عنوان: **MVVM و نمایش دیالوگ‌ها**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۱۲ ۱۰:۱۱:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVVM

بسیاری از برنامه‌های دسکتاپ نیاز به نمایش پنجره‌های دیالوگ استاندارد ویندوز مانند OpenFileDialog و SaveFileDialog را دارند و سؤال اینجا است که چگونه اینگونه موارد را باید از طریق پیاده سازی صحیح الگوی MVVM مدیریت کرد؛ از آنجائیکه خیلی راحت در فایل ViewModel می‌توان نوشت new OpenFileDialog و الی آخر. این مورد هم یکی از دلایل اصلی استفاده از الگوی MVVM را زیر سؤال می‌برد : این ViewModel دیگر قابل تست نخواهد بود. همیشه شرایط آزمون‌های واحد را به این صورت در نظر بگیرید:
سروری وجود دارد در جایی که به آن دسترسی نداریم. روی این سرور با اتوماسیونی که راه انداخته‌ایم، آخر هر روز آزمون‌های واحد موجود به صورت خودکار انجام شده و یک گزارش تهیه می‌شود (مثلا یک نوع continuous integration سرور). بنابراین کسی دسترسی به سرور نخواهد داشت تا این OpenFileDialog ظاهر شده را مدیریت کرده، فایلی را انتخاب و به برنامه آزمون واحد معرفی کند. به صورت خلاصه ظاهر شدن هر نوع دیالوگی حین انجام آزمون‌های واحد «مسخره» است!
یکی از روش‌های حل این نوع مسایل، استفاده از dependency injection یا تزریق وابستگی‌ها است و در ادامه خواهیم دید که چگونه WPF بدون نیاز به هیچ نوع فریم ورک تزریق وابستگی خارجی، از این مفهوم پشتیبانی می‌کند.

مروری مقدماتی بر تزریق وابستگی‌ها

امکان نوشتن آزمون واحد برای new OpenFileDialog وجود ندارد؟ اشکالی نداره، یک Interface بر اساس نیاز نهایی برنامه درست کنید (نیاز نهایی برنامه از این ماجرا فقط یک رشته LoadPath است و بس) سپس در ViewModel با این اینترفیس کار کنید؛ چون به این ترتیب امکان « تقلید » آن فراهم می‌شود.

یک مثال عملی:

ViewModel نیاز دارد تا مسیر فایلی را از کاربر بپرسد. این مساله را با کمک dependency injection در ادامه حل خواهیم کرد.
ابتدا سورس کامل این مثال:

ViewModel برنامه (تعریف شده در پوشه ViewModels برنامه):

```
namespace WpfFileDialogMvvm.ViewModels
{
    public interface IFilePathContract
    {
        string GetFilePath();
    }

    public class MainWindowViewModel
    {
        IFilePathContract _filePathContract;
        public MainWindowViewModel(IFilePathContract filePathContract)
        {
            _filePathContract = filePathContract;
        }

        //...

        private void load()
        {
            string loadFilePath = _filePathContract.GetFilePath();
            if (!string.IsNullOrWhiteSpace(loadFilePath))
            {
                // Do something
            }
        }
    }
}
```

دو نمونه از پیاده سازی اینترفیس IFilePathContract تعریف شده (در پوشه Dialogs برنامه):

```
using Microsoft.Win32;
using WpfFileDialogMvvm.ViewModels;

namespace WpfFileDialogMvvm.Dialogs
{
    public class OpenFileDialogProvider : IFilePathContract
    {
        public string GetFilePath()
        {
            var ofd = new OpenFileDialog
            {
                Filter = "XML files (*.xml)|*.xml"
            };
            string filePath = null;
            bool? dialogResult = ofd.ShowDialog();
            if (dialogResult.HasValue && dialogResult.Value)
            {
                filePath = ofd.FileName;
            }
            return filePath;
        }
    }

    public class FakeOpenFileDialogProvider : IFilePathContract
    {
        public string GetFilePath()
        {
            return @"c:\path\data.xml";
        }
    }
}
```

و View برنامه:

```
<Window x:Class="WpfFileDialogMvvm.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:vm="clr-namespace:WpfFileDialogMvvm.ViewModels"
        xmlns:dialogs="clr-namespace:WpfFileDialogMvvm.Dialogs"
        Title="MainWindow" Height="350" Width="525">
    <Window.Resources>
        <ObjectDataProvider x:Key="mainWindowViewModel"
                            ObjectType="{x:Type vm:MainWindowViewModel}">
            <ObjectDataProvider.ConstructorParameters>
                <dialogs:OpenFileDialogProvider/>
            </ObjectDataProvider.ConstructorParameters>
        </ObjectDataProvider>
    </Window.Resources>
    <Grid DataContext="{Binding Source={StaticResource mainWindowViewModel}}">

    </Grid>
</Window>
```

توضیحات:

ما در ViewModel نیاز داریم تا مسیر نهایی فایل را دریافت کنیم و این عملیات نیاز به فراخوانی متد ShowDialog ایی را دارد که امکان نوشتن آزمون واحد خودکار را از ViewModel ما سلب خواهد کرد. بنابراین بر اساس نیاز برنامه یک اینترفیس عمومی به نام IFilePathContract را طراحی می‌کنیم. در حالت کلی کلاسی که این اینترفیس را پیاده سازی می‌کند، قرار است مسیری را برگرداند. اما به کمک استفاده از اینترفیس، به صورت ضمنی اعلام می‌کنیم که «برای ما مهم نیست که چگونه». می‌خواهد OpenFileDialogProvider ذکر شده باشد، یا نمونه تقلیدی مانند FakeOpenFileDialogProvider. از نمونه واقعی

OpenFileDialogProvider در برنامه اصلی استفاده خواهیم کرد، از نمونه تقلیدی FakeOpenFileDialogProvider در آزمون واحد و نکته مهم هم اینجا است که ViewModel ما چون بر اساس اینترفیس IFilePathContract پیاده سازی شده، با هر دو DialogProvider یاد شده می‌تواند کار کند.

مرحله آخر نوبت به وهله سازی نمونه واقعی، در View برنامه است. یا می‌توان در Code behind مرتبط با View نوشت:

```
namespace WpfFileDialogMvvm
{
    public partial class MainWindow
    {
        public MainWindow()
        {
            InitializeComponent();
            this.DataContext = new MainWindowViewModel(new OpenFileDialogProvider());
        }
    }
}
```

و یا از روش ObjectDataProvider توکار WPF هم می‌شود استفاده کرد؛ که مثال آن‌را در کدهای XAML مرتبط با View ذکر شده می‌توانید مشاهده کنید. ابتدا دو فضای نام vm و dialog تعریف شده (با توجه به اینکه مثلا در این مثال، دو پوشه ViewModels و Dialogs وجود دارند). سپس کار تزریق وابستگی‌ها به سازنده کلاس MainWindowViewModel، از طریق ObjectDataProvider.ConstructorParameters انجام می‌شود:

```
<ObjectDataProvider x:Key="mainWindowViewModel"
                            ObjectType="{x:Type vm:MainWindowViewModel}">
            <ObjectDataProvider.ConstructorParameters>
                <dialogs:OpenFileDialogProvider/>
            </ObjectDataProvider.ConstructorParameters>
</ObjectDataProvider>
```

نظرات خوانندگان

نویسنده: Salar
تاریخ: ۱۳۹۰/۱۰/۱۲ ۱۹:۳۸:۵۰

خواستم تشکر کنم از شما که اندک ته مانده وب فارسی رو غنی می کنید. از سری MVVM هم تشکر می کنم.
نظرتون در مورد کتابچه 54 صفحه ای Advanced MVVM چی هست؟ می خواستم در اولین فرصت مطالعش کنم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۱۲ ۱۹:۵۲:۰۱

سلام؛ ممنون. شما لطف دارید.
این کتابچه در حقیقت شرح نوشتن یک برنامه بازی به کمک الگوی MVVM است. بد نیست ولی انتظار نداشته باشید که به صورت قدم به قدم چیزی را توضیح داده باشد. فقط شرح برنامه است.

نویسنده: مومن
تاریخ: ۱۳۹۰/۱۰/۱۴ ۰۰:۵۰:۴۷

سلام
از مطالب خوب مفیدتون تشکر می کنم.
سوال: راهی برای مدیریت دسترسی کاربران به Command ها وجود داره.
به این صورت که در فرم تعریف دسترسی ها لیست فرامین نمایش داده شود و سپس آنها را برای کاربران یا گروههای کاربری فعال و غیر فعال کنیم.
خلاصه کلام: راهی برای نمایش لیست کردن فرامین یک ویومدل وجود داره یانه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۱۴ ۰۱:۰۲:۰۹

- شما تمام خواص عمومی (و حتی غیرعمومی) رو می‌تونید به کمک Reflection لیست کنید. نوع آن‌ها هم که مشخص است از جنس مثلا ICommand یا DelegateCommand هستند. بنابراین یافتن و لیست کردن خودکار Commands تعریف شده در یک ViewModel به این ترتیب امکان پذیر است.
- در حالت کلی برای مدیریت Commands فقط کافی است قسمت canExecute آن‌ها را مدیریت کنید (مثلا در delegate command معرفی شده در همین سری). اگر false برگرداند (مثلا بر اساس سطح دسترسی کاربر جاری)، خودبخود عنصر مرتبط با آن غیرفعال خواهد شد.

نویسنده: مومن
تاریخ: ۱۳۹۰/۱۰/۱۴ ۰۷:۴۶:۵۱

سپاس

نویسنده: مهرناز توکلی
تاریخ: ۱۳۹۱/۰۷/۱۹ ۱۶:۶

خیلی خوب بود .ممنون