عنوان: **خلاص شدن از شر deep null check**
نویسنده: رفیعی
تاریخ: ۱۳۹۳/۰۱/۳۰ ۱۲:۴۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Tips

آیا تا به حال مجبور به نوشتن کدی شبیه قطعه کد زیر شده اید؟

```
var store = GetStore();
string postCode = null;
if (store != null && store.Address != null && store.Address.PostCode != null)
     postCode = store.Address.PostCode.ToString();
```

بله! من مطمئن هستم برای شما هم پیش آمده است.

هدف بازیابی و یا محاسبه یک مقدار است، اما برای انجام این کار می‌بایست به چندین شیء میانی دسترسی پیدا کنیم که البته ممکن است در حالت پیش فرض خود قرار داشته باشند و حاوی هیچ مقداری نباشند. بنابراین برای جلوگیری از وقوع NullException ، مجبوریم تمامی اشیائی که در مسیر قرار دارند را بررسی کنیم که null نباشند. مثال بالا کاملا گویا ست. گاهی اوقات حتی ممکن است فراخوانی یک متد، تبدیل نوع با استفاده از as و یا دسترسی به عناصر یک مجموعه وجود داشته باشد. متاسفانه مدیریت تمامی این حالات باعث حجیم شدن کدها و در نتیجه کاهش خوانایی آنها می‌شود. بنابراین باید به دنبال یک راه حل مناسب بود.

استفاده از یک متد الحاقی شرطی (Conditional extensions)

از نظر بسیاری از برنامه نویس‌ها راه حل، استفاده از یک متد الحاقی شرطی است. اگر عبارت "c# deep null check" را گوگل کنید، پیاده سازی‌های متنوعی را پیدا خواهید کرد. اگر چه این متدها نام‌های متفاوتی دارند اما همه آن‌ها از یک ایده کلی مشترک استفاده می‌کنند:

```
public static TResult IfNotNull<TResult, TSource>(
    this TSource source,
    Func<TSource, TResult> onNotDefault)
    where TSource : class
{
    if (onNotDefault == null) throw new ArgumentNullException("onNotDefault");
    return source == null ? default(TResult) : onNotDefault(source);
}
```

همانطور که می‌بینید این متد الحاقی مقداری از نوع TResult را بر می‌گرداند. اگر source که در اینجا با توجه به الحاقی بودن متد به معنای شی جاری است، null باشد مقدار پیش فرض نوع خروجی(TResult) بازگردانده می‌شود و در غیر این صورت دیلیگیت onNotDefault فراخوانی می‌گردد.
بعد از افزودن متد الحاقی IfNotNull به پروژه می‌توانیم مثال ابتدای مطلب را به صورت زیر بنویسیم :

```
var postCode =
    GetStore()
        .IfNotNull(x => x.Address)
        .IfNotNull(x => x.PostCode)
        .IfNotNull(x => x.ToString());
```

این روش مزایای بسیاری دارد اما در موارد پیچیده دچار مشکل می‌شویم. برای مثال در نظر بگیرید قصد داریم در طول مسیر، متدی را فراخوانی کنیم و مقدار بازگشتی را در یک متغیر موقتی ذخیره کنیم و بر اساس آن ادامه مسیر را طی کنیم. متاسفانه این کارها هم اکنون امکان پذیر نیست. پس به نظر می‌رسد باید کمی متد الحاقی IfNotNull را بهبود ببخشیم.

برای بهبود عملکرد متد الحاقی IfNotNull علاوه بر موارد ذکر شده حداقل دو مورد به نظر من می‌رسد:
این متد فقط با انواع ارجاعی (reference types) کار می‌کند و می‌بایست برای کار با انواع مقداری (value types) اصلاح شود.

با انواع داده ای مثل string چه باید کرد؟ در مورد این نوع داده‌ها تنها مطمئن شدن از null نبودن کافی نیست. برای مثال در مورد string ، گاهی اوقات ما می‌خواهیم از خالی نبودن آن نیز مطمئن شویم. و یا در مورد collectionها تنها null نبودن کافی نیست بلکه زمانی که نیاز به محاسبه مجموع و یا یافتن بزرگترین عضو است، باید از خالی نبودن مجموعه و وجود حداقل یک عضو در آن مطمئن باشیم.

