عنوان: **Expression Blend WPF Tutorial**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۱/۱۴ ۱۷:۴۱:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: WPF

دنبال یک سری ویدیوی آموزشی Expression blend بودم که آدرس زیر را پیدا کردم:

Expression Blend Beta Preview

امکان مشاهده‌ی رایگان آن‌ها موجود است، همچنین اگر برنامه‌ی internet download manager را نصب کنید، هنگام گشودن هر صفحه، یک آیکون ذخیره سازی ویدیوی مورد نظر نیز ظاهر می‌شود که به این صورت می‌توان تمام ویدیوها را دانلود کرد.

عنوان: **طول و عرض WPF**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۶/۱۵ ۲۲:۲۰:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: WPF

شاید بد نباشد این فناوری را از دیدگاه مدت زمانی که باید به آن تسلط پیدا کرد، بررسی نمود:

بله، مشکل در طول و عرض WPF بوده و مدت زمان یادگیری و تسلط کامل به آن، از فناوری‌های قبلی مطرح در دات نت فریم ورک بسیار بیشتر می‌باشد. (تعداد کلاس‌های آن تقریبا مساوی مجموع تعداد کلاس‌های نگارش WinForms 2 و ASP.Net است!)

در مقایسه با WinForms و ASP.Net هم موارد زیر قابل تامل است:
ASP.NET 2.0 شامل 1098 public types و 1551 classes است.
WinForms 2.0 شامل 777 public types و 1500 classes می‌باشد.
سیلورلایت 2 را هم که در تصویر مشاهده می‌کنید. شامل 376 public types و 335 classes است.

ماخذ

عنوان: **آموزش رایگان XAML از مایکروسافت**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۶/۲۶ ۱۵:۲۳:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: WPF

یک دوره آموزشی رایگان XAML اخیرا از طرف مایکروسافت ارائه شده است که از طریق آدرس زیر قابل دسترسی است:

Clinic 6375AE: Introduction to XAML

این کلینیک آموزشی شامل موارد زیر است:

Navigation Overview
Clinic Information
Introduction to XAML
Overview of XAML
?Why XAML
XAML Layouts
Module Summary
XAML and WPF In Action
XAML in a Browser
Using XAML and code-behind in Desktop Applications
Module Summary
Unique Features of XAML
Resources
Styles and ControlTemplates
Module Summary
Glossary

این ماژول به صورت آفلاین نیز قابل دریافت است (به حجم 44 مگابایت) اما پیش از آن باید برنامه offline player آن‌را نصب نمود و طبق روال معمول سایت مایکروسافت، بهتر است از IE جهت مرور این صفحات استفاده کرد.

عنوان: **سری آموزشی PRISM**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۸/۲۰ ۱۸:۵۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: WPF

PRISM یا Composite Application Guidance الگوهایی را برای تولید برنامه‌های WPF ویا Silverlight ماژولار با قابلیت تست پذیری بالا ارائه می‌دهند. شعار این مجموعه built for change و built to last است که به معنای تهیه سیستم‌هایی با قابلیت تغییر بالا و همچنین سهولت نگهداری آن‌ها در دراز مدت می‌باشد.
جناب Mike Taulty را احتمالا با ویدیوهای آموزش WCF به خاطر دارید. ایشان مجموعه جدیدی را به نام Video Series on PRISM for Silverlight 3 تهیه کرده‌اند که از لینک‌های زیر قابل دریافت است:

Microsoft patterns & practices
proven practices for predictable results

Part 1:

Taking Sketched Code Towards Unity

Part 2:

Dependency Injection with Unity

Part 3:

Modularity with Prism

Part 4:

The Unity Bootstrapper

Part 5:

Moving to a Modular Silverlight Project

Part 6:

Shells, Regions, Views

Part 7:

Commands

Part 8:

Loosely Coupled Events with Event Aggregation

Part 9:

Sharing Data via Region Contexts

```
Part 10:
```

"A Larger Example: "Email Client

نظرات خوانندگان

نویسنده: ..::A-3BT::..
تاریخ: ۱۳۸۸/۰۸/۲۰ ۲۳:۱۱:۴۳

Mike Taulty آدم خیلی فعالی در این زمینه هستش ، من خیلی پادکست ازش دیدم در همه زمینه ها ، و از شما هم بابت لینک مفیدتون نهایت تشکر رو دارم ، به امید موفقیت روزافزون شما ، من یکی از خواننده گان ثابت وبلاگ شما هستم!

عنوان: **ویدیوهای رایگان آموزشی WPF**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۱۲/۰۷ ۱۴:۳۶:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: WPF

یک سری ویدیوی آموزشی رایگان WPF بجا مانده از Boot Camp 2008 را از اینجا می‌توانید دریافت کنید که شامل مباحث مختلف بایندینگ، prism ، styles و غیره است.

نظرات خوانندگان

نویسنده: Farid Abdi
تاریخ: ۱۳۸۸/۱۲/۰۹ ۰۹:۳۸:۲۳

با سلام
اینجانب دارای یک وبلاگ در مورد مسائل کامپیوتری هستم . در این وبلاگ ، اخبار و مسائل کامپیوتری را از نظر سیاسی و امنیتی بررسی و تحلیل می کنم . لطفا از آن دیدن کنید و در صورتیکه آن را پسندیدید ، لینکش کنید . بسیار ممنون

www.java9000.blogfa.com

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۱۲/۰۹ ۱۲:۲۷:۳۸

سلام
موفق باشید. می‌تونید مطالب جدید خودتون را در سایت زیر لینک دهید تا بقیه برنامه نویس‌ها نیز مطلع شوند
http://www.idevcenter.com

عنوان: **ارتقاء از WinForms به WPF**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۱۲/۱۰ ۱۳:۲۹:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: WPF

اگر مدت‌ها کارتان برنامه نویسی WinForms بوده و اکنون احساس کرده‌اید که دیگر WinForms آنچنان توسعه و بسط نخواهد یافت و اکنون WPF تبدیل به انتخاب اصلی شرکت‌های بزرگ شده است و همچنین از پرسه زدن در فوروم‌های وارز جهت یافتن فلان کامپوننت خاص برای زیباسازی ظاهر برنامه‌های خود خسته شده‌اید و نیاز به معادل بهتری که اساسا در جهت حذف این بازار سیاه تهیه شده است، احساس می‌کنید، بهترین گزینه‌ی موجود WPF خواهد بود که با کمی دقت، می‌توان پروژه‌های آن‌را تبدیل به پروژه‌های وب نیز نمود. مطلب 54 صفحه‌ای ذیل، خلاصه‌ی کاربردی سریعی را جهت ارتقاء برنامه نویس‌های WinForms به WPF ارائه می‌دهد:

