عنوان: **استفاده از StructureMap جهت تزریق وابستگی‌ها در برنامه‌های WPF و الگوی MVVM**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۰۵ ۱۷:۱۲
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns, Dependency Injection, IoC

در این قسمت قصد داریم همانند کنترلرها در ASP.NET MVC، کار تزریق وابستگی‌ها را در متدهای سازنده ViewModelهای WPF بدون استفاده از الگوی Service locator انجام دهیم؛ برای مثال:

```
public class TestViewModel
    {
        private readonly ITestService _testService;
        public TestViewModel(ITestService testService) //تزریق وابستگی در سازنده کلاس
        {
            _testService = testService;
        }
```

و همچنین کار اتصال یک ViewModel، به View متناظر آن‌را نیز خودکار کنیم. قراردادی را نیز در اینجا بکار خواهیم گرفت: نام تمام Viewهای برنامه به View ختم می‌شوند و نام ViewModelها به ViewModel. برای مثال **ViewModel** Test و **View** Test معرف یک ViewModel و View متناظر خواهند بود.

ساختار کلاس‌های لایه سرویس برنامه

```
namespace DI07.Services
{
    public interface ITestService
    {
        string Test();
    }
}
```

```
namespace DI07.Services
{
    public class TestService: ITestService
    {
        public string Test()
        {
            return "برای آزمایش";
        }
    }
}
```

یک پروژه WPF را آغاز کرده و سپس یک پروژه Class library دیگر را به نام Services با دو کلاس و اینترفیس فوق، به آن اضافه کنید. هدف از این کلاس‌ها صرفا آشنایی با نحوه تزریق وابستگی‌ها در سازنده یک کلاس ViewModel در WPF است.

علامتگذاری ViewModelها

در ادامه یک اینترفیس خالی را به نام IViewModel مشاهده می‌کنید:

```
namespace DI07.Core
{
    public interface IViewModel // از این اینترفیس خالی برای یافتن و علامتگذاری ویوو مدل‌ها استفاده می‌کنیم
    {
    }
}
```

از این اینترفیس برای علامتگذاری ViewModelهای برنامه استفاده خواهد شد. این روش، یکی از انواع روش‌هایی است که در مباحث Reflection برای یافتن کلاس‌هایی از نوع مشخص استفاده می‌شود.
برای نمونه کلاس TestViewModel برنامه، با پیاده سازی IViewModel، به نوعی نشانه گذاری نیز شده است:

```
using DI07.Services;
using DI07.Core;

namespace DI07.ViewModels
{
    public class TestViewModel : IViewModel // علامتگذاری ویوو مدل
    {
        private readonly ITestService _testService;
        public TestViewModel(ITestService testService) //تزریق وابستگی در سازنده کلاس
        {
            _testService = testService;
        }

        public string Data
        {
            get { return _testService.Test(); }
        }
    }
}
```

تنظیمات آغازین IoC Container مورد استفاده

در کلاس استاندارد App برنامه WPF خود، کار تنظیمات اولیه StructureMap را انجام خواهیم داد:

```
using System.Windows;
using DI07.Core;
using DI07.Services;
using StructureMap;

namespace DI07
{
    public partial class App
    {
        protected override void OnStartup(StartupEventArgs e)
        {
            base.OnStartup(e);

            ObjectFactory.Configure(cfg =>
            {
                cfg.For<ITestService>().Use<TestService>();

                cfg.Scan(scan =>
                {
                    scan.TheCallingAssembly();
                    // Add all types that implement IView into the container,
                    // and name each specific type by the short type name.
                    scan.AddAllTypesOf<IViewModel>().NameBy(type => type.Name);
                    scan.WithDefaultConventions();
                });
            });
        }
    }
}
```

در اینجا عنوان شده است که اگر نیاز به نوع ITestService وجود داشت، کلاس TestService را وهله سازی کن.
همچنین در ادامه از قابلیت اسکن این IoC Container برای یافتن کلاس‌هایی که IViewModel را در اسمبلی جاری پیاده سازی کرده‌اند، استفاده شده است. متد NameBy، سبب می‌شود که بتوان به این نوع‌های یافت شده از طریق نام کلاس‌های متناظر دسترسی یافت.

