عنوان: **عبارت using و نحوه استفاده صحیح از آن**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۲ ۱۳:۳۱
آدرس: www.dotnettips.info
برچسب‌ها: C#, Refactoring, iTextSharp

مثال ساده زیر را که در مورد تعریف یک کلاس Disposable و سپس استفاده از آن توسط عبارت using است را به همراه سه استثنایی که در این متدها تعریف شده است، در نظر بگیرید:

```
using System;

namespace TestUsing
{
    public class MyResource : IDisposable
    {
        public void DoWork()
        {
            throw new ArgumentException("A");
        }

        public void Dispose()
        {
            throw new ArgumentException("B");
        }
    }

    public static class TestClass
    {
        public static void Test()
        {
            using (MyResource r = new MyResource())
            {
                throw new ArgumentException("C");
                r.DoWork();
            }
        }
    }
}
```

به نظر شما قطعه کد زیر چه عبارتی را نمایش می‌دهد؟ C یا B یا A؟

```
try
{
     TestClass.Test();
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```

پاسخ: برخلاف تصور (که احتمالا C است؛ چون قبل از فراخوانی متد DoWork سبب بروز استثناء شده است)، فقط B را در خروجی مشاهده خواهیم کرد!
و این دقیقا مشکلی است که در حین کار با کتابخانه iTextSharp برای اولین بار با آن مواجه شدم. روش استفاده متداول از iTextSharp به نحو زیر است:

```
using (var pdfDoc = new Document(PageSize.A4))
{
   //todo: ...
}
```

در این بین هر استثنایی رخ دهد، در لاگ‌های خطای سیستم شما تنها خطاهای مرتبط با خود iTextSharp را مشاهده خواهید کرد و نه مشکل اصلی را که در کدهای ما وجود داشته است. البته این یک مشکل عمومی است و اگر « using statement and suppressed exceptions » را در گوگل جستجو کنید به نتایج مشابه زیادی خواهید رسید.
و خلاصه نتایج هم این است:

اگر به ثبت جزئیات خطاهای سیستم اهمیت می‌دهید (یکی از مهم‌ترین مزیت‌های دات نت نسبت به بسیاری از فریم ورک‌های مشابه که حداکثر خطای 0xABC12EF را نمایش می‌دهند)، از using استفاده نکنید! using در پشت صحنه به try/finally ترجمه می‌شود و بهتر است این مورد را دستی نوشت تا اینکه کامپایلر اینکار را به صورت خودکار انجام دهد.
در اینجا باز هم به یک سری کد تکراری try/finally خواهیم رسید و همانطور که در مباحث کاربردهای Action و Func در این سایت ذکر شد، می‌توان آن‌را تبدیل به کدهایی با قابلیت استفاده مجدد کرد. یک نمونه از پیاده سازی آن‌را در این سایت « C# Using Blocks can Swallow Exceptions » می‌توانید مشاهده کنید که خلاصه آن کلاس زیر است:

```
using System;

namespace Guard
{
    public static class SafeUsing
    {
        public static void SafeUsingBlock<TDisposable>(this TDisposable disposable, Action<TDisposable>
action)
            where TDisposable : IDisposable
        {
            disposable.SafeUsingBlock(action, d => d);
        }

        internal static void SafeUsingBlock<TDisposable, T>(this TDisposable disposable, Action<T>
action, Func<TDisposable, T> unwrapper)
            where TDisposable : IDisposable
        {
            try
            {
                action(unwrapper(disposable));
            }
            catch (Exception actionException)
            {
                try
                {
                    disposable.Dispose();
                }
                catch (Exception disposeException)
                {
                    throw new AggregateException(actionException, disposeException);
                }

                throw;
            }

            disposable.Dispose();
        }
    }
}
```

برای استفاده از کلاس فوق مثلا در حالت بکارگیری iTextSharp خواهیم داشت:

```
new Document(PageSize.A4).SafeUsingBlock(pdfDoc =>
{
  //todo: ...
});
```

علاوه بر اینکه SafeUsingBlock یک سری از اعمال تکراری را کپسوله می‌کند، از AggregateException نیز استفاده کرده است (معرفی شده در دات نت 4). به این صورت چندین استثنای رخ داده نیز در سطحی بالاتر قابل دریافت و بررسی خواهند بود و استثنایی در این بین از دست نخواهد رفت.

نظرات خوانندگان

نویسنده: بهمن خلفی
تاریخ: ۱۳۹۱/۰۶/۱۳ ۷:۵۹

با سلام خدمت شما جناب نصیری
با توجه به این مطلب که زیاد هم قدیمی نیست به نظر شما استفاده از using پیشنهاد میشود یا استفاده از try catch ؟
چون بنده در برنامه‌های کاربردی تحت وب بیشتر از using استفاده میکنم و از Elmah هم برای ثبت خطاهای سیستم استفاده میکنم .

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۳ ۸:۴۲

مطلب جاری بیشتر به شبیه سازی **finally** /try معادل using که توسط کامپایلر به صورت خودکار تولید می‌شود مرتبط است نه try/catch کلی. بحث dispose خودکار اشیاء disposable و اینکه استفاده از using به دلیلی که عنوان شد مناسب نیست. بنابراین بجای using از SafeUsingBlock استفاده کنید (شبیه سازی بهتر کاری است که کامپایلر در پشت صحنه جهت معادل سازی یا پیاده سازی using انجام می‌دهد؛ اما بدون از دست رفتن استثناهای رخ داده). مابقی را هم ELMAH انجام می‌دهد.
اگر از using استفاده کنید و ELMAH، فقط خطاهای مرتبط با مثلا iTextSharp رو در لاگ‌ها خواهید یافت؛ مثلا شیء document آن dispose شده، اما خطا و مشکل اصلی که به کدهای ما مرتبط بوده و نه iTextSharp، این میان گم خواهد شد. اما با استفاده از SafeUsingBlock ، دلیل اصلی نیز لاگ می‌شود.

نویسنده: مرادی
تاریخ: ۱۳۹۱/۰۶/۱۳ ۹:۹

البته از حق هم نمی‌شه گذشت که طراح‌های iText Sharp در این جا هم از Best Practiceها پیروی نکردند
این قاعده کلی کار هستش که در Dispose و متد Finalize خطایی ایجاد نشه
حتی چند بار فراخوانی Dispose هم نباید ایجاد خطا کنه، حتی خطای Object Disposed Exception
متاسفانه تفاوت‌های Java و .NET. تونسته رو این تیم که iText Sharp رو از Java به .NET. برده، رو به یک سری اشتباهات بندازه، مثل این مورد، و عدم استفاده از Enum و چند تا مورد دیگه
اگه فرض کنیم ما Dispose رو فراخونی نکنیم و این فراخونی توسط CLR و در فاینالایزر کلاس رخ بده، این خطایی موجود در Dispose می‌تونه مسائل بدتری رو هم در پی داشته باشه
به دوستان توصیه می‌کنم حتما با Best Practiceهای مدیریت خطا مخصوص .NET. آشنا بشن، که در اینترنت منابع زیادی برای این مهم موجوده

نویسنده: افشین
تاریخ: ۱۳۹۱/۰۶/۱۳ ۱۳:۱۰

واقا خسته نباشید.
هر روز بابات این سایت شما رو دعا میکنیم.
ببخشید این موضوع رو این جا مطرح میکنم چون راه دیگه ای پیدا نکردم

نویسنده: سید امیر سجادی
تاریخ: ۱۳۹۲/۰۲/۰۵ ۹:۴۹

سلام. من این مشکلی که گفتید رو توی VB.NET تست کردم مشکلی نداشت. آیا .NET این مشکل رو داره و یا فقط C#?

نویسنده: وحید نصیری

تاریخ: ۱۳۹۲/۰۲/۰۵ ۱۰:۱۴

این رفتار در VB.NET هم قابل مشاهده است:

```
Public Class MyResource
    Implements IDisposable
    Public Sub DoWork()
        Throw New ArgumentException("A")
    End Sub

    Public Overloads Sub Dispose() Implements System.IDisposable.Dispose
        Throw New ArgumentException("B")
    End Sub
End Class

Public NotInheritable Class TestClass
    Private Sub New()
    End Sub
    Public Shared Sub Test()
        Using r As New MyResource()
            Throw New ArgumentException("C")
            r.DoWork()
        End Using
    End Sub
End Class
Module Module1

    Sub Main()
        Try
            TestClass.Test()
        Catch ex As Exception
            Console.WriteLine(ex.Message)
        End Try
    End Sub

End Module
```

عبارت نمایش داده شده در اینجا هم B است.