WPF for those who know Windows Forms
ماخذ

نظرات خوانندگان

نویسنده: ..::A-3BT::..
تاریخ: ۱۳۸۸/۱۲/۱۱ ۰۸:۲۱:۰۱

خیلی ممنون ، ولی یه برنامه بود ویندوز فرم رو با WPF تبدیل می کرد ، لینکی که دادی خیلی خوب بود ممنون

نویسنده: Mehran
تاریخ: ۱۳۸۸/۱۲/۱۱ ۱۴:۴۷:۴۲

winform ها جای خودشان را در صنعت نرمافزار دارند با اینکه من مدتی است wpf کار میمکنم اما واقعا بعضی از پروژه ها نیازی به پیاده سازی با wpf ندارند. این احساس که برنامه نویس ها دوست دارند نرم افزار ها از ui بهتری بهره ببرند دلیل به بهترین دلیل استفاده ار نرم افزار های wpf است اما تغییر شیوه طراحی یک فرم در یک winform یک فایل XAML (بخوانید zammel) بزرگترین مانع برای مهاجرت برنامه نویسان winfrom به wpf است.

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۱۲/۱۱ ۱۵:۰۶:۵۳

درسته. ادیتور xaml ویژوال استودیوی 2008 به نظر من نیم پخته است. این مورد در VS2010 خیلی بهتر شده خصوصا اینکه خود vs2010 هم بر مبنای wpf است.
البته blend هم جایگاه خودش را دارد و شاید MS برای فروش بیشتر blend ادیتور VS2008 را جدی نگرفته.

نویسنده: ..::A-3BT::..
تاریخ: ۱۳۸۸/۱۲/۱۳ ۰۱:۳۲:۲۰

البته blend 3 خیلی نسبت به نسخه های قبلی اش بهتر شده

نویسنده: Sirasad
تاریخ: ۱۳۸۸/۱۲/۱۳ ۱۲:۴۲:۳۵

"از پرسه زدن در فوروم‌های وارز جهت یافتن فلان کامپوننت خاص برای زیباسازی ظاهر برنامه‌های خود خسته شده‌اید"

آیا فقط برای این ویژگی می توان ریسک مهاجرت به wpf را قبول کرد ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۱۲/۱۳ ۱۳:۰۰:۱۷

- تعریف شما از ریسک چی هست؟ از چه چیزی واهمه دارید؟! یا اگر به wpf مهاجرت کردید چه چیزی را از دست خواهید داد؟!
"تفاوت‌های یک برنامه نویس کارمند با یک برنامه نویس علاقمند" را مطالعه کردید؟
http://www.dotnettips.info/2010/02/blog-post.html

برای یک کارمند شاید زیاد فرقی نکنه. حق با شماست.

- ضمنا ظاهر زیبا فقط قسمتی از قابلیت‌هایی است که بدست می‌آورید؟ سیلورلایت که برادر کوچکتر wpf محسوب می‌شود را هم فراموش نکنید. با یک تیر دو نشان (البته نیاز به رعایت یک سری مسایل دارد).

اگر وقت کردید مطلب زیر را مطالعه کنید تا ابعاد مزایای این مهاجرت را به صورت تخمینی درک کنید:
http://www.dotnettips.info/2009/09/wpf.html

نویسنده: ..::A-3BT::..
تاریخ: ۱۳۸۸/۱۲/۱۴ ۱۳:۵۴:۲۸

البته از یک لحاظ به سیر اسد حق میدم , مهاجرت به یک تکنولوژی مستلزم زمانی برای یادگیری و اون هست و ترس از Backward Compatibility که خیلی ها از رفتن به سمت تکنولوژی های جدید باز داشته ولی خوب اینها همش ناشی از عدم آشنایی با چیزی و ابعاد اون هست شما به WPF این مشکل رو نخواهید داشت WPF می تونه خیلی راحت با سیستمهای قبلی شما سازگار بشه حتی شما در سیستمهایی که با C++ نوشتید می تونید از اون استفاده کنید ولی خوب این مستلزم اینه که شما به C++ دات نت کوچ کنید , خوب ممکنه برنامه های شما با زبانهای دیگه نوشته باشه مثل VB نه VB.NET شما بازهم نگران استفاده از WPF در اونها نباشید بدون مشکل این امر محقق میشه , یک مورد دیگری که خیلی مساله ساز میشه برای شرکتها و خیلی از برنامه نویسها هزینه های آموزش تکنولوژی های جدید هست , ولی خوب به نظر من این امر هم خیلی مساله بغرنجی نیست که نگران اون هستند اگر تیم برنامه نویسی واقعا" اصولی مطالب رو آموخته باشن می تونند براحتی به سمت تکنولوژی های جدید برن این امر خیلی دور از واقعیت نیست , ولی خوب بهر حال عامه جامعه برنامه نویسها زیر با تکنولوژی جدید نمی رن و جدای از چیزهایی که گفتم آنها یک شعار دارن که می گن "اینی که داریم کفایت کار ما رو میکنه و پول ساز هست " , پس دلیلی بر کوچ کردن به تکنولوژی های جدید وجود نداره.

نویسنده: DoctorX
تاریخ: ۱۳۸۹/۰۳/۱۲ ۱۵:۵۳:۵۲

لینک فوت شده وحید جان ...

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۳/۱۲ ۱۸:۲۳:۱۲

به همان لینک ماخذ مراجعه کنید. لینک جدید را بر اساس تغییرات جدید وبلاگ‌های MSDN دارد (آخر مطلب پیوست شده).

نویسنده: سیروان عفیفی
تاریخ: ۱۳۹۱/۰۷/۱۴ ۲۱:۵۷

به نظرتون با توجه به شرایط حال حاضر لزومی به یادگیری WPF هست یا خیر؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۴ ۲۲:۵

بله. در WinRT هم با همین مفاهیم سروکار داریم. کمی کمتر شده، تعدادی کلاس و کنترل جدید به آن اضافه شده، اما اصول یکی است.
ضمن اینکه هنوز هم برنامه نویسی دسکتاپ مشتری خاص خودش را دارد.