اتصال خودکار ViewModelها به Viewهای برنامه

```
using System.Windows.Controls;
using StructureMap;
```

```
namespace DI07.Core
{
    /// <summary>
    /// Stitches together a view and its view-model
    /// </summary>
    public static class ViewModelFactory
    {
        public static void WireUp(this ContentControl control)
        {
            var viewName = control.GetType().Name;
            var viewModelName = string.Concat(viewName, "Model"); //قرار داد نامگذاری ما است
            control.Loaded += (s, e) =>
            {
                control.DataContext = ObjectFactory.GetNamedInstance<IViewModel>(viewModelName);
            };
        }
    }
}
```

اکنون که کار علامتگذاری ViewModelها انجام شده و همچنین IoC Container ما می‌داند که چگونه باید آن‌ها را در اسمبلی جاری جستجو کند، مرحله بعدی، ایجاد کلاسی است که از این تنظیمات استفاده می‌کند. در کلاس ViewModelFactory، متد WireUp، وهله‌ای از یک View را دریافت کرده، نام آن‌را استخراج می‌کند و سپس بر اساس قراردادی که در ابتدای بحث وضع کردیم، نام ViewModel متناظر را یافته و سپس زمانیکه این View بارگذاری می‌شود، به صورت خودکار DataContext آن‌را به کمک StructureMap وهله سازی می‌کند. این وهله سازی به همراه تزریق خودکار وابستگی‌ها در سازنده کلاس ViewModel نیز خواهد بود.

استفاده از کلاس ViewModelFactory

در ادامه کدهای TestView و پنجره اصلی برنامه را مشاهده می‌کنید:

```
<UserControl x:Class="DI07.Views.TestView"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
    <Grid>
        <TextBlock Text="{Binding Data}" />
    </Grid>
</UserControl>


<Window x:Class="DI07.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:Views="clr-namespace:DI07.Views"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Views:TestView />
    </Grid>
</Window>
```

در فایل Code behind مرتبط با TestView تنها کافی است سطر فراخوانی this.WireUp اضافه شود تا کار تزریق وابستگی‌ها، وهله سازی ViewModel متناظر و همچنین مقدار دهی DataContext آن به صورت خودکار انجام شود:

```
using DI07.Core;

namespace DI07.Views
{
    public partial class TestView
    {
        public TestView()
        {
            InitializeComponent();
            this.WireUp(); //تزریق خودکار وابستگی‌ها و یافتن ویوو مدل متناظر
```

```
        }
    }
}
```

دریافت پروژه کامل این قسمت

DI07.zip

نظرات خوانندگان

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۲/۰۵ ۱۷:۵۵

مدیریت طول عمر DbContext به کمک StructureMap در برنامه‌های WPF و الگوی MVVM چگونه است ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۰۵ ۱۹:۵۲

- هر دوره قسمت اختصاصی رو داره به نام « پرسش و پاسخ » برای طرح این نوع سؤالات خارج از موضوع مطلب جاری، اما مرتبط با عنوان دوره.
- در این مورد DbContext در همان پرسش و پاسخ‌های قسمت 12 سری EF بحث شده. اینجا

برای تکرار:

«... در یک برنامه مبتنی بر MVVM، مدیریت طول عمر یک context در طول عمر ViewModel برنامه است. در یک برنامه ویندوزی تا زمانیکه یک فرم باز است، اشیاء آن تخریب نخواهند شد. بنابراین مدیریت context در برنامه‌های ویندوزی «دستی» است. در زمان شروع فرم context شروع خواهد شد، زمان تخریب/بستن آن، با بستن یا dispose یک context، خودبخود اتصالات هم قطع خواهند شد.

بنابراین در برنامه‌های وب «context/session *per* http *request* » داریم؛ در برنامه‌های ویندوزی «context *per operation* or *per form* ». یعنی می‌تونید بسته به معماری برنامه ویندوزی خود، context را در سطح یک فرم تعریف کنید و مدیریت؛ *و یا در سطح یک عملیات* کوتاه مانند یک کلیک ...»

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۲۷ ۱۱:۳۹

یک نکته جالب!

```
public class FrameFactory : Frame
{
        protected override void OnContentChanged(object oldContent, object newContent)
        {
            base.OnContentChanged(oldContent, newContent);
            ((FrameworkElement)newContent).WireUp(); //کار تزریق وابستگی‌ها و وهله سازی ویوو مدل مرتبط
انجام خواهد شد
        }
}
```

می‌شود یک کنترل فریم سفارشی ایجاد کرد. سپس در متد OnContentChanged فرصت تزریق خودکار وابستگی‌ها به صفحه‌ای که در حال اضافه شدن و نمایش است وجود خواهد داشت.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۲ ۱۳:۱۳

چند مثال تکمیلی دیگر

Restructuring your legacy code using MVVM and Dependency Injection with Unity

IOC Containers and MVVM

Using Structuremap to resolve ViewModels

Implementing MVVM Light with Structure Map