عنوان: **آموزش Prism #1**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۴/۰۱ ۸:۱۵
آدرس: www.dotnettips.info
گروه‌ها: MVVM, Silverlight, WPF, prism

امروزه تقریبا تمام کسانی که پروژه‌های WPF یا Silverlight رو توسعه می‌دهند با مدل برنامه نویسی MVVM آشنایی دارند. فریم ورک‌های مختلفی برای توسعه پروژه‌ها به صورت MVVM وجود دارد. نظیر:

MVVM Light

Prism

Caliburn

Cinch

WAF

Catel

Onyx

MVVM helpers

و...

هر کدوم از فریم ورک‌های بالا مزایا، معایب و طرفداران خاص خودشون رو دارند(^) ولی به جرات می‌تونیم Prism رو به عنوان قوی‌ترین فریم ورک برای پیاده سازی پروژهای بزرگ و قوی و ماژولار با تکنولوژی WPF یا Silverlight بنامیم. در این پست به معرفی و بررسی مفاهیم اولیه Prism خواهیم پرداخت و در پست‌های دیگر به پیاده سازی عملی همراه با مثال می‌پردازیم.

*اگر به هر دلیلی مایل به یادگیری و استفاده از Prism نیستید، بهتون پیشنهاد می‌کنم از WAF استفاده کنید.

پیش نیازها:

برای یادگیری PRISM ابتدا باید با مفاهیم زیر در WPF یا Silverlight آشنایی داشته باشید.(فرض بر این است که به UserControl و Xaml و Dependency Properties، **تسلط** کامل دارید)

Data binding

Resources

Commands

Behaviors

چرا Prism ؟

Prism به صورت کامل از Modular Programming برای پروژه‌های WPF و Silverlight پشتیانی می‌کند*

از Prism هم می‌توانیم در پروژه‌های WPF استفاده کنیم و هم Silverlight.

Prism به صورت کامل از الگوی MVVM برای پیاده سازی پروژه‌ها پشتیبانی می‌کند.

پیاده سازی مفاهیمی نظیر Composite Command و Command Behavior و Asynchronous Interacion به راحتی در Prism امکان پذیر است.

مفاهیم تزریق وابستگی به صورت توکار در Prism فراهم است که برای پیاده سازی این مفاهیم به طور پیش فرض امکان استفاده از UnityContainer و MEF در Prism تدارک دیده شده است.

پیاده سازی Region navigation در Prism به راحتی امکان پذیر است.

به وسیله امکان Event Aggregation به راحتی می‌توانیم بین ماژول‌های مختلف ارتباط برقرار کنیم.

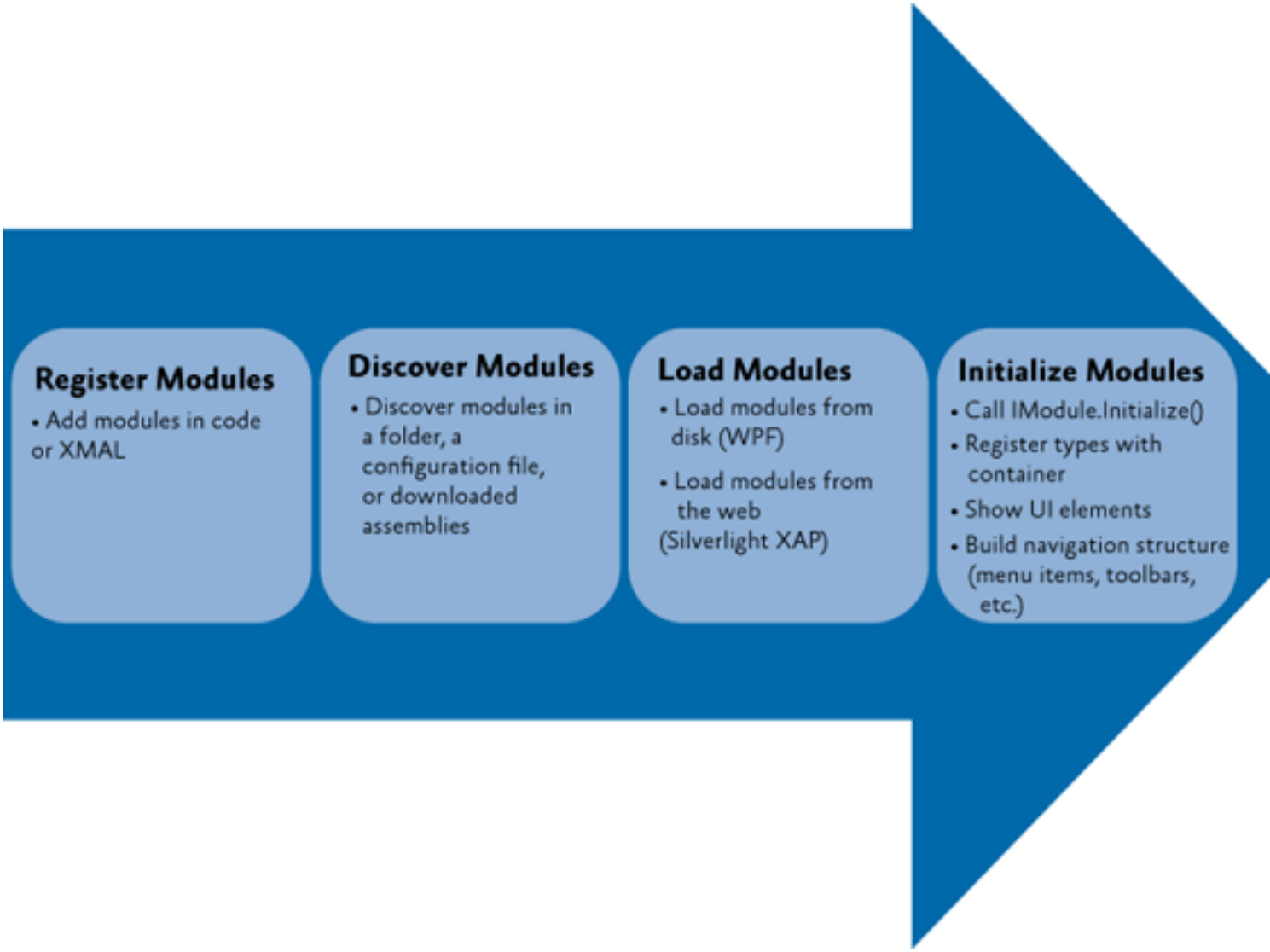
***توضیح درباره برنامه‌های ماژولار**

در تولید پروژهای نرم افزاری بزرگ هر چه قدر هم اگر در تهیه فایل‌های اسمبلی، کلاس ها، اینترفیس‌ها و کلا طراحی پروژه به صورت شی گرا دقت به خرج دهیم باز هم ممکن است پروژه به صورت یک پارچه طراحی نشود. یعنی بعد از اتمام پروژه، توسعه، تست پذیری و نگهداری آن سخت و در بعضی مواقع غیر ممکن خواهد شد. برنامه نویسی ماژولار این امکان را فراهم می‌کنه که یک پروژه با مقیاس برزگ به چند پروژه کوچک تقسیم شده و همه مراحل طراحی و توسعه و تست برای هر کدام از این ماژول‌ها به صورت جدا انجام شود.

Prism امکاناتی رو برای طراحی و توسعه این گونه پروژه‌ها به صورت ماژولار فراهم کرده است:

ابتدا باید نام و مکان هر ماژول رو به Prism معرفی کنیم که می‌تونیم اون‌ها رو در کد یا Xaml یا Configuration File تعریف کنیم.
با استفاده از Metadata باید وابستگی‌ها و مقادیر اولیه برای هر ماژول مشخص شود.
با کمک تزریق وابستگی‌ها ارتباطات بین ماژول‌ها میسر می‌شود.
ماژول مورد نظر به دو صورت OnDemand و Available لود خواهد شد.

در شکل زیر مراحل بالا قابل مشاهده است:

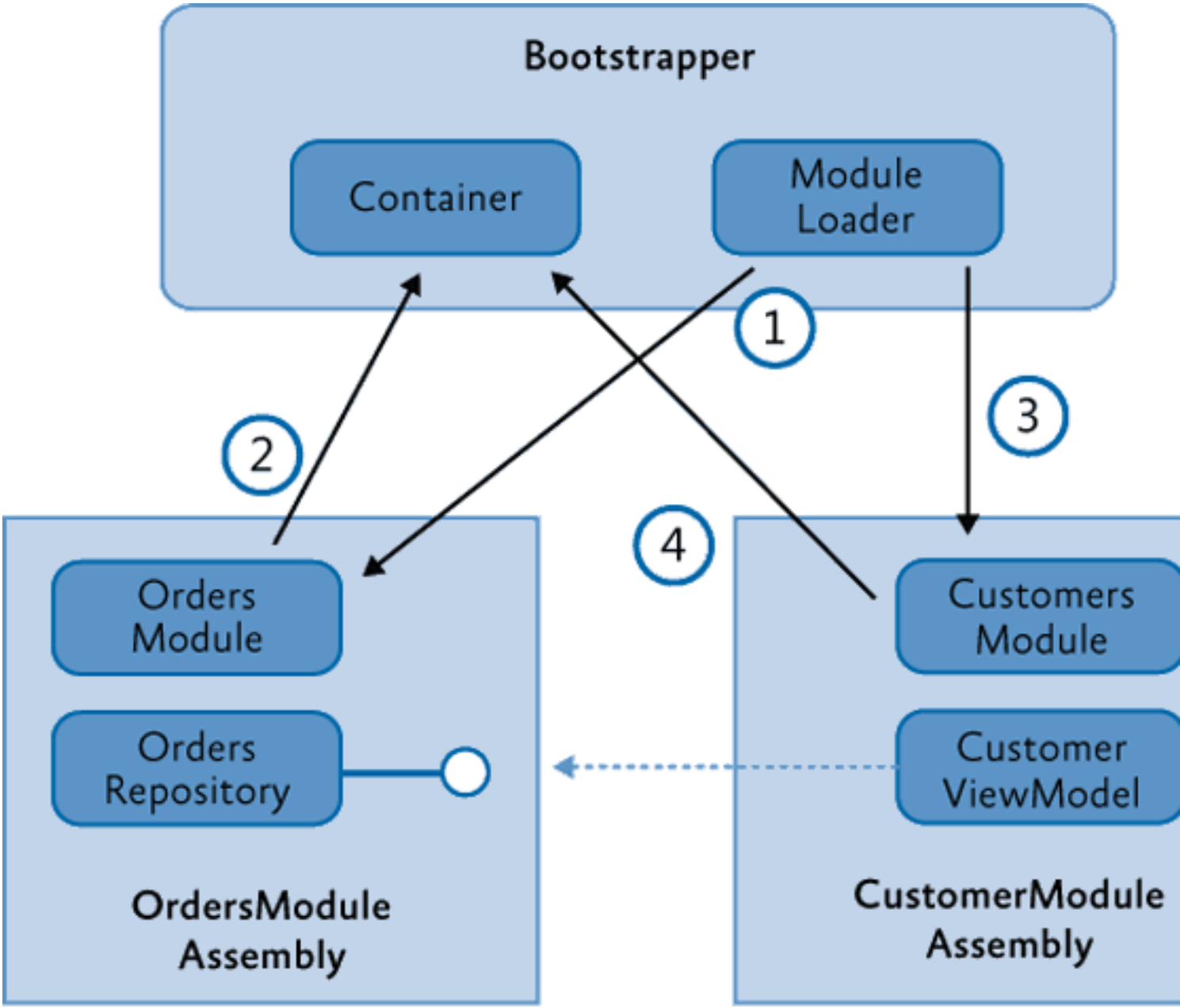


Bootstrapper چیست؟

در هر پروژه ماژولار (مختص Prism نیست) برای اینکه ماژول‌های مختلف یک پروژه، قابلیت استفاده به صورت یک پارچه رو در یک Application داشته باشند باید مفهومی به نام Bootstapper رو پیاده سازی کنیم که وظیفه اون شناسایی و پیکربندی و لود ماژول هاست. در Prism دو نوع Bootstrapper پیش فرض وجود دارد.

MefBootstrapper : کلاس پایه Bootstrapper که مبنای آن MEF است. اگر قصد استفاده از MEF رو در پروژه‌های خود دارید(^) Bootstrapper شما باید از این کلاس ارث ببرد.
UnityBootstrapper : کلاس پایه Bootstrapper که مبنای آن UnityContainer است. اگر قصد استفاده از UnityContainer یا Service Locator (^) رو در پروژه‌های خود دارید Bootstrapper شما باید از این کلاس ارث ببرد.

تصویری از ارتباط Bootstrapper با ماژول‌های سیستم



مفهوم Shell

در پروژه‌های WPF، در فایل App.xaml توسط یک Uri نقطه شروع پروژه را تعیین می‌کنیم. در پروژه‌های Silverlight به وسیله خاصیت RootVisual نقطه شروع سیستم تعیین می‌شود. در Prism نقطه شروع پروژه توسط bootsrapper تعیین می‌شود. دلیل این امر این است که Shell در پروژه‌های مبتنی بر Prism متکی بر Region Manager است. از Region برای لود و نمایش ماژول‌ها استفاده خواهیم کرد.

ادامه دارد...

نظرات خوانندگان

نویسنده: محمد احمدی
تاریخ: ۱۳۹۲/۰۴/۰۱ ۱۰:۶

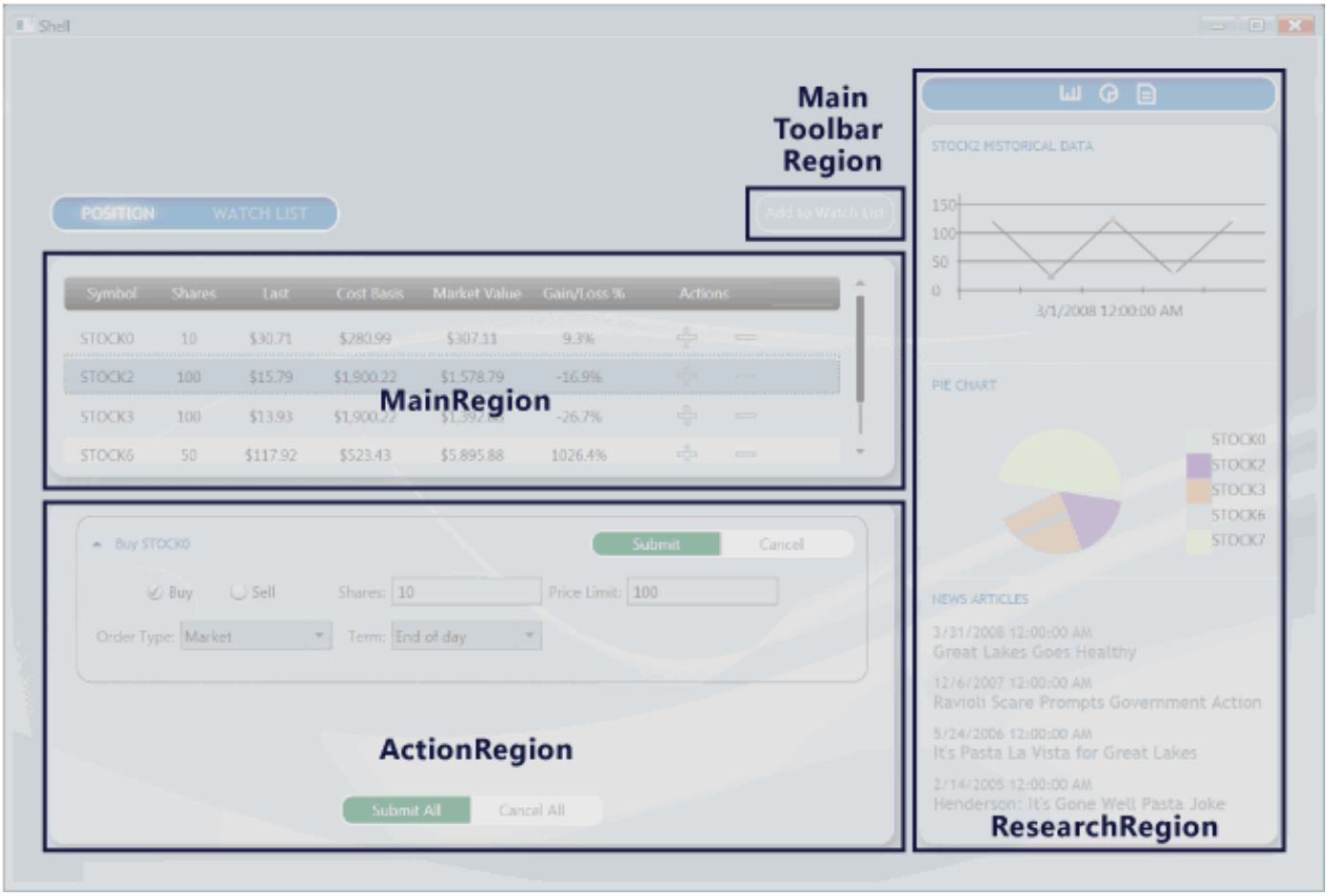
با سلام و تشکر از مطلب مفیدتون
همانطور که می‌دانید مدل‌های مختلف توسعه MVVM برای مقاصد مختلف بهتر است و به طور کلی نمی‌توان گفت که کدام بهتر است لطفا در ادامه مطلب این فریم ورک را با MVVM Light هم مقایسه بفرمائید تا موارد استفاده هر کدام بهتر مشخص شود

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۴/۰۱ ۱۱:۵۶

ممنون.
من از Prism به عنوان بهترین فریم ورک نام نبردم بلکه از عنوان قوی‌ترین فریم ورک استفاده کردم
"می‌تونیم Prism رو به عنوان قوی‌ترین فریم ورک برای پیاده سازی پروژهای بزرگ و قوی و ماژولار با تکنولوژی WPF یا Silverlight بنامیم." که لزوما به معنی بهترین نیست.
MVVM Light در حال حاضر به عنوان محبوب‌ترین فریم ورک برای MVVM است که این محبوبیت بیشتر به خاطر راحتی کار با اون هست.

MVVM Light نظیر Prism هم قابلیت استفاده در WPF را دارد و هم Silverlight (مزیت). MVVM Light راهکار مشخصی برای پیاده سازی پروژه‌های ماژولار ندارد(منظور Modular Composite Application است) در حالی که Prism برای تولید Modular Composite Applicationها طراحی شده است. برای اینکه بتونید، بعضی از قابلیت‌ها موجود در Prism را برای پروژه‌های ماژولار شبیه سازی کنید باید از ترکیب MEF و MVVM Light استفاده کنید.

Prism به شما این امکان رو می‌ده که حتی برای Popup Windowها هم Region طراحی کنید(مزیت). با Prism می‌تونید به راحتی برای یک Command رفتار تعریف کنید(به صورت توکار از Interactionها استفاده می‌کنه(مزیت)) برای این کار در MVVM Light شما باید از EventToCommandها استفاده کنید که اصلا قابل مقایسه به مباحث Composite Command و Command Behavior نیست.
معادل Messaging در MVVM Light در Prism شما EventAggregatorها رو در اختیار دارید.
Prism به صورت توکار از dependency Injection استفاده میکنه و دو فریم ورک هم به شما پیشنهاد میده یکی MEF و دیگری UnityContainer(مزیت).
Prism به صورت توکار از Composite UI هم پشتیبانی می‌کند. به تصویر زیر دقت کنید:



به راحتی می‌تونید با استفاده از RegionManager موجود در Prism نواحی هر صفحه رو تقسیم بندی کنید و هر ناحیه هم می‌تونه توسط یک ماژول لود شود. برای طراحی و مدیریت صفحات در MVVM Light باید خودتون دست به کار بشید.
یادگیری و استفاده از قابلیت‌های MVVM Light در حد دو یا سه روز زمان می‌برد در حالی که برای یادگیری قابلیت‌های Prism یک کتاب نوشته شده است(^)
*در پایان دوباره تاکید می‌کنم که اگر نیازی به تولید و توسعه پروژه به صورت ماژولار رو ندارید بهتره که اصلا به Prism فکر نکنید.