برای حل این مشکلات می‌توانیم متد الحاقی IfNotNull را به متد الحاقی IfNotDefault تبدیل کنیم:

```
public static TResult IfNotDefault<TResult, TSource>(
    this TSource source,
    Func<TSource, TResult> onNotDefault,
    Predicate<TSource> isNotDefault = null)
{
    if (onNotDefault == null) throw new ArgumentNullException("onNotDefault");
    var isDefault = isNotDefault == null
        ? EqualityComparer<TSource>.Default.Equals(source, default(TSource))
        : !isNotDefault(source);
   return isDefault ? default(TResult) : onNotDefault(source);
}
```

تعریف این متد خیلی با تعریف متد قبلی متفاوت نیست. به منظور پشتیبانی از struct ها، قید where TSource : class حذف شده است. بنابراین دیگر نمی‌توان از مقایسه‌ی ساده null با استفاده از عملگر == استفاده کرد چراکه structها nullable نیستند. پس مجبوریم از EqualityComparer<TSource>.Default بخواهیم که این کار را انجام دهد. متد الحاقی IfNotDefault همچنین شامل یک predicate اختیاری با نام isNotDefault است. در صورتی که مقایسه پیش فرض کافی نبود می‌توان از این predicate استفاده کرد.
در انتها اجازه بدهید چند مثال کاربردی را مرور کنیم:
1- انجام یک سری اعمال بر روی string در صورتی که رشته خالی نباشد:

```
return person
        . IfNotDefault(x => x.Name)
        . IfNotDefault(SomeOperation, x => !string.IsNullOrEmpty(x));
```

محاسبه‌ی مقدار میانگین. متد الحاقی IfNotDefault به زیبایی در یک زنجیره‌ی LINQ کار می‌کند:

```
var avg = students
        .Where(IsNotAGraduate)
        .FirstOrDefault()
        .IfNotDefault(s => s.Grades)
        .IfNotDefault(g => g.Average(), g => g != null && g.Length > 0);
```

برای مطالعه بیشتر

Get rid of deep null checks

Chained null checks and the Maybe monad

Maybe or IfNotNull using lambdas for deep expressions

Dynamically Check Nested Values for IsNull Values

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۳۰ ۱۳:۱۱

با تشکر از شما. در سی‌شارپ 6 ، برای این مورد ویژه قرار است عملگر جدیدی اضافه شود:

Null propagation	customer?.Orders?[5]?.$price	Planned	Planned

بحثی در این مورد

نویسنده: رفیعی
تاریخ: ۱۳۹۳/۰۱/۳۰ ۱۴:۷

بله همین طوره!ممنون از دقت نظرتون. اتفاقاً پیشنهاد این کار رو هم فردی به نام جمشید اسدزاده که احتمالا ایرانیه در قسمت پیشنهادهای مایکروسافت مطرح کرده.

نویسنده: حمید سامانی
تاریخ: ۱۳۹۳/۰۱/۳۰ ۱۷:۱۱

احتمالا ایشون SafeNavigationOperator توی زبان Groovy را قبلا دیده بودن :) .

نویسنده: یاسر مرادی
تاریخ: ۱۳۹۳/۰۱/۳۰ ۱۸:۲۶

متاسفانه روش فوق کد نویسی را تا حد زیادی تحت تاثیر قرار می‌دهد، مگر این که روش استفاده از متد الحاقی شما را به خوبی متوجه نشده باشم
به مثال زیر دقت کنید:

```
public class Customer
    {
        public CustomerInfo Info { get; set; }
        public Int32 GetNameLength()
        {
            return this.IfNotDefault(city => city.Info)
                .IfNotDefault(info => info.CityInfo)
                .IfNotDefault(cityInfo => cityInfo.Name)
                .IfNotDefault(name => name.Length);
        }
    }
    public class CustomerInfo
    {
        public CustomerCityInfo CityInfo { get; set; }
    }
    public class CustomerCityInfo
    {
        public String Name { get; set; }
    }
```

