عنوان: **تکمیل کلاس DelegateCommand**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۹/۱۹ ۱۳:۱۱:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVVM

مدت‌ها از کلاس DelegateCommand معرفی شده در این آدرس استفاده می‌کردم. این کلاس یک مشکل جزئی دارد و آن هم عدم بررسی مجدد قسمت canExecute به صورت خودکار هست.

**خلاصه‌ای برای کسانی که بار اول هست با این مباحث برخورد می‌کنند؛ یا MVVM به زبان بسیار ساده:**

در برنامه نویسی متداول سیستم مایکروسافتی، در هر سیستمی که ایجاد کرده و در هر فناوری که ارائه داده از زمان VB6 تا امروز، شما روی یک دکمه مثلا دوبار کلیک می‌کنید و در فایل اصطلاحا code behind این فرم و در روال رخدادگردان آن شروع به کد نویسی خواهید کرد. این مورد تقریبا در همه جا صادق است؛ از WinForms تا WPF تا Silverlight تا حتی ASP.NET Webforms . به عمد هم این طراحی صورت گرفته تا برنامه نویس‌ها در این محیط‌ها زیاد احساس غریبی نکنند. اما این روش یک مشکل مهم دارد و آن هم «توهم» جداسازی رابط کاربر از کدهای برنامه است. به ظاهر یک فایل فرم وجود دارد و یک فایل جدای code behind ؛ اما در عمل هر دوی این‌ها یک partial class یا به عبارتی «یک کلاس» بیشتر نیستند. «فکر می‌کنیم» که از هم جدا شدند اما واقعا یکی هستند. شما در code behind صفحه به صورت مستقیم با عناصر رابط کاربری سروکار دارید و کدهای شما به این عناصر گره خورده‌اند.
شاید بپرسید که چه اهمیتی دارد؟
مشکل اول: امکان نوشتن آزمون‌ها واحد برای این متدها وجود ندارد یا بسیار سخت است. این متدها فقط با وجود فرم و رابط کاربری متناظر با آن‌ها هست که معنا پیدا می‌کنند و تک تک عناصر آن‌ها وهله سازی می‌شوند.
مشکل دوم: کد نوشته فقط برای همین فرم جاری آن قابل استفاده است؛ چون به صورت صریح به عناصر موجود در فرم اشاره می‌کند. نمی‌تونید این فایل code behind رو بردارید بدون هیچ تغییری برای فرم دیگری استفاده کنید.
مشکل سوم: نمی‌تونید طراحی فرم رو بدید به یک نفر، کد نویسی اون رو به شخصی دیگر. چون ایندو لازم و ملزوم یکدیگرند.

این سیستم کد نویسی دهه 90 است.
چند سالی است که طراحان سعی کرده‌اند این سیستم رو دور بزنند و روش‌هایی رو ارائه بدن که در آن‌ها فرم‌های برنامه و فایل‌های پیاده سازی کننده‌ی منطق آن هیچگونه ارتباط مستقیمی باهم نداشته باشند؛ به هم گره نخورده باشند؛ ارجاعی به هیچیک از عناصر بصری فرم را در خود نداشته باشند. به همین دلیل ASP.NET MVC به وجود آمده و در همان سال‌ها مثلا MVVM .

سؤال:
الان که رابط کاربری از فایل پیاده سازی کننده منطق آن جدا شده و دیگر Code behind هم نیست (همان partial class های متداول)، این فایل‌ها چطور متوجه می‌شوند که مثلا روی یک فرم، شیءایی قرار گرفته؟ از کجا متوجه خواهند شد که روی دکمه‌ای کلیک شده؟ این‌ها که ارجاعی از فرم را در درون خود ندارند.
در الگوی MVVM این سیم کشی توسط امکانات قوی Binding موجود در WPF میسر می‌شود. در ASP.NET MVC چیزی شبیه به آن به نام Model binder و همان مکانیزم‌های استاندارد HTTP این کار رو می‌کنه. در MVVM شما بجای code behind خواهید داشت ViewModel (اسم جدید آن). در ASP.NET MVC این اسم شده Controller. بنابراین اگر این اسامی رو شنیدید زیاد تعجب نکنید.
این‌ها همان Code behind قدیمی هستند اما ... بدون داشتن ارجاعی از رابط کاربری در خود که ... اطلاعات موجود در فرم به نحوی به آن‌ها Bind و ارسال می‌شوند.
این سیم کشی‌ها هم نامرئی هستند. یعنی فایل ViewModel یا فایل Controller نمی‌دونند که دقیقا از چه کنترلی در چه فرمی این اطلاعات دریافت شده.
این ایده هم جدید نیست. شاید بد نباشه به دوران طلایی Win32 برگردیم. همان توابع معروف PostMessage و SendMessage را به خاطر دارید؟ شما در یک ترد می‌تونید با مثلا PostMessage شیءایی رو به یک فرم که در حال گوش فرا دادن به تغییرات است ارسال کنید (این سیم کشی هم نامرئی است). بنابراین پیاده سازی این الگوها حتی در Win32 و کلیه فریم ورک‌های ساخته شده بر