نویسنده: Petek
تاریخ: ۱۳۹۲/۰۴/۰۲ ۰:۳۶

با سلام
دوست عزیز ممنون میشم این مطلب جالب و مفید رو هر چه بیشتر و سریعتر ادامه بدید . با تشکر

نویسنده: محمد احمدی
تاریخ: ۱۳۹۲/۰۴/۰۲ ۱۳:۱۴

دوست عزیز
ممنونم از راهنمایی جامع و مفیدتون . امیدوارم هر چه زودتر مطالب بیشتری در این زمینه از شما یاد بگیریم

عنوان:	**آموزش Prism #2**
نویسنده:	مسعود پاکدل
تاریخ:	۱۳۹۲/۰۴/۰۳ ۹:۵
آدرس:	www.dotnettips.info
گروه‌ها:	MVVM, WPF, prism

در پست قبلی توضیح کلی درباره فریم ورک Prism داده شد. در این بخش قصد داریم آموزش‌های داده شده در پست قبلی را با هم در یک مثال مشاهده کنیم. در پروژه‌های ماژولار طراحی و ایجاد زیر ساخت قوی برای مدیریت ماژول‌ها بسیار مهم است. Prism فریم ورکی است که فقط چارچوب و قواعد اصول طراحی این گونه پروژه‌ها را در اختیار ما قرار می‌دهد. در پروژه‌های ماژولار هر ماژول باید در یک اسمبلی جدا قرار داشته باشد که ساختار پیاده سازی آن می‌تواند کاملا متفاوت با پیاده سازی سایر ماژول‌ها باشد.

برای شروع باید فایل‌های اسمبلی Prism رو دانلود کنید(لینک دانلود).

تشریح پروژه:

می‌خواهیم برنامه ای بنویسیم که دارای سه ماژول زیر است.:

ماژول Navigator : برای انتخاب و Switch کردن بین ماژول‌ها استفاده می‌شود؛

ماژول طبقه بندی کتاب‌ها : لیست طبقه بندی کتاب‌ها را به ما نمایش می‌دهد؛

ماژول لیست کتاب‌ها : عناوین کتاب‌ها به همراه نویسنده و کد کتاب را به ما نمایش می‌دهد.

*در این پروژه از UnityContainer برای مباحث Dependency Injection استفاده شده است.

ابتدا یک پروژه WPF در Vs.Net ایجاد کنید(در اینجا من نام آن را FirstPrismSample گذاشتم). قصد داریم یک صفحه طراحی کنیم که دو ماژول مختلف در آن لود شود. ابتدا باید Shell پروژه رو طراحی کنیم. یک Window جدید به نام Shell بسازید و کد زیر را

در آن کپی کنید.

```
<Window x:Class="FirstPrismSample.Shell"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:com="http://www.codeplex.com/CompositeWPF"
    Title="Prism Sample By Masoud Pakdel" Height="400" Width="600"
WindowStartupLocation="CenterScreen">
    <DockPanel>
      <ContentControl com:RegionManager.RegionName="WorkspaceRegion" Width="400"/>
      <ContentControl com:RegionManager.RegionName="NavigatorRegion"  DockPanel.Dock="Left" Width="200"
/>
    </DockPanel>
</Window>
```

در این صفحه دو ContentControl تعریف کردم یکی به نام Navigator و دیگری به نام Workspace. به وسیله RegionName که یک AttachedProperty است هر کدوم از این نواحی را برای Prism تعریف کردیم. حال باید یک ماژول برای Navigator و دو ماژول دیگر یکی برای طبقه بندی کتاب‌ها و دیگری برای لیست کتاب‌ها بسازیم.

#پروژه Common

قبل از هر چیز یک پروژه Common می‌سازیم و مشترکات بین ماژول‌ها رو در آن قرار می‌دهیم(این پروژه باید به تمام ماژول‌ها رفرنس داده شود). این مشترکات شامل :

کلاس پایه ViewModel

کلاس ViewRequestEvent

کلاس ModuleService

کد کلاس ViewModelBase که فقط اینترفیس INotifyPropertyChanged رو پیاده سازی کرده است:

```
using System.ComponentModel;

namespace FirstPrismSample.Common
{
    public abstract class ViewModelBase : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        protected void RaisePropertyChangedEvent( string propertyName )
        {
            if ( PropertyChanged != null )
            {
                PropertyChangedEventArgs e = new PropertyChangedEventArgs( propertyName );
                PropertyChanged( this, e );
            }
        }
    }
}
```

کلاس ViewRequestEvent که به صورت زیر است:

```
using Microsoft.Practices.Composite.Presentation.Events;

namespace FirstPrismSample.Common.Events
{
    public class ViewRequestedEvent : CompositePresentationEvent<string>
    {
    }
}
```

توضیح درباره CompositePresentationEvent :

در طراحی و توسعه پروژه‌های ماژولار نکته ای که باید به آن دقت کنید این است که ماژول‌های پروژه نباید به هم وابستگی مستقیم داشته باشند در عین حال ماژول‌ها باید بتوانند با هم در ارتباط باشند. CPE یا **C**omposite **P**resentation **E**vent دقیقا برای این

منظور به وجود آمده است. CPE که در این جا طراحی کردم فقط کلاسی است که از CompositePresentationEventارث برده است و دلیل آن که به صورت string generic استفاده شده است این است که می‌خواهیم در هر درخواست نام ماژول درخواستی را داشته باشیم و به همین دلیل نام آن را ViewRequestedEvent گذاشتم.

توضیح درباره EventAggregator

EventAggregator یا به اختصار EA مکانیزمی است در پروژهای ماژولار برای اینکه در Composite UIها بتوانیم بین کامپوننت‌ها ارتباط برقرار کنیم. استفاده از EA وابستگی بین ماژول‌ها را از بین خواهد برد. برنامه نویسانی که با MVVM Light آشنایی دارند از قابلیت Messaging موجود در این فریم ورک برای ارتباط بین View و ViewModel استفاده می‌کنند. در Prism این عملیات توسط EA انجام می‌شود. یعنی برای ارتباط با Viewها باید از EA تعبیه شده در Prism استفاده کنیم. در ادامه مطلب، چگونگی استفاده از EA را خواهید آموخت.

اینترفیس IModuleService که فقط شامل یک متد است:

```
namespace FirstPrismSample .Common
{
    public interface IModuleServices
    {
        void ActivateView(string viewName);
    }
}
```

کلاس ModuleService که اینترفیس بالا را پیاده سازی کرده است:

```
using Microsoft.Practices.Composite.Regions;
using Microsoft.Practices.Unity;

namespace FirstPrismSample.Common
{
    public class ModuleServices : IModuleServices
    {
        private readonly IUnityContainer m_Container;

        public ModuleServices(IUnityContainer container)
        {
            m_Container = container;
        }

        public void ActivateView(string viewName)
        {
            var regionManager = m_Container.Resolve<IRegionManager>();

            // غیر فعال کردن ویو
            IRegion workspaceRegion = regionManager.Regions["WorkspaceRegion"];
            var views = workspaceRegion.Views;
            foreach (var view in views)
            {
                workspaceRegion.Deactivate(view);
            }

            //فعال کردن ویو انتخاب شده
            var viewToActivate = regionManager.Regions["WorkspaceRegion"].GetView(viewName);
            regionManager.Regions["WorkspaceRegion"].Activate(viewToActivate);
        }
    }
}
```

متد ActivateView نام view مورد نظر برای فعال سازی را دریافت می‌کند. برای فعال کردن View ابتدا باید سایر viewهای فعال در RegionManager را غیر فعال کنیم. سپس فقط view مورد نظر در RegionManager انتخاب و فعال می‌شود.

*نکته: در هر ماژول ارجاع به اسمبلی‌های Prism مورد نیاز است.

#ماژول طبقه بندی کتاب ها:

برای شروع یک Class Library جدید به نام ModuleCategory به پروژه اضافه کنید. یک UserControl به نام CategoryView

بسازید و کدهای زیر را در آن کپی کنید.

```
<UserControl x:Class="FirstPrismSample.ModuleCategory.CategoryView "
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             Background="LightGray" FlowDirection="RightToLeft" FontFamily="Tahoma">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <TextBlock Text=" طبقه بندی ها"/>
        <ListView Grid.Row="1"  Margin="10" Name="lvCategory">
            <ListView.View>
                <GridView>
                    <GridViewColumn Header="کد" Width="50" />
                    <GridViewColumn Header="عنوان" Width="200"  />
                </GridView>
            </ListView.View>
        </ListView>
    </Grid>
</UserControl>
```

یک کلاس به نام CategoryModule بسازید که اینترفیس IModule رو پیاده سازی کند.

```
using Microsoft.Practices.Composite.Events;
using Microsoft.Practices.Composite.Modularity;
using Microsoft.Practices.Composite.Regions;
using Microsoft.Practices.Unity;
using FirstPrismSample.Common;
using FirstPrismSample.Common.Events;
using Microsoft.Practices.Composite.Presentation.Events;

namespace FirstPrismSample.ModuleCategory
{
    [Module(ModuleName = "ModuleCategory")]
    public class CategoryModule : IModule
    {
        private readonly IUnityContainer m_Container;
        private readonly string moduleName = "ModuleCategory";

        public CategoryModule(IUnityContainer container)
        {
            m_Container = container;
        }

        ~CategoryModule()
        {
            var eventAggregator = m_Container.Resolve<IEventAggregator>();
            var viewRequestedEvent = eventAggregator.GetEvent<ViewRequestedEvent>();
            viewRequestedEvent.Unsubscribe(ViewRequestedEventHandler);
        }

        public void Initialize()
        {
            var regionManager = m_Container.Resolve<IRegionManager>();
            regionManager.Regions["WorkspaceRegion"].Add(new CategoryView(), moduleName);

            var eventAggregator = m_Container.Resolve<IEventAggregator>();
            var viewRequestedEvent = eventAggregator.GetEvent<ViewRequestedEvent>();
            viewRequestedEvent.Subscribe(this.ViewRequestedEventHandler, true);
        }

        public void ViewRequestedEventHandler(string moduleName)
        {
            if (this.moduleName != moduleName) return;

            var moduleServices = m_Container.Resolve<IModuleServices>();
            moduleServices.ActivateView(moduleName);
        }
    }
}
```

چند نکته :

*ModuleAttribute استفاده شده در بالای کلاس برای تعیین نام ماژول استفاده می‌شود. این Attribute دارای دو خاصیت دیگر

هم است :

OnDemand : برای تعیین اینکه ماژول باید به صورت OnDemand (بنا به درخواست) لود شود.
StartupLoaded : برای تعیین اینکه ماژول به عنوان ماژول اول پروژه لود شود.(البته این گزینه Obsolute شده است)

*برای تعریف ماژول کلاس مورد نظر حتما باید اینترفیس IModule را پیاده سازی کند. این اینترفیس فقط شامل یک متد است به نام Initialize.