عنوان: **خلاصه‌ای کاربردی در مورد Observable collection**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۱۲/۱۴ ۱۳:۳۲:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: WPF

Observable collection

در WPF را می‌توان نوعی لیست جنریک ویژه تعریف کرد که زمانیکه به کنترلی بایند شد، کنترل را از تغییرات خودش آگاه می‌کند. برای مثال اگر آیتمی به این لیست اضافه شد بلافاصله آن آیتم را در کنترل مقید به آن نیز خواهید دید، به همین ترتیب در مورد ویرایش و یا حذف یک آیتم، بدون نیاز به کوچکترین تماسی با کنترل مورد نظر. برای مثال اگر مقدار یک خاصیت را تغییر دادید، بلافاصله بدون اینکه به کنترل مقید به آن اعلام کنیم که لطفا این مورد ویژه را برای من تغییر بده، شاهد نتیجه‌ی نهایی خواهیم بود.

اما استفاده‌ی پیشرفته از این لیست جنریک ویژه به همینجا ختم نشده و حین اضافه کردن کمی پیچیدگی به برنامه مشکلات عدیده‌ای بروز می‌کنند که آن‌ها را جهت دسترسی ساده‌ی بعدی در زیر لیست می‌کنم:

الف) اصلا Observable collection چیست؟ چکار می‌کند؟
List vs ObservableCollection vs INotifyPropertyChanged in Silverlight

ب) نمی‌توانم از این مجموعه‌ی اشیای خودآگاه سازنده در یک ترد استفاده کنم. مشکل کجاست؟
این روزها نمی‌توان یک برنامه‌ی دسکتاپ خوب را بدون استفاده از تردها متصور شد. اما به محض سعی در به روز رسانی این لیست جنریک در یک ترد دیگر (ترد دیگر منظور هر تردی بجز ترد اصلی برنامه است که کار مدیریت رابط کاربر را به عهده دارد) خطای زیر ظاهر می‌شود:

```
This type of CollectionView does not support changes to its SourceCollection from a thread different
from the Dispatcher thread
```

راه حل:
Adding to an ObservableCollection from a background thread

ج) یکی از خاصیت‌های یک شیء این لیست جنریک ویژه را تغییر داده‌ام. اما هیچ تغییری در کنترل بایند شده به آن مشاهده نمی‌کنم. مشکل در کجاست؟
راه حل: پیاده سازی اینترفیس INotifyPropertyChanged را فراموش کرده‌اید:

Data Binding in WPF with the Monostate Pattern

د) خوب، این که خیلی دردسر دارد! راه ساده‌تری برای تعریف این موارد نیست؟!
هوشمندانه‌ترین روشی که برای حل این مساله تابحال دیده‌ام:
An easier way to manage INotifyPropertyChanged

نظرات خوانندگان

نویسنده: ..::A-3BT::..
تاریخ: ۱۳۸۸/۱۲/۱۴ ۱۴:۰۴:۳۳

تا پیش از این , این الگو فقط در WPF از طرف ماکروسافت پیاده سازی شده بود ولی در 4 net. ماکروسافت اون رو به عنوان بخشی از BCL در نظر گرفته

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۱۲/۱۴ ۱۴:۱۰:۳۷

البته بهتره بگیم استفاده از این الگو وگرنه تعریف آن در فضای نام system قرار دارد (System.Collections.ObjectModel) .

نویسنده: Mehran
تاریخ: ۱۳۸۸/۱۲/۱۴ ۱۶:۴۱:۰۰

ممنون از مطالب مفید شما واقعا بر روی لبه تکنولوژی های دات نت قدم بر می دارید. در مورد COLLECTION های OBSERVABLE این نکته هم حائز اهمیت است که این لیست ها دقیقا به اندازه نیاز کاربر در UI برنامه مقادیر را در حافظه لود کرده و اصطلاحا می توان با بکاربری این COLLECTIONها در یک کنترل خاصیت LazyLoading به کنترل بخشید.

نویسنده: mojtabakaviani
تاریخ: ۱۳۸۸/۱۲/۱۹ ۰۹:۳۰:۱۴

ممنون از مطالب مفیدتون...
اگه ممکنه قسمت خبر ها و تازه های دنیای کامپیوتر رو دوباره اضافه کنید...
از لینک های که می گذاشتید به یه عالمه تازه ای بیشتر می رسیدیم.

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۱۲/۱۹ ۱۰:۵۹:۵۳

سلام
هنوز هم اینکار رو می‌کنم. اما محل انتشار آن تغییر کرده:
http://www.idevcenter.com/links/upcoming

نویسنده: Meysam
تاریخ: ۱۳۸۹/۰۱/۱۰ ۲۰:۰۳:۳۷

تو این CheckForIllegalCrossThreadCalls نیست؟ اینکار استفاده از Thread بسیار کم میکنه!

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۱/۱۰ ۲۳:۳۴:۲۳

چرا، Dispatcher.CheckAccess دارد.

عنوان: **دو تنظیم ضروری VS.NET جهت کار با WPF و Silverlight**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۵/۰۴ ۲۲:۰۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: WPF

تنظیم اول: تغییر نحوه‌ی نمایش پیش فرض فایل‌های XAML

اگر فایل XAML شما اندکی حجیم شود نمایش آن در VS.NET کمی طولانی خواهد شد و حالت پیش فرض نمایش در VS.NET هم split view mode است (نمایش XAML و پیش نمایش آن با هم). این مورد هم پس از مدتی تبدیل به عذاب می‌شود. برای رفع آن می‌توان حالت پیش فرض نمایش یک فایل XAML را به XAML View تنها تغییر داد.
برای این منظور به منوی Tools ، گزینه‌ی Options و سپس قسمت تنظیمات Text editor مراجعه کنید. در اینجا در قسمت XAML ، گزینه‌ی Miscellaneous را انتخاب کرده و سپس "Always open documents in full XAML view" را تیک بزنید.

حتی ممکن است این مورد هم رضایت بخش نباشد. در این حالت می‌توان ویرایشگر پیش فرض را کلا تغییر داد. Design tab را در پایین صفحه از دست می‌دهیم اما هنوز intellisense کار می‌کند و اگر نیاز به designer بود فقط کافی است کلیک راست کرده و گزینه‌ی View designer را انتخاب کرد:
روی یک فایل XAML دلخواه کلیک راست کرده و گزینه‌ی Open with را انتخاب کنید. سپس "Source Code (Text) Editor" را انتخاب کرده و روی دکمه‌ی Set as Default کلیک کنید. تمام!
هر چند Blend این مشکلات را ندارد و با فایل‌های حجیم XAML به خوبی کاری می‌کند.