پایه آن‌ها مانند WinForms ، VB6 ، VCL و غیره ... «از روز اول» وجود داشته و می‌تونستند بعد از 10 سال نیان بگن که اون روش‌های RAD ایی رو که ما پیشنهاد دادیم، می‌شد خیلی بهتر از همان ابتدا، طور دیگری پیاده سازی بشه.

ادامه بحث!
این سیم کشی یا اصطلاحا Binding ، در مورد رخدادها هم در WPF وجود داره و اینبار به نام Commands معرفی شده‌است. به این معنا که بجای اینکه بنویسید:

```
<Button  Click="btnClick_Event">Last</Button>
```

بنویسید:

```
<Button Command="{Binding GoLast}">Last</Button>
```

حالا باید مکانیزمی وجود داشته باشه تا این پیغام رو به ViewModel برنامه برساند. اینکار با پیاده سازی اینترفیس ICommand قابل انجام است که معرفی یک کلاس عمومی از پیاده سازی آن‌را در ابتدای بحث مشاهده نمودید.
در یک DelegateCommand، توسط متد منتسب به executeAction، مشخص خواهیم کرد که اگر این سیم کشی برقرار شد (که ما دقیقا نمی‌دانیم و نمی‌خواهیم که بدانیم از کجا و کدام فرم دقیقا)، لطفا این اعمال را انجام بده و توسط متد منتسب به canExecute به سیستم Binding خواهیم گفت که آیا مجاز هستی این اعمال را انجام دهی یا خیر. اگر این متد false برگرداند، مثلا دکمه یاد شده به صورت خودکار غیرفعال می‌شود.
اما مشکل کلاس DelegateCommand ذکر شده هم دقیقا همینجا است. این دکمه تا ابد غیرفعال خواهد ماند. در WPF کلاسی وجود دارد به نام CommandManager که حاوی متدی استاتیکی است به نام InvalidateRequerySuggested . اگر این متد به صورت دستی فراخوانی شود، یکبار دیگر کلیه متدهای منتسب به تمام canExecute های تعریف شده، به صورت خودکار اجرا می‌شوند و اینجا است که می‌توان دکمه‌ای را که باید مجددا بر اساس شرایط جاری تغییر وضعیت پیدا کند، فعال کرد. بنابراین فراخوانی متد InvalidateRequerySuggested یک راه حل کلی رفع نقیصه‌ی ذکر شده است.
راه حل دومی هم برای حل این مشکل وجود دارد. می‌توان از رخدادگردان CommandManager.RequerySuggested استفاده کرد. روال منتسب به این رخدادگردان هر زمانی که احساس کند تغییری در UI رخ داده، فراخوانی می‌شود. بنابراین پیاده سازی بهبود یافته کلاس DelegateCommand به صورت زیر خواهد بود:

```
using System;
using System.Windows.Input;

namespace MvvmHelpers
{
    // Ref.
    // - http://johnpapa.net/silverlight/5-simple-steps-to-commanding-in-silverlight/
    // - http://joshsmithonwpf.wordpress.com/2008/06/17/allowing-commandmanager-to-query-your-icommand-
objects/
    public class DelegateCommand<T> : ICommand
    {
        readonly Func<T, bool> _canExecute;
        bool _canExecuteCache;
        readonly Action<T> _executeAction;

        public DelegateCommand(Action<T> executeAction, Func<T, bool> canExecute = null)
        {
            if (executeAction == null)
                throw new ArgumentNullException("executeAction");

            _executeAction = executeAction;
            _canExecute = canExecute;
        }

        public event EventHandler CanExecuteChanged
        {
            add { if (_canExecute != null) CommandManager.RequerySuggested += value; }
            remove { if (_canExecute != null) CommandManager.RequerySuggested -= value; }
```

```
        }

        public bool CanExecute(object parameter)
        {
            return _canExecute == null ? true : _canExecute((T)parameter);
        }

        public void Execute(object parameter)
        {
            _executeAction((T)parameter);
        }
    }
}
```