*در این پروژه چون Viewهای برنامه صرفا جهت نمایش هستند در نتیجه نیاز به ایجاد ViewModel برای آن‌ها نیست. در پروژه‌های اجرایی حتما برای هر View باید ViewModel متناظر با آن تهیه شود.

توضیح درباره متد Initialize

در این متد ابتدا با استفاده از Container موجود RegionManager را به دست می‌آوریم. با استفاده از RegionManager می‌تونیم یک CompositeUI طراحی کنیم. در فایل Shell مشاهده کردید که یک صفحه به دو ناحیه تقسیم شد و به هر ناحیه هم یک نام اختصاص دادیم. دستور زیر به یک ناحیه اشاره خواهد داشت:

```
regionManager.Regions["WorkspaceRegion"]
```

در خط بعد با استفاده از EA یا Event Aggregator توانستیم CPE را بدست بیاوریم. متد Subscribe در کلاس CPE یک ارجاع قوی به delegate مورد نظر ایجاد می‌کند(پارامتر دوم این متد که از نوع boolean است) که به این معنی است که این delegate هیچ گاه توسط GC جمع آوری نخواهد شد. در نتیجه، قبل از اینکه ماژول بسته شود باید به صورت دستی این کار را انجام دهیم که مخرب را برای همین ایجاد کردیم. اگر به کدهای مخرب دقت کنید می‌بینید که با استفاده از EA توانستیم ViewRequestEventHandler را Unsubscribe کنیم به دلیل اینکه از ارجاع قوی با strong Reference در متد Subscribe استفاده شده است. دستور moduleService.ActiveateView ماژول مورد نظر را در region مورد نظر هاست خواهد کرد.

#ماژول لیست کتاب ها:

ابتدا یک Class Library به نام ModuleBook بسازید و همانند ماژول قبلی نیاز به یک Window و یک کلاس داریم:
BookWindow که کاملا مشابه به CategoryView است.

```
<UserControl x:Class="FirstPrismSample.ModuleBook.BookView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Background="LightGray" FontFamily="Tahoma" FlowDirection="RightToLeft">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <TextBlock Text="لیست کتاب ها"/>
        <ListView Grid.Row="1" Margin="10" Name="lvBook">
            <ListView.View>
                <GridView>
                    <GridViewColumn Header="کد" Width="50"  />
                    <GridViewColumn Header="عنوان" Width="200" />
                    <GridViewColumn Header="نویسنده" Width="150" />
                </GridView>
            </ListView.View>
        </ListView>
    </Grid>
</UserControl>
```

کلاس BookModule که پیاده سازی و توضیحات آن کاملا مشابه به CategoryModule می‌باشد.

```
using Microsoft.Practices.Composite.Events;
using Microsoft.Practices.Composite.Modularity;
using Microsoft.Practices.Composite.Presentation.Events;
using Microsoft.Practices.Composite.Regions;
using Microsoft.Practices.Unity;
using FirstPrismSample.Common;
using FirstPrismSample.Common.Events;

namespace FirstPrismSample.ModuleBook
{
    [Module(ModuleName = "moduleBook")]
    public class BookModule : IModule
    {
        private readonly IUnityContainer m_Container;
        private readonly string moduleName = "ModuleBook";

        public BookModule(IUnityContainer container)
        {
            m_Container = container;
        }

        ~BookModule()
        {
            var eventAggregator = m_Container.Resolve<IEventAggregator>();
            var viewRequestedEvent = eventAggregator.GetEvent<ViewRequestedEvent>();

            viewRequestedEvent.Unsubscribe(ViewRequestedEventHandler);
        }

        public void Initialize()
        {
            var regionManager = m_Container.Resolve<IRegionManager>();
            var view = new BookView();
            regionManager.Regions["WorkspaceRegion"].Add(view, moduleName);
            regionManager.Regions["WorkspaceRegion"].Deactivate(view);

            var eventAggregator = m_Container.Resolve<IEventAggregator>();
            var viewRequestedEvent = eventAggregator.GetEvent<ViewRequestedEvent>();
            viewRequestedEvent.Subscribe(this.ViewRequestedEventHandler, true);
        }

        public void ViewRequestedEventHandler(string moduleName)
        {
            if (this.moduleName != moduleName) return;

            var moduleServices = m_Container.Resolve<IModuleServices>();
            moduleServices.ActivateView(m_WorkspaceBName);
        }
    }
}
```

#ماژول Navigator

برای این ماژول هم ابتدا View مورد نظر را ایجاد می‌کنیم:

```
<UserControl x:Class="FirstPrismSample.ModuleNavigator.NavigatorView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
    <Grid>
        <StackPanel VerticalAlignment="Center">
            <TextBlock Text="انتخاب ماژول" Foreground="Green" HorizontalAlignment="Center"
                VerticalAlignment="Center" FontFamily="Tahoma" FontSize="24" FontWeight="Bold" />
        <Button Command="{Binding ShowModuleCategory}" Margin="5" Width="125">طبقه بندی کتاب
ها</Button>
        <Button Command="{Binding ShowModuleBook}" Margin="5" Width="125">لیست کتاب ها</Button>
        </StackPanel>
        </Grid>
</UserControl>
```

حال قصد داریم برای این View یک ViewModel بسازیم. نام آن را INavigatorViewModel خواهیم گذاشت:

```
public interface INavigatorViewModel
    {
        ICommand ShowModuleCategory { get; set; }
        ICommand ShowModuleBook { get; set; }
```

```
        string ActiveWorkspace { get; set; }

        IUnityContainer Container { get; set; }

        event PropertyChangedEventHandler PropertyChanged;
    }
```

*در اینترفیس بالا دو Command داریم که هر کدام وظیفه لود یک ماژول را بر عهده دارند.
*خاصیت ActiveWorkspace برای تعیین workspace فعال تعریف شده است.

حال به پیاده سازی مثال بالا می‌پردازیم:

```
public class NavigatorViewModel : ViewModelBase, INavigatorViewModel
    {
        public NavigatorViewModel(IUnityContainer container)
        {
            this.Initialize(container);
        }

        public ICommand ShowModuleCategory { get; set; }

        public ICommand ShowModuleBook { get; set; }

        public string ActiveWorkspace { get; set; }

        public IUnityContainer Container { get; set; }

        private void Initialize(IUnityContainer container)
        {
            this.Container = container;
            this.ShowModuleCategory = new ShowModuleCategoryCommand(this);
            this.ShowModuleBook = new ShowModuleBookCommand(this);
            this.ActiveWorkspace = "ModuleCategory";
        }
    }
```

تنها نکته مهم در کلاس بالا متد Initialize است که دو Command مورد نظر را پیاده سازی کرده است. ماژول پیش فرض هم ماژول طبقه بندی کتاب‌ها یا ModuleCategory در نظر گرفته شده است. همان طور که می‌بینید پیاده سازی Commandها بالا توسط دو کلاس ShowModuleCategoryCommand و ShowModuleBookCommand انجام شده که در زیر کدهای آن‌ها را می‌بینید.

#کد کلاس ShowModuleCategoryCommand

```
public class ShowModuleCategoryCommand : ICommand
    {
        private readonly NavigatorViewModel viewModel;
        private const string workspaceName = "ModuleCategory";

        public ShowModuleCategoryCommand(NavigatorViewModel viewModel)
        {
            this.viewModel = viewModel;
        }

        public bool CanExecute(object parameter)
        {
            return viewModel.ActiveWorkspace != workspaceName;
        }

        public event EventHandler CanExecuteChanged
        {
            add { CommandManager.RequerySuggested += value; }
            remove { CommandManager.RequerySuggested -= value; }
        }

        public void Execute(object parameter)
        {
            CommandServices.ShowWorkspace(workspaceName, viewModel);
        }
    }
```

#کد کلاس ShowModuleBookCommand

```
public class ShowModuleBookCommand : ICommand
    {
        private readonly NavigatorViewModel viewModel;
        private readonly string workspaceName = "ModuleBook";

        public ShowModuleBookCommand( NavigatorViewModel viewModel )
        {
            this.viewModel = viewModel;
        }

        public bool CanExecute( object parameter )
        {
            return viewModel.ActiveWorkspace != workspaceName;
        }

        public event EventHandler CanExecuteChanged
        {
            add { CommandManager.RequerySuggested += value; }
            remove { CommandManager.RequerySuggested -= value; }
        }

        public void Execute( object parameter )
        {
            CommandServices.ShowWorkspace( workspaceName , viewModel );
        }
    }
```

با توجه به این که فرض است با متدهای Execute و CanExecute و CanExecuteChanged آشنایی دارید از توضیح این مطالب خودداری خواهم کرد. فقط کلاس CommandServices در متد Execute دارای متدی به نام ShowWorkspace است که کدهای زیر را شامل می‌شود:

```
public static void ShowWorkspace(string workspaceName, INavigatorViewModel viewModel)
  {
            var eventAggregator = viewModel.Container.Resolve<IEventAggregator>();
            var viewRequestedEvent = eventAggregator.GetEvent<ViewRequestedEvent>();
            viewRequestedEvent.Publish(workspaceName);

            viewModel.ActiveWorkspace = workspaceName;
 }
```

در این متد با استفاده از CPE که در پروژه Common ایجاد کردیم ماژول مورد نظر را لود خواهیم کرد. و بعد از آن مقدار ActiveWorkspace جاری در ViewModel به نام ماژول تغییر پیدا می‌کند. متد Publish در CPE این کار را انجام خواهد دارد.