تنظیم دوم: تغییر نحوه‌ی نمایش مشکلات ناشی از Binding

عموما اگر مشکلاتی در حین عملیات Binding در WPF یا Silverlight وجود داشته باشند، خطاها در Debugger Output Window نمایش داده می‌شوند. حالت پیش فرض هم فقط روی Error تنظیم شده است به این معنا که warning ها را مشاهده نخواهید کرد.
برای تغییر این مورد باید به صورت زیر عمل کرد:
به منوی Tools ، گزینه‌ی Options و سپس قسمت تنظیمات Debugging مراجعه کنید. گزینه‌ی Output Window -> WPF Trace Settings را انتخاب نمائید. سپس در اینجا قسمت WPF trace settings را یافته و مقدار پیش فرض Data binding را که به Error تنظیم شده است، به Warning تنظیم نمائید.

Options
Environment
Projects and Solutions
Source Control
Text Editor
Debugging
General
Edit and Continue
Just-In-Time
Native
Output Window
Symbols
IntelliTrace
Performance Tools
Database Tools
F# Tools
HTML Designer
Office Tools
Test Tools
Text Templating
Windows Forms Designer
General Output Settings
All debug output On
Exception Messages On
Module Load Messages Off
Module Symbol Search Messages Off
Module Unload Messages Off
Process Exit Messages Off
Step Filtering Messages Off
Thread Exit Messages Off
WPF Trace Settings
Animation Off
Data Binding Warning
Dependency Properties Off
Documents Off
Freezable Off
HWND hosting Off
Data Binding
Sets the trace level for Data Binding in WPF
OK
Cancel

عنوان: **معرفی WPF Extended toolkit**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۶/۲۰ ۲۰:۵۶:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: WPF

یکی از نکات جالبی که در مورد Silverlight وجود دارد این است که هر چند تنها قسمتی از WPF را به ارث برده (برای اینکه حجم افزونه‌ی آن قابل قبول باشد)، اما بیشتر از خود WPF مورد توجه مایکروسافت است! شاید یک دلیل آن استفاده از Silverlight در Windows phone 7 باشد. به عبارتی اگر برنامه نویس Silverlight هستید، هم اکنون برنامه نویس Windows phone 7 نیز می‌باشید.

این توجه بیشتر در Silverlight toolkit کاملا مشخص است. Silverlight toolkit از یک سری ابزار و کامپوننت برای توسعه‌ی ساده‌تر برنامه‌های Silverlight به صورت سورس باز و تهیه شده توسط مایکروسافت، تشکیل شده است. حجم WPF toolkit که آن هم توسط مایکروسافت به صورت سورس باز ارائه و به روز می‌شود حدود 2 مگابایت است؛ اما حجم Silverlight toolkit حدود 18 مگابایت می‌باشد! بسیاری از کنترل‌ها و امکانات Silverlight toolkit را در WPF نمی‌توانید پیدا کنید مانند DataForm ، ChildWindow ، BusyIndicator و غیره. نمونه‌ی دیگر این توجه WCF RIA Services است. هدفگیری اصلی این مورد نیز Silverlight است و نه WPF (که از آن در Visual studio LightSwitch هم استفاده کرده‌اند).

اخیرا یک گروه خیّر کار تبدیل و انتقال کنترل‌های Silverlight toolkit به WPF toolkit را شروع کرده است که حاصل آن از آدرس ذیل قابل دریافت است: (این هم یکی از مزیت‌های پروژه‌های سورس باز است)

WPF Extended toolkit

نظرات خوانندگان

نویسنده: مهدی پایروند
تاریخ: ۱۳۸۹/۰۶/۲۲ ۱۵:۰۲:۳۳

نکته جالبی بود، مخصوصا حجم توکلیت‌ها

نویسنده: رضا
تاریخ: ۱۳۹۱/۰۵/۲۱ ۱۳:۲۰

میخواستم بدونم بهترین کنترل DatePicker فارسی برای WPF چی هستش؟ من خودم یه کنترل با 3 تا TextBox درست کردم ولی Binding اش رو نتونستم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۱ ۱۳:۲۷

(+)
یا
(+) هم نسخه WPF دارد

عنوان: **WPF و قالب‌هایی جهت کنترل DataGrid**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۷/۰۸ ۱۱:۲۷:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: WPF

در مورد معرفی WPF Extended toolkit چندی قبل مطلبی منتشر شد . در ادامه این بی مهری‌ها (!) می‌توان به عدم به روز رسانی قالب‌های ارائه شده برای WPF اشاره کرد. در WPF4 ، کنترل DataGrid از WPF toolkit به مجموعه‌ی کنترل‌های اصلی WPF منتقل شده است، اما قالب‌های منتشر شده‌ی آن جهت لحاظ کردن این مورد به روز نشده‌اند. یعنی اگر برای مثال یکی از قالب‌های موجود را به برنامه خود اعمال کنید و سپس DataGrid را بر روی فرم قرار دهید، وصله‌ی ناهماهنگی را مشاهده خواهید نمود. این مشکلات در Silverlight وجود ندارند و قالب‌های ارائه شده‌ی برای آن به روز بوده و همچنین روز به روز هم تعدادشان بیشتر می‌شوند.

اما باز هم نمی‌توان ایراد گرفت چون کار ارائه شده سورس باز است. به عبارتی اگر مایکروسافت این قالب‌ها را به روز نکرده، خوب، لطفا خود شما وقت بگذارید و این کار را انجام داده و سپس یک patch ارائه دهید. ایرادی دارد؟!

برای این منظور پروژه‌ای در سایت CodePlex ایجاد شده است و تنها به پوشش دات نت سه و نیم و دیتاگرید متعلق به WPF Toolkit پرداخته است :

WPF DataGrid Themes from Silverlight

اگر علاقمند باشید که از دیتاگرید بومی دات نت 4 استفاده کنید می‌توانید از این patch استفاده کنید.

عنوان: **خلاصه‌ای از مبحث نمایش اطلاعات hierarchical در WPF**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۷/۲۱ ۲۰:۳۹:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: WPF

در این مطلب خلاصه‌ای را در مورد نحوه‌ی نمایش اطلاعات hierarchical (سلسله مراتبی، درختی) در WPF به همراه یک سری لینک مرتبط ملاحظه خواهید نمود.