استفاده از آن هم در ViewModel ساده است. یکبار خاصیتی به این نام تعریف می‌شود. سپس در سازنده کلاس مقدار دهی شده و متدهای متناظر آن تعریف خواهند شد:

```
public DelegateCommand<string> GoLast { set; get; }

//in ctor
GoLast = new DelegateCommand<string>(goLast, canGoLast);

private bool canGoLast(string data)
{
    //ex.
    return ListViewGuiData.CurrentPage != ListViewGuiData.TotalPage - 1;
}

private void goLast(string data)
{
  //do something
}
```

مزیت کلاس DelegateCommand جدید هم این است که مثلا متد canGoLast فوق، به صورت خودکار با به روز رسانی UI ، فراخوانی و تعیین اعتبار مجدد می‌شود.

## نظرات خوانندگان

نویسنده: mohammad azad
تاریخ: ۱۳۹۰/۰۹/۱۹ ۱۷:۱۰:۴۷

سلام.این کلاس تفاوتی با کلاس RelayCommand داره؟ من از mvvmlight استفاده می کنم جالب اینجاست این بررسی خودکار canexequte رو در ورژن آخریش که البته بتا هست برداشته شایدم فراموش کرده!!!!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۹/۱۹ ۱۷:۲۱:۳۹

سلام؛ اگر از mvvm light استفاده می‌کنید نیازی به این نیست.
البته به نظر من این فریم ورک‌های تهیه شده فقط یک جمع آوری مطلب از وب هستند. بنابراین بهتر است کمی هم با پشت صحنه این‌ها آشنا شد که چرا مثلا از CommandManager.RequerySuggested استفاده شده. چرا روش جان پاپا که ابتدای بحث مثال زدم درست کار نمی‌کنه (حداقل برای WPF البته) یا مثلا متد CommandManager.InvalidateRequerySuggested چی هست و به چه علتی پیش بینی شده؟

نویسنده: mohammad azad
تاریخ: ۱۳۹۰/۰۹/۱۹ ۲۰:۱۴:۱۹

حق با شماست آقای نصیری.اما CommandManager.RequerySuggested تو آخریت نسخه mvvmlight استفاده نشده .البته نسخه 4.0 تا زمانی که من اطلاع داشتم نسخه بتا بود.برام جای سوال بود که تو ورژن های قبلی mvvmlight بود و بعد تو ورژن بالاتر حذف شد.البته شایدم فراموش شده یا هر چیز دیگه...سوالم هم این بود این کلاس با RelayCommand josh Smith چه فرقی داره؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۹/۱۹ ۲۰:۵۵:۰۷

RelayCommand موجود در mvvm light دقیقا نسخه معادل mvvm foundation است (یا بود یک زمانی).

نویسنده: امیری
تاریخ: ۱۳۹۰/۰۹/۲۰ ۰۰:۰۷:۵۷

درود بر شما؛ در راستای فرمایشات شما، ویدیوهای زیر میتونه برای دوستان جالب باشه:

http://channel9.msdn.com/posts/NYC-DevReady-MVVM-Session-2-of-5-Programming-with-MVVM-Part-1
http://channel9.msdn.com/posts/NYC-DevReady-MVVM-Session-3-of-5-Programming-with-MVVM-Part-2
تو بخش اول مفصلا راجع به DelegateCommand بحث شده؛ پاینده باشید.

نویسنده: Arcabdelahi
تاریخ: ۱۳۹۰/۰۹/۲۰ ۱۳:۱۸:۴۴

بسیار عالی با تشکر ازمطلب خوب شما که بسیار خوب توضیح داده اید.

نویسنده: mohsen bahrzadeh
تاریخ: ۱۳۹۰/۱۰/۰۵ ۰۰:۴۸:۰۱

راهی برای پیاده سازی این موضوع در سیلورلایت وجود دارد؟!!!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۰۵ ۰۱:۰۳:۳۹

command manager implementation for Silverlight