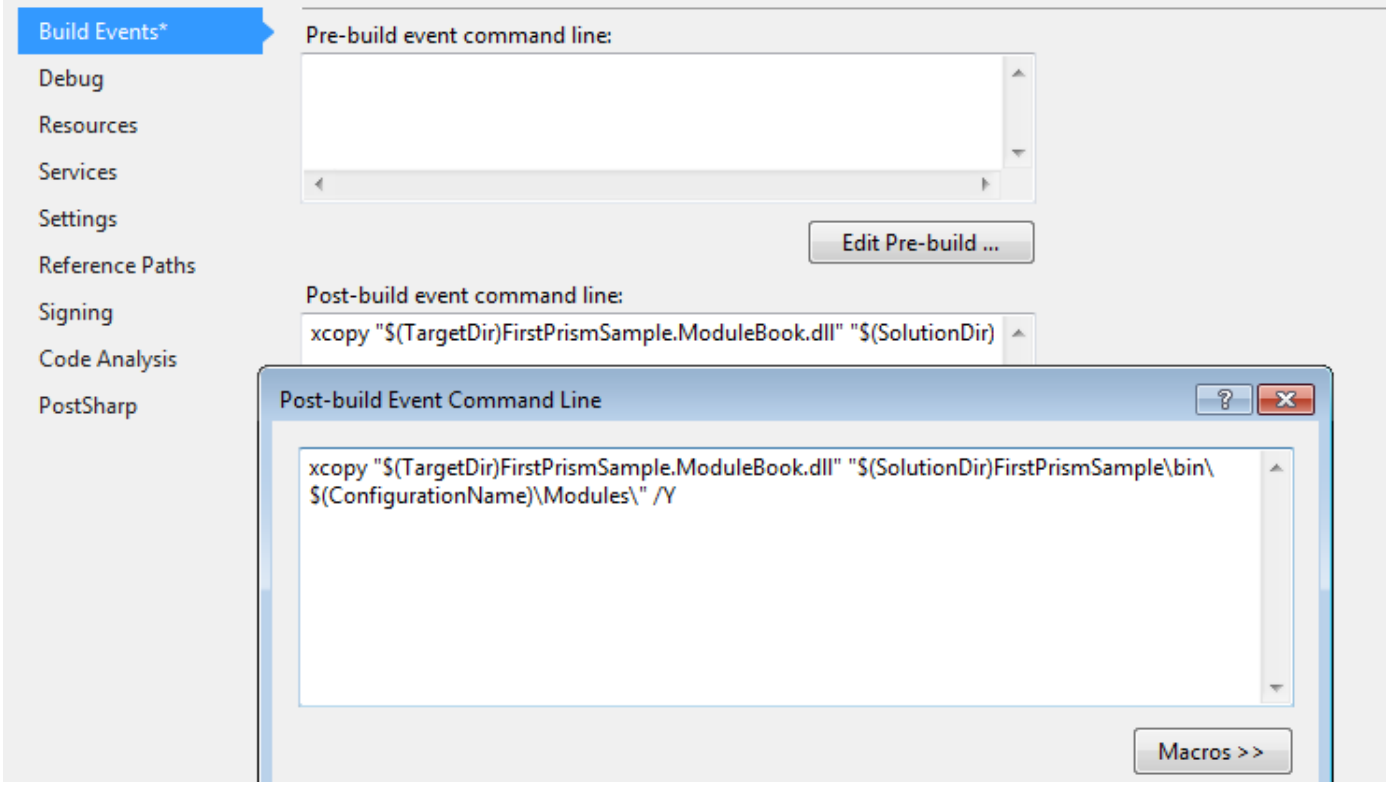
عدم وابستگی ماژول ها

همان طور که می‌بینید ماژول‌های پروژه به هم Reference داده نشده اند حتی هیچ Reference هم به پروژه اصلی یعنی جایی که فایل App.xaml قرار دارد، داده نشده است ولی در عین حال باید با هم در ارتباط باشند. برای حل این مسئله این ماژول‌ها باید در فولدر bin پروژه اصلی خود را کپی کنند. بهترین روش استفاده از Pre-Post Build Event خود VS.Net است. برای این کار از پنجره Project Properties وارد برگه Build Events شوید و از قسمت Post Build Event Command Line استفاده کنید و کد زیر را در آن کپی نمایید:

```
xcopy "$(TargetDir)FirstPrismSample.ModuleBook.dll"
"$(SolutionDir)FirstPrismSample\bin\$(ConfigurationName)\Modules\" /Y
```

قطعا باید به جای FirstPrismSample نام Solution خود و به جای ModuleBook نام ماژول را وارد نمایید.

مانند:



مراحل بالا برای هر ماژول باید تکرار شود(ModuleNavigation , ModuleBook , ModuleCategory). بعد از Rebuild پروژه در فولدر bin پروژه اصلی یک فولدر به نام Module ایجاد می‌شود که اسمبلی هر ماژول در آن کپی خواهد شد.

ایجاد Bootstrapper

حال نوبت به Bootstrapper میرسد(در پست قبلی در باره مفهوم Bootstrapper شرح داده شد). در پروژه اصلی یعنی جایی که فایل App.xaml قرار دارد کلاس زیر را ایجاد کنید.

```
public class Bootstrapper : UnityBootstrapper
    {
        protected override void ConfigureContainer()
        {
            base.ConfigureContainer();
            Container.RegisterType<IModuleServices, ModuleServices>();
        }

        protected override DependencyObject CreateShell()
        {
            var shell = new Shell();
            shell.Show();
            return shell;
        }

        protected override IModuleCatalog GetModuleCatalog()
        {
            var catalog = new DirectoryModuleCatalog();
            catalog.ModulePath = @".\Modules";
            return catalog;
        }
    }
```

متد ConfigureContainer برای تزریق وابستگی به وسیله UnityContainer استفاده می‌شود. در این متد باید تمامی Registrationهای مورد نیاز برای DI را انجام دهید. نکته مهم این است که عملیات وهله سازی و Initialization برای Container در متد base کلاس UnityBootstrapper انجام خواهد شد پس همیشه باید متد base این کلاس در ابتدای این متد فراخوانی شود در غیر این صورت با خطا متوقف خواهید شد.

متد CreateShell برای ایجاد و وهله سازی از Shell پروژه استفاده می‌شود. در این جا یک وهله از Shell Window برگشت داده می‌شود.
متد GetModuleCatalog برای تعیین مسیر ماژول‌ها در پروژه کاربرد دارد. در این متد با استفاده از خاصیت ModulePath کلاس DirectoryModuleCatalog تعیین کرده ایم که ماژول‌های پروژه در فولدر Modules موجود در bin اصلی پروژه قرار دارد. اگر به دستورات کپی در Post Build Event قسمت قبل توجه کنید می‌بینید که دستور ساخت فولدر وجود دارد.

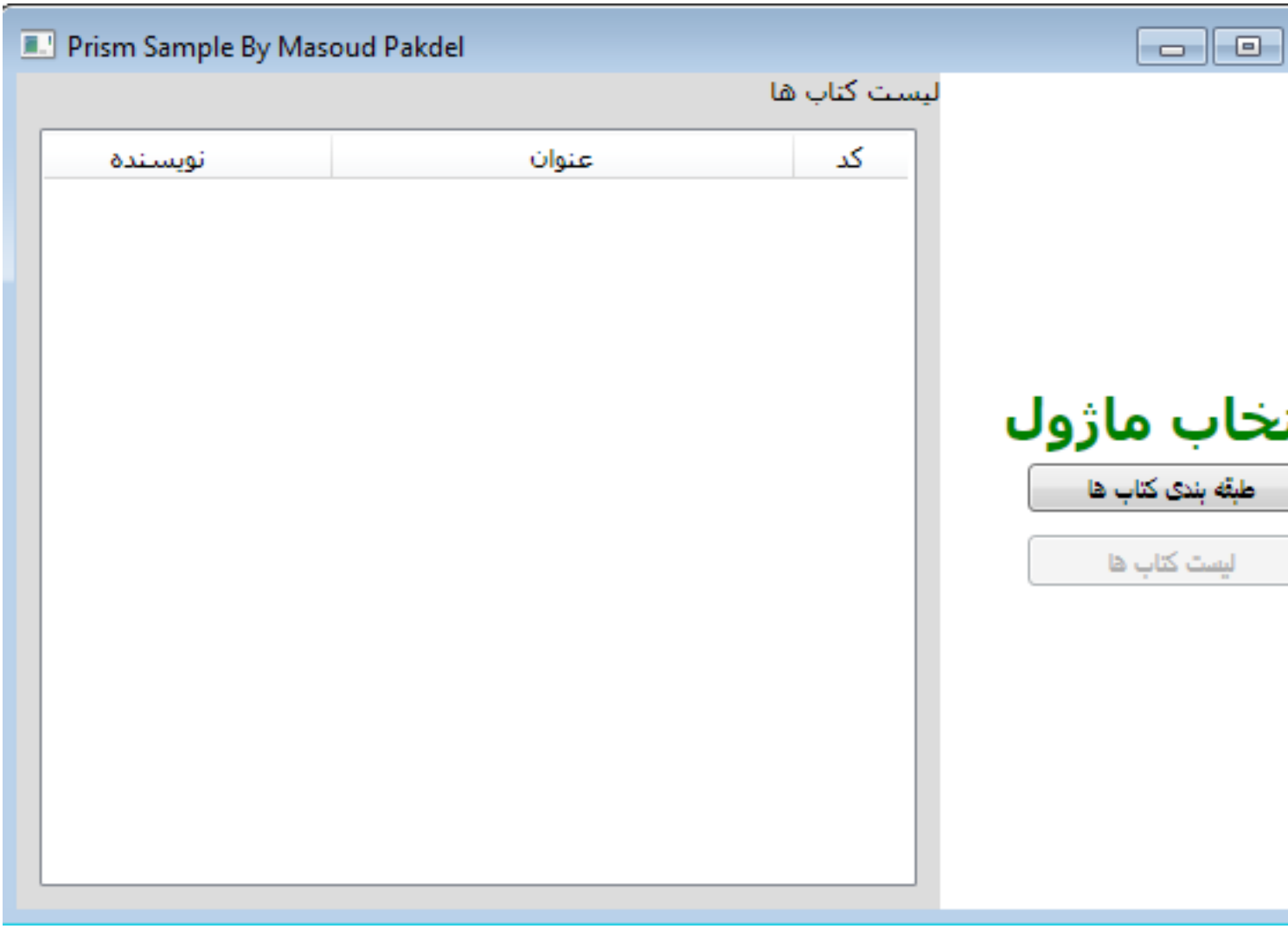
```
"$(SolutionDir)FirstPrismSample\bin\$(ConfigurationName)\Modules\" /Y
```

***نکته:** اگر استفاده از این روش برای شناسایی ماژول‌ها توسط Bootstrapper را چندان جالب نمی‌دانید می‌تونید از MEF استفاده کنید که اسمبلی ماژول‌های پروژه را به راحتی شناسایی می‌کند و در اختیار Bootsrtapper قرار می‌دهد(از آن جا در مستندات مربوط به Prism، بیشتر به استفاده از MEF تاکید شده است من هم در پست‌های بعدی، مثال‌ها را با MEF پیاده سازی خواهم کرد)

در پایان باید فایل App.xaml را تغییر دهید به گونه ای که متد Run در کلاس Bootstapper ابتدا اجرا شود.

```
public partial class App : Application
    {
        protected override void OnStartup(StartupEventArgs e)
        {
            base.OnStartup(e);
            var bootstrapper = new Bootstrapper();
            bootstrapper.Run();
        }
    }
```

اجرای پروژه:
بعد از اجرا، با انتخاب ماژول مورد نظر اطلاعات ماژول در Workspace Content Control لود خواهد شد.