کلاس زیر را در نظر بگیرید:

```
using System.Collections.Generic;

namespace WpfTests.Hierarchy.Raw.Model
{
   public class Person
   {
       private readonly List<Person> _children = new List<Person>();
       public IList<Person> Children
       {
           get { return _children; }
       }

       public string Name { get; set; }
   }
}
```

و همچنین یک ObservableCollection ساخته شده از آن‌را با مقدار دهی اولیه:

```
using System.Collections.ObjectModel;

namespace WpfTests.Hierarchy.Raw.Model
{
   public class People : ObservableCollection<Person>
   {
       public People()
       {
           this.Add(
           new Person
           {
               Name = "P1",
               Children =
               {
                   new Person
                   {
                       Name="P2",
                       Children=
                       {
                           new Person
                           {
                               Name="P3",
                               Children=
                               {
                                   new Person
                                   {
                                       Name="P4",
                                   }
                               }
                           }
                       }
                   }
               }
           }
           );
       }
   }
}
```

قصد داریم این اطلاعات را در یک TreeView نمایش دهیم.

روش صحیح Binding این نوع اطلاعات در WPF استفاده از HierarchicalDataTemplate است به صورت زیر :

```
<TreeView ItemsSource="{Binding People}">
    <TreeView.ItemTemplate>
         <HierarchicalDataTemplate ItemsSource="{Binding Children}">
               <TextBlock Text="{Binding Name}" />
          </HierarchicalDataTemplate>
     </TreeView.ItemTemplate>
</TreeView>
```

یک سری منبع آموزشی برای آشنایی بیشتر با HierarchicalDataTemplate

Hierarchical Databinding in WPF

Binding WPF Treeview and Objects

A TreeView, a HierarchicalDataTemplate, and a 2D collection

Non-recursive WPF TreeView controls

همچنین هنگام کار با بانک‌های اطلاعاتی:

- یک Extension method عالی قابل استفاده در LINQ to SQL و همچنین Entity framework به نام AsHierarchy
- مثالی دیگر از کاربرد LINQ to SQL برای این منظور
- و یا مثالی از ADO.NET و DataSets و مثالی دیگر

نظرات خوانندگان

نویسنده: علی اقدم
تاریخ: ۲۳:۴۸:۰۷ ۱۳۸۹/۰۷/۲۱

آقای نصیری واقعا مفید بود

مخصوصا لینک ها
خیلی ممنون

نویسنده: arya
تاریخ: ۱۵:۲۹:۵۷ ۱۳۸۹/۰۷/۲۹

بسیار ممنون بابت پست های مفیدتون.

نویسنده: Meysam Javadi
تاریخ: ۱۴:۳۸:۴۷ ۱۳۹۰/۰۱/۲۹

http://www.codeproject.com/KB/WPF/CustomTreeViewLayout.aspx

عنوان: **WPF4 و ویندوز 7 : به خاطر سپاری لیست آخرین فایل‌های گشوده شده توسط برنامه**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۹/۲۷ ۱۸:۲۳:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: WPF

اگر به برنامه‌های جدید نوشته شده برای ویندوز 7 دقت کنیم، از یک سری امکانات مخصوص آن جهت بهبود دسترسی پذیری به قابلیت‌هایی که ارائه می‌دهند، استفاده شده است. برای مثال برنامه‌ی OneNote مجموعه‌ی آفیس را در نظر بگیرید. اگر بر روی آیکون آن در نوار وظیفه‌ی ویندوز کلیک راست کنیم، لیست آخرین فایل‌های گشوده شده توسط آن مشخص است و با کلیک بر روی هر کدام، به سادگی می‌توان این فایل را گشود. یک چنین قابلیتی در منوی آغازین ویندوز نیز تعبیه شده است (شکل‌های زیر):

خبر خوب اینکه برای اضافه کردن این قابلیت به برنامه‌های WPF4 نیازی به کد نویسی نیست و این موارد که تحت عنوان استفاده از Jump list ویندوز 7 تعریف شده‌اند، با کمی دستکاری فایل App.Xaml برنامه، فعال می‌گردند:

```
<Application x:Class="WpfApplication1.App"
            xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
            xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
            StartupUri="MainWindow.xaml">
```

```
    <Application.Resources>
    </Application.Resources>
    <JumpList.JumpList>
        <JumpList ShowRecentCategory="True" />
    </JumpList.JumpList>
</Application>
```

همین! از این پس هر فایلی که توسط برنامه‌ی شما با استفاده از common file dialog boxes باز شود به صورت خودکار به لیست مذکور اضافه می‌گردد (بدیهی است Jump lists جزو ویژگی‌های ویندوز 7 است و در سایر سیستم عامل‌ها ندید گرفته خواهد شد).

سؤال: من اینکار را انجام دادم ولی کار نمی‌کنه!؟

پاسخ: بله. کار نمی‌کنه! این قابلیت تنها زمانی فعال خواهد شد که علاوه بر نکته‌ی فوق، پسوند فایل یا فایل‌هایی نیز به برنامه‌ی شما منتسب شده باشد. این انتساب‌ها مطلب جدیدی نیست و در تمام برنامه‌های ویندوزی باید توسط بکارگیری API ویندوز مدیریت شود. قطعه کد زیر اینکار را انجام خواهد داد:

```
using System;
using System.Runtime.InteropServices;
using Microsoft.Win32;

namespace Common.Files
{
  //from : http://www.devx.com/vb2themax/Tip/19554?type=kbArticle&trk=MSCP
  public class FileAssociation
  {
      const int ShcneAssocchanged = 0x8000000;
      const int ShcnfIdlist = 0;

      public static void CreateFileAssociation(
          string extension,
          string className,
          string description,
           string exeProgram)
      {
          // ensure that there is a leading dot
          if (extension.Substring(0, 1) != ".")
              extension = string.Format(".{0}", extension);

          try
          {
              if (IsAssociated(extension)) return;

              // create a value for this key that contains the classname
              using (var key1 = Registry.ClassesRoot.CreateSubKey(extension))
              {
                  if (key1 != null)
                  {
                      key1.SetValue("", className);
                      // create a new key for the Class name
                      using (var key2 = Registry.ClassesRoot.CreateSubKey(className))
                      {
                          if (key2 != null)
                          {
                              key2.SetValue("", description);
                              // associate the program to open the files with this extension
                              using (var key3 =
Registry.ClassesRoot.CreateSubKey(string.Format(@"{0}\Shell\Open\Command", className)))
                              {
                                  if (key3 != null) key3.SetValue("", string.Format(@"{0} ""%1""",
exeProgram));
                              }
                          }
                      }
                  }
              }

              // notify Windows that file associations have changed
              SHChangeNotify(ShcneAssocchanged, ShcnfIdlist, 0, 0);
          }
          catch (Exception ex)
          {
```

```
            //todo: log ...
        }
    }

    // Return true if extension already associated in registry
    public static bool IsAssociated(string extension)
    {
        return (Registry.ClassesRoot.OpenSubKey(extension, false) != null);
    }

    [DllImport("shell32.dll")]
    public static extern void SHChangeNotify(int wEventId, int uFlags, int dwItem1, int dwItem2);
  }
}
```

