سلام ،
اگر فرض کنیم RoleRepository مطلب جاری پیاده سازی منطق تجاری قسمت کاربران در یک پروژه‌ی ASP.NET MVC می‌باشد، این استثناءها کجا باید مدیریت شوند ؟ در بدنه‌ی Controller ؟
به عبارتی دیگر بهتر است نوع بازگشتی لایه‌ی سرویس یک شیء باشد که موفقیت / عدم موفقیت عملیات به همراه پیغام خطا را بازگرداند یا اینکه در صورت صحیح نبودن روند مثلا تکراری بودن نام کاربری Exception ارسال شود و استفاده کننده از Service مثل Controller مسئولیت Handle کردن استثناءها را بر عهده بگیرد ؟

نویسنده: وحید نصیری
تاریخ: ۱۶:۵۳ ۱۳۹۱/۰۶/۰۳

من تا حد امکان هیچ نوع استثنایی رو مدیریت نمی‌کنم. استثناء یعنی مشکل و باید کاربر با کرش برنامه متوجه آن بشود. فقط برای لاگ کردن خطاهای برنامه‌های ASP.NET از ELMAH استفاده می‌کنم به علاوه تنظیم نمایش صفحه خطای عمومی. بیشتر از این نیازی

نیست کاری انجام شود. اولین اصل مدیریت خطاها، عدم مدیریت آن‌ها است .