Prism Sample By Masoud Pakdel
لیست کتاب ها
کد
عنوان
نویسنده
طبقه بندی کتاب ها
لیست کتاب ها

ادامه دارد...

نظرات خوانندگان

نویسنده: Petek
تاریخ: ۱۳۹۲/۰۴/۰۳ ۱۰:۲۷

با سلام مهندس
خیلی عالیه امیدوارم ادامه بدید . با تشکر

نویسنده: مهدی
تاریخ: ۱۳۹۲/۰۴/۰۳ ۱۹:۵۶

ممنون از آموزش خوبتون ، نظرتون در مورد استفاده از Prism به همراه StructerMap چیه ؟

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۴/۰۳ ۲۲:۲۳

شدنی است. فقط همانند UnityBootstrapper نیاز به یک StructureMapBootstrapper دارید. این کار قبلا توسط Richard Cerirol انجام شده. می‌تونید از nuget استفاده کنید:

```
PM> Install-Package Prism.StructureMapExtensions
```

نویسنده: بهنام
تاریخ: ۱۳۹۲/۰۴/۰۵ ۱:۲۶

با سلام و با تشکر مطلب مفیدتان
چند اصلاح کوچک در مطلب هست که اینجا بیان می‌کنم
بخش اول (مبدا)دستور xcopy باید به دستور زیر تبدیل شود:

```
xcopy "$(SolutionDir)\PrismProject.ModuleBook\bin\$(ConfigurationName)\PrismProject.ModuleBook.dll"
"$(SolutionDir)PrismProject\bin\$(ConfigurationName)\Modules\" /Y
```

همچنین متد GetModuleCatalog به CreateModuleCatalog تبدیل شده است.
با تشکر مجدد

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۴/۰۵ ۹:۳۰

ممنونم دوست عزیز.
در مورد دستور اول روش ذکر شده کاملا صحیح است و نیازی به اصلاح نیست.

$TargetDir دقیقا به مسیر فایل‌های اجرایی اشاره می‌کند و $ConfigurationName را در خودش پشتیبانی می‌کنه. یعنی اگر پروژه در حال Relase باشد با استفاده از $TargetDir دقیقا به فایل‌های موجود در فولدر Release در bin پروژه اشاره می‌کند و در حالت Debug به فایل‌های موجود در فولدر Debug در bin پروژه. با استفاده از گزینه Macros درقسمت Edit Post-Build مشاهده می‌کنید که مقدار $TargetDir دقیقا صحیح است. اما دلیل اینکه چرا در بخش دوم دستور از $SolutionDir استفاده شده است به این دلیل است که می‌خواهیم به فولدر bin پروژه اصلی اشاره داشته باشیم و چون این پروژه حتما در مسیر Solution جاری خواهد بود در نتیجه از این آدرس استفاده شده است.(در این جا TargetDir و TargetPath نمی‌تواند کمکی به ما بکند). به تصویر زیر دقت کنید:(چون پروژه در حالت release است در نتیجه مقادیر TargetDir و TargetPath به release ختم می‌شود)

Macro	Value
OutDir	bin\Release\
ConfigurationName	Release
ProjectName	XLIFFProject
TargetName	WpfApplication١
TargetPath	E:\Workspace\Projects\XLIFFProject\XLIFFProject\bin\Release\WpfApplication١.exe
ProjectPath	E:\Workspace\Projects\XLIFFProject\XLIFFProject\XLIFFProject.csproj
ProjectFileName	XLIFFProject.csproj
TargetExt	.exe
TargetFileName	WpfApplication١.exe
DevEnvDir	C:\Program Files (x٨٦)\Microsoft Visual Studio ١٠,٠\Common٧\IDE\
TargetDir	E:\Workspace\Projects\XLIFFProject\XLIFFProject\bin\Release\
ProjectDir	E:\Workspace\Projects\XLIFFProject\XLIFFProject\
SolutionFileName	XLIFFProject.sln
SolutionPath	E:\Workspace\Projects\XLIFFProject\XLIFFProject.sln
SolutionDir	E:\Workspace\Projects\XLIFFProject\
SolutionName	XLIFFProject
PlatformName	x٨٦
ProjectExt	.csproj
SolutionExt	.sln

به تفاوت مقادیر بین $TargetDir و $TargetPath و $SolutionDir و ... دقت کنید.

در مورد متد GetModuleCatalog هم باید عنوان کنم که این متد در اسمبلی Microsoft.Practices.Composite.UnityExtensions ورژن 2.0.1.0 وجود دارد. در ورژن 4 نسخه Prism این متد به این نام تغییر کرده است. در این جا می‌تونید تغییرات بین Prism Library 2 و Prism Library 4 رو ببینید

نویسنده: یوسف
تاریخ: ۱۳۹۲/۰۴/۲۲ ۱۹:۴۹

درود؛

لطفاً سورس پروژهٔ مثال را هم جهت دانلود اینجا بذارین، چون توی مقاله اشاره‌ای به اینکه پروژه‌ها از چه نوعی باشند و کدوم رفرنس‌ها را لازم دارند نشده و برای یکی مثل من که کلاً آشناییش با مقالات شما آغاز شده پیشرفت کار خیلی کند میشه.
سپاسگزارم.

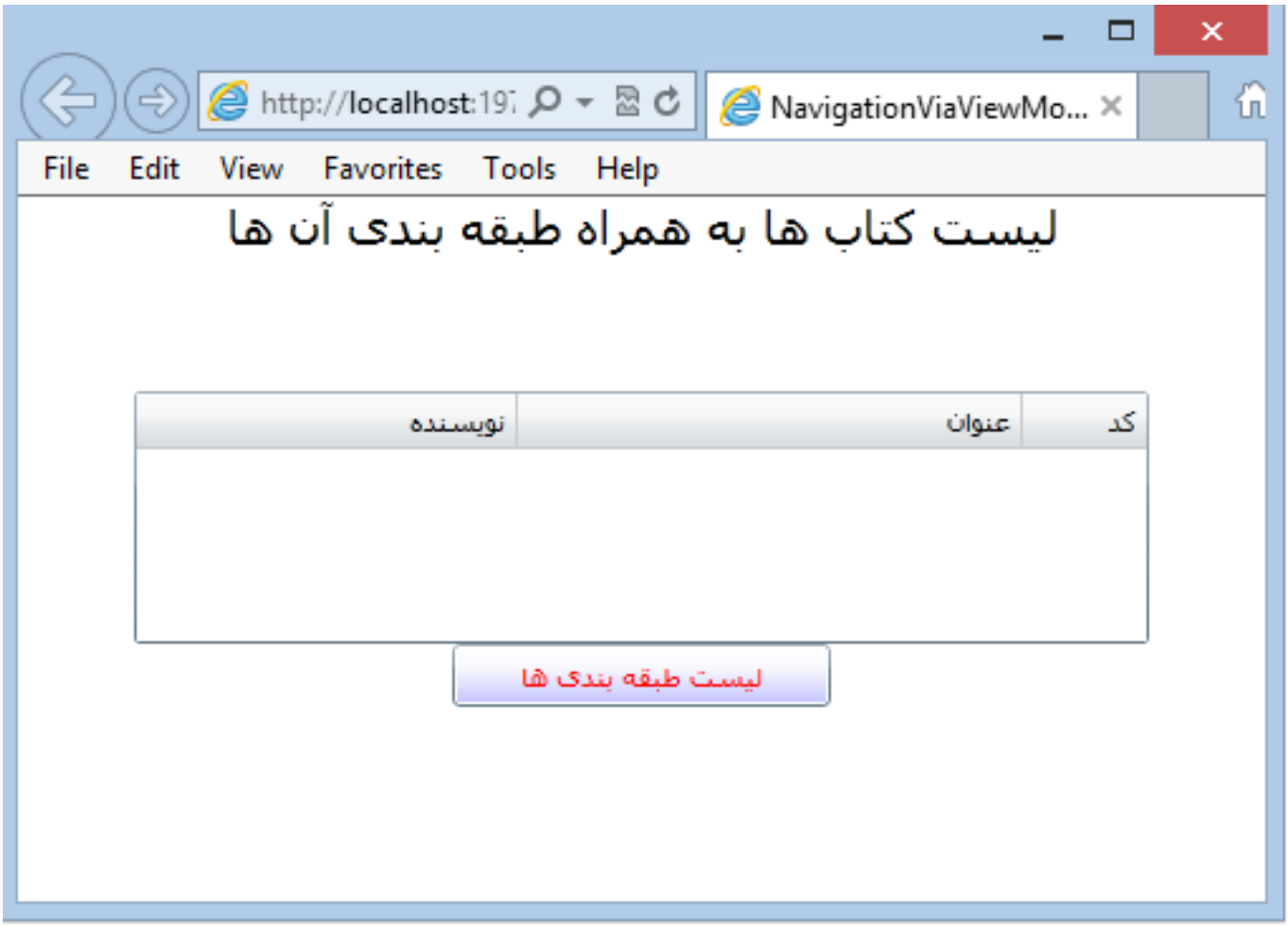
نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۴/۲۲ ۲۰:۱۶

در قسمت سوم ، سورس پیوست شده

عنوان: **آموزش Prism #3**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۴/۱۶ ۸:۳۰
آدرس: www.dotnettips.info
گروه‌ها: MVVM, Silverlight, WPF, prism

در پست‌های قبلی با Prism و روش استفاده از آن آشنا شدیم (قسمت اول) و (قسمت دوم). در این پست با استفاده از Mef قصد ایجاد یک پروژه Silverlight رو به صورت ماژولار داریم. مثال پیاده سازی شده در پست قبلی را در این پست به صورت دیگر پیاده سازی خواهیم کرد.
تفاوت‌های پیاده سازی مثال پست قبلی با این پست:
در مثال قبل پروژه به صورت Desktop و با WPF پیاده سازی شده بود ولی در این مثال با Silverlight می‌باشد؛
در مثال قبل از UnityBootstrapper استفاده شده بود ولی در این مثال از MefBootstrapper؛
در مثال قبل هر View در یک ماژول قرار داشت ولی در این مثال هر دو View را در یک ماژول قرار دادم؛
در مثال قبل از Prism Libary 2.x استفاده شده بود ولی در این مثال از PrismLibrary 4.x؛
و...