و مثالی از نحوه‌ی استفاده از آن:

```
private static void createFileAssociation()
{
   var appPath = Assembly.GetExecutingAssembly().Location;
    FileAssociation.CreateFileAssociation(".xyz", "xyz", "xyz File",
              appPath
              );
}
```

لازم به ذکر است که این کد در ویندوز 7 فقط با دسترسی مدیریتی قابل اجرا است (کلیک راست و اجرا به عنوان ادمین) و در سایر حالات با خطای Access is denied متوقف خواهد شد. به همین جهت بهتر است برنامه‌ی نصاب مورد استفاده این نوع انتسابات را مدیریت کند؛ زیرا اکثر آن‌ها با دسترسی مدیریتی است که مجوز نصب را به کاربر جاری خواهند داد. اگر از فناوری Click once استفاده می‌کنید به این مقاله و اگر برای مثال از NSIS کمک می‌گیرید به این مطلب مراجعه نمائید.

سؤال: من این کارها را هم انجام دادم. الان به چه صورت از آن استفاده کنم؟

زمانیکه کاربری بر روی یکی از این فایل‌های ذکر شده در لیست آخرین فایل‌های گشوده شده توسط برنامه کلیک کند، آدرس این فایل به صورت یک آرگومان به برنامه ارسال خواهد شد. برای مدیریت آن در WPF باید به فایل App.Xaml.cs مراجعه کرده و چند سطر زیر را به آن افزود:

```
public partial class App
  {
      public App()
      {
          this.Startup += appStartup;
      }

      void appStartup(object sender, StartupEventArgs e)
      {
          if (e.Args.Any())
          {
              this.Properties["StartupFileName"] = e.Args[0];
          }
      }
  //...
```

در این کد، e.Args حاوی مسیر فایل انتخابی است. برای مثال در اینجا مقدار آن به خاصیت StartupFileName انتساب داده شده است. این خاصیت در برنامه‌های WPF به صورت یک خاصیت عمومی تعریف شده است و در سراسر برنامه (مثلا در رخداد آغاز فرم اصلی آن یا هر جای دیگری) به صورت زیر قابل دسترسی است:

```
var startupFileName = Application.Current.Properties["StartupFileName"];
```

سؤال: برنامه‌ی من از OpenFileDialog برای گشودن فایل‌ها استفاده نمی‌کند. آیا راه دیگری برای افزودن مسیرهای باز شده به Jump lists ویندوز 7 وجود دارد؟

پاسخ: بله. همانطور که می‌دانید عناصر XAML با اشیاء دات نت تناظر یک به یک دارند. به این معنا که JumpList تعریف شده در ابتدای این مطلب در فایل App.XAML ، دقیقا معادل کلاسی به همین نام در دات نت فریم ورک است (تعریف شده در فضای نام System.Windows.Shell) و با کد نویسی نیز قابل دسترسی و مدیریت است. برای مثال:

```
var jumpList = JumpList.GetJumpList(App.Current);
var jumpPath = new JumpPath();
jumpPath.Path = "some path goes here....";
// If the CustomCategory property is null
// or Empty, the item is added to the Tasks category
jumpPath.CustomCategory = "Files";
JumpList.AddToRecentCategory(jumpPath);
jumpList.Apply();
```

به همین ترتیب، JumpPath ذکر شده در کدهای فوق، در کدهای XAML نیز قابل تعریف است:

```
<Application x:Class="Win7Wpf4.App"
           xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
           xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
           StartupUri="MainWindow.xaml">
  <Application.Resources>
  </Application.Resources>
  <JumpList.JumpList>
      <JumpList ShowRecentCategory="True">
          <JumpPath
               CustomCategory="Files"
               Path="Some path goes here..."
              />
      </JumpList>
  </JumpList.JumpList>
</Application>
```

عنوان: **نمایش یک فایل PDF در WPF ، WinForms و سیلورلایت**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۴/۲۰ ۲۲:۵۴:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: WPF

شاید PDF را بشود تنها فرمت گزارشگیری دانست که همه‌جا و در تمام سیستم عامل‌ها پشتیبانی می‌شود. از ویندوز تا لینوکس از وب تا WPF تا سیلورلایت تا همه جا و از همه مهم‌تر اینکه خروجی آن دقیقا همان چیزی است که کاربر نهایی می‌خواهد: من می‌خوام اون چیزی رو که می‌بینم، دقیقا همان را، بدون کم و کاست و با همان صفحه بندی، بتوانم چاپ کنم.
برای تولید PDF می‌شود از کتابخانه‌ی iTextSharp استفاده کرد اما برای نمایش آن حداقل در ویندوز بهترین راه حل استفاده از COM Components شرکت Adobe است که به همراه برنامه رایگان Adobe PDF reader ارائه می‌شود. در ادامه نحوه‌ی استفاده از این Active-X را بررسی خواهیم کرد.

نمایش PDF در WPF

در تمام حالت‌ها هدف این است که به نحوی به اکتیوایکس شرکت Adobe دسترسی پیدا کنیم؛ یا با اضافه کردن آن به پروژه یا استفاده از امکانات یکپارچه مرورگرها. در WPF از زمان ارائه سرویس پک یک دات نت سه و نیم (به بعد)، کنترل مرورگر وب هم به جمع کنترل‌های قابل استفاده در آن اضافه شده است. در اینجا به سادگی چند سطر زیر می‌شود یک فایل PDF را در WPF نمایش داد:

```
<Window x:Class="WpfAppTests.MainWindow"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="Pdf Report" Height="495" WindowState="Maximized"
       WindowStartupLocation="CenterScreen" Width="703">
   <Grid>
       <WebBrowser x:Name="WebBrowser1"/>
   </Grid>
</Window>
```

و بعد هم در کدهای برنامه تنها کافی است که مقدار Source کنترل WebBrowser را مقدار دهی کرد:

```
WebBrowser1.Source = new Uri(PdfFilePath);
```

نمایش PDF در WinForms

اکتیوایکس نمایش دهنده PDF شرکت Adobe اساسا در فایل ذیل قرار گرفته است:

```
C:\Program Files\Common Files\Adobe\Acrobat\ActiveX\AcroPDF.dll
```

بنابراین برای استفاده از آن در یک برنامه‌ی WinForms باید مراحل ذیل طی شود:
الف) در VS.NET از طریق منوی Tools گزینه‌ی Choose toolbox items ، برگه‌ی Com components را انتخاب کنید.
ب) سپس گزینه‌ی Adobe PDF reader که به همان مسیر dll فوق اشاره می‌کند را انتخاب نمائید و بر روی دکمه‌ی OK کلیک کنید.
ج) اکنون این کنترل جدید را بر روی فرم برنامه قرار دهید. به صورت خودکار COMReference های متناظر به پروژه اضافه می‌شوند.