و برای استفاده داریم:

```
Customer customer = new Customer();
            String cityName = customer
                .IfNotDefault(cust => cust.Info)
                .IfNotDefault(info => info.CityInfo)
```

```
                .IfNotDefault(city => city.Name);
            Int32 length = customer.GetNameLength();
```

در حالی که با متد الحاقی زیر داریم

```
public static TValue GetValue<TObj, TValue>(this TObj obj, Func<TObj, TValue> member, TValue
defaultValueOnNull = default(TValue))
        {
            if (member == null)
                throw new ArgumentNullException("member");

            if (obj == null)
                throw new ArgumentNullException("obj");

            try
            {
                return member(obj);
            }
            catch (NullReferenceException)
            {
                return defaultValueOnNull;
            }
        }
```

تعریف ساده‌تر کلاس

```
public class Customer
    {
        public CustomerInfo Info { get; set; }

        public Int32 GetNameLength()
        {
            return this.Info.CityInfo.Name.Length;
        }
    }

    public class CustomerInfo
    {
        public CustomerCityInfo CityInfo { get; set; }
    }

    public class CustomerCityInfo
    {
        public String Name { get; set; }
    }
```

و سادگی در استفاده

```
Customer customer = new Customer();

            String cityName = customer.GetValue(cust => cust.Info.CityInfo.Name, "Not Selected");

            Int32 i = customer.GetValue(cust => cust.GetNameLength());
```

شاید بگویید استفاده از Try-Catch سیستم را کند می‌کند، البته نه در آن حدی که فکر می‌کنید، و اگر قسمتی از کد شما به تعداد زیادی در بازه‌ی زمانی کوتاه فراخوانی می‌شود، می‌توانید آنرا به صورت کاملا عادی بنویسید، چون واقعا تعداد این شرایط زیاد نیست و این مورد سناریوی فراگیری نیست، در عوض خوانایی کد بسیار بسیار بالاتر از حالات عادی است.
در ضمن دقت کنید که تا زمانی که خطای NullReference رخ ندهد، سرعت سیستم در حد همان حداقل نیز کاهش نمی‌یابد، بدین جهت که بسیاری از افراد فکر می‌کنند Try-Catch نوشتن به خودی خود برنامه را کند می‌کند، ولی این رخ دادن خطا و جمع آوری StackTrace و ... است که برنامه را کند می‌کند، که شاید در خیلی از موارد اصلا رخ ندهد.
البته کدهای نوشته صرفا نمونه کد است، به هیچ وجه اصول طراحی در آن رعایت نشده است، بلکه سعی کرده ام مثال واضح‌تری بزنم
موفق و پایدار باشید

نویسنده: رضا عرب
تاریخ: ۱۳۹۳/۰۱/۳۱ ۹:۲۶

با تشکر از شما من یک Extension Method نوشتم که به نظرم کار کردن باهاش راحتتره. GetValueOrDefault

نویسنده: رفیعی
تاریخ: ۱۳۹۳/۰۱/۳۱ ۱۰:۰

بدیهی است راه‌های زیادی برای این کار وجود دارد اگرچه هسته همه اون‌ها خیلی شبیه...
متد الحاقی IfNotDefault چند ویژگی مهم دارد :

همانطور که در متن ذکر شده، بحث فقط چک برای null نبودن نیست بلکه چک برای قرار نداشتن در حالت پیش فرضه! که در انواعی مثل string و collectionها خیلی مهمه.
گاهی اوقات هر کدام از اشیاء در طول زنجیره برای ما مهم هستند. متد الحاقی IfNotDefault این امکان را دارد که هر کدام از اشیاء جداگانه بررسی شوند. روش ارایه شده در C# 6.0 هم همینگونه است.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۰۲ ۰:۳۲

دو روش دیگر برای حل این مساله

- استفاده از روش‌های AOP مانند Minimizing the null ref with dynamic proxies
- استفاده از expression trees مانند Avoiding nulls with expression trees