نکته : برای فهم بهتر مفاهیم، آشنایی اولیه با MEF و مفاهیمی نظیر Export و Import و AggregateCatalog و AssemblyCatalog نیاز است. در صورتی که با این مطالب آشنایی ندارید می‌توانید از (^) شروع کنید.



برای شروع یک پروژه Silverlight ایجاد کنید. بعد از اضافه شدن دو پروژه Silverlight و Web، یک Silverlight Class

Library جدید بسازید.
ابتدا یک Page ایجاد کنید و کدهای زیر را در آن کپی کنید.

```
<UserControl
             x:Class="Module1.Module1View1"
             xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" FlowDirection="RightToLeft"
FontFamily="Tahoma">
    <StackPanel>
        <sdk:DataGrid Height="100">
            <sdk:DataGrid.Columns>
                <sdk:DataGridTextColumn Header="کد" Width="50"  />
                <sdk:DataGridTextColumn Header="عنوان" Width="200" />
                <sdk:DataGridTextColumn Header="نویسنده" Width="150"  />
            </sdk:DataGrid.Columns>
        </sdk:DataGrid>
        <Button x:Name="NextViewButton"
                Width="150"
                Height="25"
                Foreground="Red"
                Background="Blue"
                Content="لیست طبقه بندی ها" />
    </StackPanel>
</UserControl>
```

بر روی Page مربوطه راست کلیک کنید و گزینه ViewCode را انتخاب کنید و کدهای زیر را در آن کپی کنید.

```
[Export(typeof(Module1View1))]
    public partial class Module1View1 : UserControl
    {
        [Import]
        public IRegionManager TheRegionManager { private get; set; }

        public Module1View1()
        {
            InitializeComponent();

            NextViewButton.Click += NextViewButton_Click;
        }

        void NextViewButton_Click(object sender, RoutedEventArgs e)
        {
            TheRegionManager.RequestNavigate
            (
                "MyRegion1",
                new Uri("Module1View2", UriKind.Relative),
                a => { }
            );
        }
    }
```

ابتدا خود این View باید حتما Export شود. در رویداد کلیک با استفاده از متد RequestNavigate می‌توانیم به View مورد نظر برای نمایش در Shell اشاره کنیم و این View در Region نمایش داده می‌شود. به دلیل اینکه در این کلاس به RegionManager نیاز داریم از ImportAttribute استفاده کردیم. این بدین معنی است که کلاس Module1View1 وابستگی مستقیم به IRegionManager دارد.

حال یک Page دیگر برای طبقه بندی کتاب‌ها ایجاد کنید و کدهای زیر را در آن کپی کنید.

```
<UserControl
             xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk"
x:Class="Module1.Module1View2"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" FlowDirection="RightToLeft"
FontFamily="Tahoma">
    <StackPanel>
        <sdk:DataGrid Height="100">
            <sdk:DataGrid.Columns>
                <sdk:DataGridTextColumn Header="کد" Width="150"/>
                <sdk:DataGridTextColumn Header="عنوان" Width="150"/>
```

```
            </sdk:DataGrid.Columns>
        </sdk:DataGrid>
        <Button x:Name="NextViewButton"
                Width="150"
                Height="25"
                Foreground="Green"
                Background="Yellow"
                Content="لیست کتاب ها" />
    </StackPanel>
</UserControl>
```

در Code Behind این Page نیز کدهای زیر را قرار دهید.

```
using Microsoft.Practices.Prism.Regions;
using System;
using System.ComponentModel.Composition;
using System.Windows;
using System.Windows.Controls;

namespace Module1
{
    [Export]
    public partial class Module1View2 : UserControl
    {
        IRegion _region1;

        [ImportingConstructor]
        public Module1View2( [Import] IRegionManager regionManager )
        {
            InitializeComponent();

            ViewModel viewModel = new ViewModel();
            DataContext = viewModel;

            viewModel.ShouldNavigateFromCurrentViewEvent += () => { return true; };

            _region1 = regionManager.Regions["MyRegion1"];

            NextViewButton.Click += NextViewButton_Click;
        }

        void NextViewButton_Click( object sender, RoutedEventArgs e )
        {
            _region1.RequestNavigate
            (
                new Uri( "Module1View1", UriKind.Relative ),
                a => { }
            );
        }
    }
}
```

در این ماژول برای اینکه بتوانیم حالت گردشی در فراخوانی ماژول‌ها را داشته باشیم ابتدا DataContext این کلاس را برابر با ViewModel ساخته شده قرار دادیم. با استفاده از رویداد ShoudlNavigateFromCurrentViewEvent که در کلاس ViewModel وجود دارد تعیین می‌کنیم که آیا باید از این View به View قبلی برگشت داشته باشیم یا نه. در صورتی که مقدار false برگشت داده شود خواهید دید که امکان فراخوانی View1 از View2 امکان پذیر نیست. در رویداد کلیک نیز همانند Page قبلی با استفاده از RegionManager و متد RequestNavigate به View مورد نظر راهبری کرده ایم.

نکته: اگر یک کلاس، سازنده با پارامتر داشته باشد باید با استفاده از ImportingConstructor حتما سازنده مورد نظر را هنگام وهله سازی مشخص کنیم در غیر این صورت با Exception مواجه خواهید شد.

حال قصد ایجاد کلاس ViewModel بالا را داریم:

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
```

```
using System.Windows.Shapes;
using System.ComponentModel.Composition;
using Microsoft.Practices.Prism.Regions;

namespace Module1
{
    public class ViewModel : IConfirmNavigationRequest
    {
        public event Func<bool> ShouldNavigateFromCurrentViewEvent;

        public bool IsNavigationTarget( NavigationContext navigationContext )
        {
            return true;
        }

        public void OnNavigatedTo( NavigationContext navigationContext )
        {

        }

        public void OnNavigatedFrom( NavigationContext navigationContext )
        {

        }

        public void ConfirmNavigationRequest( NavigationContext navigationContext, Action<bool>
continuationCallback )
        {
            bool shouldNavigateFromCurrentViewFlag = false;

            if ( ShouldNavigateFromCurrentViewEvent != null )
                shouldNavigateFromCurrentViewFlag = ShouldNavigateFromCurrentViewEvent();

            continuationCallback( shouldNavigateFromCurrentViewFlag );
        }
    }
}
```

توضیح متدهای بالا:

IsNavigateTarget : برای تعیین اینکه آیا کلاس پیاده سازی کننده اینترفیس، می‌تواند عملیات راهبری را مدیریت کند یا نه.

OnNavigateTo : زمانی عملیات راهبری وارد View شود(بهتره بگم View مورد نظر در Region صفحه لود شود) این متد فراخوانی می‌شود.

OnNavigateFrom : زمانی که راهبری از این View خارج می‌شود (View از حالت لود خارج می‌شود) این متد فراخوانی خواهد شد.

ConfirmNavigationRequest : برای تایید عملیات راهبری توسط کلاس پیاده سازی کننده اینترفیس استفاده می‌شود.

حال یک کلاس برای پیاده سازی و مدیریت ماژول می‌سازیم.

```
using Microsoft.Practices.Prism.MefExtensions.Modularity;
using Microsoft.Practices.Prism.Modularity;
using Microsoft.Practices.Prism.Regions;
using System.ComponentModel.Composition;

namespace Module1
{
    [ModuleExport(typeof(Module1Impl))]
    public class Module1Impl : IModule
    {
        [Import]
        public IRegionManager TheRegionManager { private get; set; }

        public void Initialize()
        {
            TheRegionManager.RegisterViewWithRegion("MyRegion1", typeof(Module1View1));

            TheRegionManager.RegisterViewWithRegion("MyRegion1", typeof(Module1View2));
        }
    }
}
```

همان طور که مشاهده می‌کنید از ModuleExportAttribute برای شناسایی ماژول توسط MefBootstrapper استفاده کردیم و نوع آن را ModuleImpl قرار دادیم. ImportAttribute استفاده شده در این کلاس و خاصیت TheRegionManager برای این است که در هنگام ساخت Instance از این کلاس IRegionManager موجود در Container باید در اختیار این کلاس قرار گیرد(نشان دهنده وابستگی مستقیم این کلاس با IRegionManager است). روش دیگر این است که در سازنده این کلاس هم این اینترفیس را تزریق کنیم.

در متد Initialize برای RegionManager دو View ساخته شده را رجیستر کردیم. این کار باید به تعداد Viewهای موجود در ماژول انجام شود.

Shell

در پروژه اصلی یک Page به نام Shell ایجاد کنید و کدهای زیر را در آن کپی کنید.