اکنون نحوه‌ی استفاده از این شیء COM به همراه آزاد سازی منابع مرتبط به شرح زیر خواهند بود:

```
using System.Windows.Forms;

namespace WindowsFormsAppTests
```

```
{
   public partial class Form1 : Form
   {
       public Form1()
       {
           InitializeComponent();
           this.Load += Form1_Load;
           this.FormClosing += Form1_FormClosing;
       }

       void Form1_FormClosing(object sender, FormClosingEventArgs e)
       {
           axAcroPDF1.Dispose();
       }

       void Form1_Load(object sender, System.EventArgs e)
       {
           axAcroPDF1.LoadFile(PdfFilePath);
           axAcroPDF1.setShowToolbar(true);
           axAcroPDF1.Show();
       }
   }
}
```

نمایش PDF در Silverlight

در Silverlight هم از نسخه‌ی 4 به بعد کنترل WebBrowser همانند آنچه که در WPF موجود است، اضافه شده است؛ اما این کنترل فقط در حالت اجرای در خارج از مرورگر برنامه Silverlight در دسترس می‌باشد. بنابراین روش دیگری را باید انتخاب کرد. این روش بر اساس تعامل سیلورلایت با کدهای HTML صفحه کار می‌کند. یک IFrame مخفی را در صفحه بالای شیء مرتبط با سیلورلایت قرار خواهیم داد. سپس در سیلورلایت Src این IFrame را به مسیر فایل PDF تنظیم می‌کنیم و همین. به این ترتیب فایل PDF نمایش داده می‌شود.

این IFrame به صورت زیر در همان صفحه‌ی aspx ایی که object مرتبط با Silverlight نمایش داده می‌شود قرار می‌گیرد:

```
<iframe id="pdfFrame" style="visibility:hidden; position:absolute"><b>No Content</b></iframe>
<div id="silverlightControlHost">
```

سپس در کدهای سیلورلایت، ابتدا این IFrame یافت شده:

```
var iFrame = HtmlPage.Document.GetElementById("pdfFrame");
```

در ادامه بر اساس اطلاعات مکانی یک Grid ساده به نام pdfHost که در صفحه قرار گرفته، این iFrame بالاتر از سطح Grid (بر اساس z-index تنظیم شده) نمایش داده می‌شود:

```
var gt = pdfHost.TransformToVisual(Application.Current.RootVisual);
var offset = gt.Transform(new Point(0, 0));
var controlLeft = (int)offset.X;
var controlTop = (int)offset.Y;
iFrame.SetStyleAttribute("left", string.Format("{0}px", controlLeft));
iFrame.SetStyleAttribute("top", string.Format("{0}px", controlTop));
iFrame.SetStyleAttribute("visibility", "visible");
iFrame.SetStyleAttribute("height", string.Format("{0}px", pdfHost.ActualHeight));
iFrame.SetStyleAttribute("width", string.Format("{0}px", pdfHost.ActualWidth));
iFrame.SetStyleAttribute("z-index", "1000");
```

و در آخر نام فایلی را که می‌خواهیم مشاهده کنیم به یک صفحه‌ی aspx در همان سایت ارسال می‌کنیم:

```
iFrame.SetProperty("src", "ShowPdf.aspx?file=" + fileName);
```

کدهای این صفحه در حد یک Response.Redirect ساده برای نمایش دادن فایل pdf در مرورگر کافی هستند. در کل در اینجا

سیلورلایت تنها نقش انتخاب فایل را به عهده دارد و کار اصلی را خود مرورگر انجام می‌دهد.

نظرات خوانندگان

نویسنده: Yari AliReza 2010
تاریخ: ۱۳۹۰/۰۴/۲۱ ۰۹:۳۲:۲۲

خیلی ممنون که تجربیات خود را در اختیار بقیه می گذارید.

نویسنده: Naeim Rezaeian
تاریخ: ۱۳۹۰/۰۶/۱۲ ۰۰:۲۷:۰۲

دست شما درد نکنه فقط میشه بگید که چطوری میشه همه تولبار و اون منوی که روی فایل میاد رو غیر فعال کرد .با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۶/۱۲ ۰۸:۲۵:۴۵

در حالت windows forms ، فقط کافی است بنویسید: axAcroPDF1.setShowToolbar(false)
در دو حالت دیگر که از مرورگر استفاده می‌شود، اینکار بی‌فایده است چون فایل مورد نظر از کش مرورگر قابل استخراج است.
اما اگر نیاز به حداقل کنترل وجود داشت باید تگ object را به صورت زیر درست کرد (کمی باید html نویسی کرد):

```
object type="application/pdf" data="file1.pdf#navpanes=0&scrollbar=0 &toolbar=0" width="500"
"height="650
```

عنوان: **چند نکته در مورد WPF MediaElement و ویندوز XP**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۷/۰۷ ۱۱:۴۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: WPF

WPF MediaElement
به صورت پیش فرض در ویندوز XP کار نمی‌کند؛ مگر اینکه حتما آخرین نگارش موجود Windows Media Player بر روی سیستم نصب شده باشد و حداقل نیاز به نگارش 10 به بعد را دارد.
اگر این نگارش نصب نباشد یا هر خطای دیگری رخ دهد، آن‌را می‌توان از طریق روال رویداد گردان MediaFailed بدست آورد.
اگر نگارش بتای مدیاپلیر 11 بر روی سیستم نصب باشد، با پیغام نه چندان آشنای "insufficient memory" مواجه خواهید شد و اهمیتی هم ندارد که سیستم در حال حاضر به چه میزان حافظه‌ی مهیا دسترسی دارد.
و کلا هر آنچه را که Windows Media Player بتواند پخش کند، WPF MediaElement نیز قادر به پخش آن‌ها خواهد بود. برای فرمت‌های ناشناخته و جدید باید Codec مخصوص آن‌ها در سیستم نصب شده باشد.

راه حل بهتر، استفاده از پروژه‌ی دیگری است به نام WPF Media Kit . این پروژه، هر آنچه را که بتوان توسط برنامه GraphEdit پخش کرد، می‌تواند نمایش دهد.