```
<UserControl x:Class="NavigationViaViewModel.Shell"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:prism="http://www.codeplex.com/prism" FlowDirection="RightToLeft"
FontFamily="Tahoma">

    <Grid x:Name="LayoutRoot"
          Background="White">
        <TextBlock Text="لیست کتاب‌ها به همراه طبقه بندی آن ها"
                   FontSize="19"
                   Foreground="Black"
                   HorizontalAlignment="Center"
                   VerticalAlignment="Top" />
        <ContentControl HorizontalAlignment="Center"
                        VerticalAlignment="Center"
                        prism:RegionManager.RegionName="MyRegion1" />
    </Grid>
</UserControl>
```

همانند مثال قبلی یک ContentControl داریم و به وسیله RegionName که یک AttachedProperty است یک Region به نام MyRegion1 ایجاد کردیم. تمام ماژول‌های این مثال در این محدوده نمایش داده خواهند شد.

Bootstrapper

حال نیاز به یک Bootstrapper داریم. برای این کار یک کلاس به نام TheBootstrapper بسازید:

```
using Microsoft.Practices.Prism.MefExtensions;
using Microsoft.Practices.Prism.Modularity;
using System.ComponentModel.Composition.Hosting;
using System.Windows;

namespace NavigationViaViewModel
{
    public class TheBootstrapper : MefBootstrapper
    {
        protected override void InitializeShell()
        {
            base.InitializeShell();

            Application.Current.RootVisual = (UIElement)Shell;
        }

        protected override DependencyObject CreateShell()
        {
            return Container.GetExportedValue<Shell>();
        }

        protected override void ConfigureAggregateCatalog()
        {
            base.ConfigureAggregateCatalog();
            AggregateCatalog.Catalogs.Add(new AssemblyCatalog(this.GetType().Assembly));
        }

        protected override IModuleCatalog CreateModuleCatalog()
        {
            ModuleCatalog moduleCatalog = new ModuleCatalog();

            moduleCatalog.AddModule
            (
                new ModuleInfo
                {
```

```
                    InitializationMode = InitializationMode.WhenAvailable,
                    Ref = "Module1.xap",
                    ModuleName = "Module1Impl",
                    ModuleType = "Module1.Module1Impl, Module1, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null"
                }
            );

            return moduleCatalog;
        }
    }
}
```

متد CreateShell اولین متد در این کلاس است که اجرا خواهد شد. بعد از متد CreateShell، متد InitializeShell اجرا خواهد شد. خاصیت Shell دقیقا به مقدار برگشتی متد CreateShell اشاره خواهد کرد. در متد InitializeShell مقدار خاصیت Shell به RootVisual این پروژه اشاره می‌کند(مانند MainWindow در کلاس Application پروژه‌های WPF).

متد ConfigureAggregateCatalog برای مدیریت کاتالوگ‌ها و ماژول‌ها که هر کدام در یک اسمبلی جدا وجود خواهند شد استفاده می‌شود. در این متد من از AssemblyCatalog استفاده کردم. AssemblyCatalog تمام کلاس هایی که ExportAttribute را به همراه دارند شناسایی می‌کند و آن‌ها را در Container نگهداری خواهد کرد(^). مانند یک ServiceLocator در Microsoft unity Service Locator(^) .

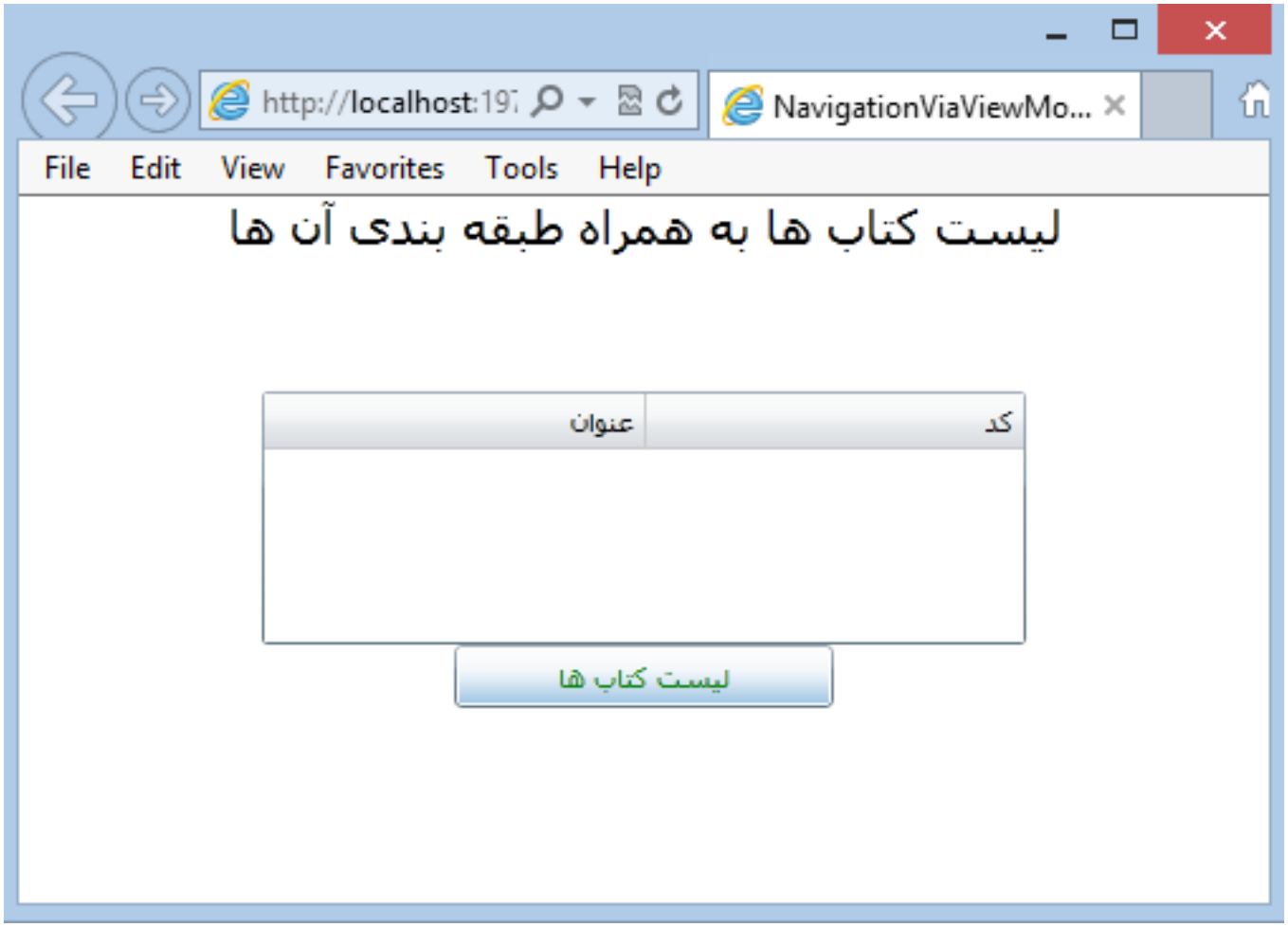
متد آخر به نام CreateModuleCatalog است و باید در آن تمام ماژول‌های برنامه را به کلاس ModuleCatalog اضافه کنیم. در مثال پست قبلی به دلیل استفاده از UnityBootstrapper باید این کار را از طریق BuildEvent ها مدیریت می‌کردیم ولی در این جا Mef به راحتی این کار را انجام خواهد داد.

تغییرات زیر را در فایل App.Xaml قرار دهید و پروژه را اجرا کنید.

```
public partial class App : Application
 {
        public App()
        {
            this.Startup += this.Application_Startup;
            InitializeComponent();
        }

        private void Application_Startup(object sender, StartupEventArgs e)
        {
           var bootstrapper = new TheBootstrapper();
            bootstrapper.Run();
        }
}
```

با کلیک بر روی ماژول عملیات راهبری برای ماژول انجام خواهد شد.



دریافت سورس پروژه

ادامه دارد..

نظرات خوانندگان

نویسنده: javad
تاریخ: ۱۳۹۲/۰۵/۰۵ ۱۲:۱

سلام
اگه می‌شه آموزش استفاده از Entity Framwork در prism را نیز قرار دهید . می‌خوام ماژول‌های مختلف از یک دیتا بیس استفاده کنند و یک EF مشترک داشته باشند ؟

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۵/۰۵ ۱۳:۱۰

بسیار ساده است. شما نیاز به طراحی یک UnitOfWork بر اساس EF دارید(^). بعد از آن کافیست کدهای مورد نظر برای عملیات CRUD رو در ViewModelهای هر ماژول بنویسید. در پروژه‌های Silverlight هم می‌تونید از RIA Service و EF استفاده کنید. سعی می‌کنم در صورت داشتن زمان کافی یک پست را به این مطلب اختصاص بدم.

نویسنده: imo0
تاریخ: ۱۳۹۲/۰۶/۲۰ ۱۶:۳۴

سلام . دستتون درد نکنه اقای پاکدل . فقط یه چیزی!
یکی اینکه این اموزشتونو اگه میشه یکم سریعتر بدید . اون روش قبلیه که گفتید رو من خوندم خیلی واضح‌تر توضیح داده بودین . اما از این یکی زیاد نمیتونم درکش کنم.
اگه میشه لطفا رو یه ساختار کنین . یعنی مثلا همین Prism رو با همون الگویه MVVM ای که داره تویه WPF بگین که ما هم بتونیم استفاده کنیم . شما یکی شو با یه روش، یکی دیگشو با یه روشه دیگه و باز اینارو هر کدوم یکی تو Silver و اون یکی تو WPF . این نظر منه . اگه شما یه دونشونو انتخاب کنید و همینطوری ادامه بدین بهتره که ما هم بتونیم برای خودمون یه جمع بندی و یه راه مشخص پیدا کنیم . سایت واقعا عالی دارین . خیلی چیزا من از این سایت یاد گرفتم . این ماژولار بودن تو این سبک و تا این سطح خیلی برام کاربردی و مهمه . میخوام پایه پروژه‌های شرکتو بر همین روال قرار بدم . اگه میشد شما از همین Prism و این MEF یه پروژه WPF بسازین فقط یکی دوتا ماژول ساده براش پیاده سازه کنین و یه فیلم بگیرین خیلی ممنون میشم . میخوام ا این روش استفاده کنم اما روال کار برام مبهمه . اگه کتاب یا سری آموزشی در این باره هم دارین بزارین ما استفاده کنیم . اموزش هاتونم من هر روز میام میخونم و چک میکنم اما خیلی دیر دیر مطلب میزارین . حتما این اموزشو ادامه بدین . مخصوصا Prism With MEF In WPF . خلیی باحالین....

نویسنده: imo0
تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۷:۱۵

سلام . خسته نباشید . من اگه بخوام تمام ماژول‌ها به صورت داینامیک از تو یک فولدر بخونه باید چیکار کرد. داخل WPF از کلاس DirectoryCatalog استفاده میشه کرد . اما برای سیلورلایت این کلاس وجود نداره . اگه میشه راهنمایی بفرمایین .

نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۷:۲۸

ابتدا اسمبلی System.ComponentModel.Composition را به پروژه خود اضافه نمایید. در فضای نام System.ComponentModel.Composition.Hosting کلاس DirectoryCatalog موجود است.

نویسنده: imo0

تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۷:۴۲

با تشکر ولی به نظر سیلورلایت نداره . لطفا اینجا رو یه چک بکنید . نوشته که

"**Note:** DirectoryCatalog is not supported in Silverlight ".

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۹/۲۵ ۲۲:۴۴

در همون لینکی که دادید یک پیاده سازی کمکی ذکر شده: A DirectoryCatalog class for Silverlight

DeploymentCatalog هم هست