استفاده از WPF MediaElement به کمک الگوی MVVM درد بزرگی است؛ چون آنچنان از Binding و Commanding پشتیبانی نمی‌کند.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۷/۰۸ ۱۱:۴۱:۳۱

یک نکته‌ی دیگر:

ممکن هست جهت پخش فایل‌های mp4 ، از یک سری codec استفاده کرده باشید. مثلا: Haali's Media Splitter و FFDShow

تا زمانیکه در media player ویندوز در پاسخ به سؤال «فایل‌های mp4 را هم پخش کنم یا نه؟»، گزینه‌ی به خاطر سپاری پاسخ را تیک نزده باشید، در WPF Media Element با خطای زیر مواجه خواهید شد:

Media file download failed.

Exception from HRESULT: 0xC00D0FEA

عنوان: **تعیین Fallback font برای قلم‌های فارسی در WPF**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۱/۲۶ ۱۷:۵۹:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: WPF

اکثر قلم‌های فارسی، فاقد تعاریف مرتبط با حروف انگلیسی هستند. البته عموم کاربران متوجه این امر نمی‌شوند چون ویندوز دو مفهوم Font Fallback و Font Linking را جهت پوشش glyph های تعریف نشده، در پشت صحنه اعمال خواهد کرد. جزئیات بیشتر در اینجا: (^ و ^)

به صورت خلاصه کار Font Fallback در ویندوز جایگزینی خودکار قلم مورد استفاده است؛ تحت شرایط زیر:
- فونت تعریف شده در برنامه، در سیستم کاربر وجود نداشته باشد.
- تعاریف Glyphهای بکارگرفته شده در متن جاری، در قلم انتخابی وجود نداشته باشند.

در WPF این مساله کاملا قابل کنترل است. قلمی که به صورت خودکار به عنوان جایگزین مطرح می‌شود در قلمی به نام "Global User Interface" تعریف شده است. تعاریف این قلم ترکیبی هم در فایلی به نام GlobalUserInterface.CompositeFont در پوشه فونت‌های سیستم موجود است (برای مثال، مسیر c:\windows\fonts حاوی این فایل متنی است).
اگر این فایل XML را با یک ادیتور متنی باز کنید، مشاهده خواهید کرد که بازه‌های مختلف کاراکترهای یونیکد، به فونت‌های پیش فرضی نگاشت شده‌اند. بنابراین اگر این سؤال وجود دارد که در متن مخلوط فارسی و انگلیسی من، فونت پیش فرض حروف انگلیسی از کجا تامین و مشخص می‌شود، پاسخ را در این فایل می‌توانید مشاهده کنید.

روش دیگری هم برای تعیین Fallback font در WPF وجود دارد. یک مثال:

```
<Window x:Class="WpfFontTest.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <TextBlock
            Text="با هم English نمایش مخلوطی از متن فارسی و متن"
            Margin="7"
            FontFamily="Fonts/BNazanin.ttf#B Nazanin, Comic Sans Ms"
            FontSize="25"
            FlowDirection="RightToLeft"
            VerticalAlignment="Top" HorizontalAlignment="Center"  />
    </Grid>
</Window>
```

در این مثال فونت B Nazanin در برنامه قرار داده شده است (embedded font). همچنین در کنار آن پس از علامت کاما، Fallback font مشخص است. به این معنا که تاجایی که میسر است لطفا از فونت B Nazanin برای نمایش متن مورد نظر استفاده شود؛ اگر نشد از قلم Comic Sans Ms استفاده گردد. قلم B Nazanin حاوی تعاریف حروف انگلیسی نیست. بنابراین WPF جهت نمایش آن‌ها از فونت دوم معرفی شده کمک می‌گیرد. توضیحات بیشتر در اینجا: (^)

نظرات خوانندگان

نویسنده: حسین
تاریخ: ۱۳۹۱/۰۸/۰۵ ۱۳:۱۳

آقای نصیری من توی Embed کردن فونت به مشکل خوردم. میخوام از فونت Iranian Sans توی برنامم استفاده کنم. فایل فونت (irsans.ttf) رو در مسیر Resources/Fonts گذاشتم. حالا وقتی از روش‌های زیر برای تعیین فونت استفاده میکنم فونت کار نمیکنه.

```
<Setter Property="FontFamily" Value="./Resources/Fonts/#Irsans" />
```

```
<Setter Property="FontFamily" Value="Resources/Fonts/irsans.ttf" />
```

فونت هم همون فونتی هست که در کتابخانه PdfReport شما مورد استفاده قرار گرفته.
ممنون میشم راهنماییم کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۰۵ ۱۳:۳۳

به این صورت کار می‌کنه:

```
<TextBlock
            Text="نمایش مخلوطی از متن فارسی و متن English با هم"
            Margin="7"
            FontFamily="Fonts/irsans.ttf#Iranian Sans, Comic Sans Ms"
            FontSize="25"
            FlowDirection="RightToLeft"
            VerticalAlignment="Top" HorizontalAlignment="Center"  />
```

نویسنده: حسین
تاریخ: ۱۳۹۱/۱۰/۲۸ ۱۰:۵

سلام. برای تعیین Fallback font میشه مثلاً چندتا فونت دیگه هم گذاشت که به همین ترتیب اگر از هر کدوم نشد استفاده کرد از بعدی استفاده کنه یا فقط یک مورد رو میشه؟
و سوال دومم اینه که چرا باید بعد از علامت # Iranian Sans (در مورد همین مثال) رو نوشت و وقتی نمی‌نویسیم کار نمیکنه.
ممنون.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۲۸ ۱۲:۵۳

این موارد در آخرین لینکی که در متن مقاله است مفصل توضیح داده شده .

- بله. امکان تعریف چندین قلم وجود دارد.
- ذکر قسمت نام فایل اختیاری است (مثلا می‌شود به یک پوشه هم ارجاع داد). اما باید font family حتما ذکر شود.

عنوان: آموزش ایجاد برنامه های چند زبانه در WPF
نویسنده: امیر اویسی
تاریخ: ۱۳۹۱/۰۶/۰۹ ۱۵:۱۳
آدرس: www.dotnettips.info
گروه‌ها: WPF

با گسترش استفاده از کامپیوتر در بسیاری از امور روزمره انسان‌ها سازگار بودن برنامه‌ها با سلیقه کاربران به یکی از نیازهای اصلی برنامه‌های کامپیوتری تبدیل شده است. بدون شک زبان و فرهنگ یکی از مهم‌ترین عوامل در ایجاد ارتباط نزدیک بین برنامه و کاربر به شمار می‌رود و نقشی غیر قابل انکار در میزان موفقیت یک برنامه به عهده دارد. از این رو در این نوشته تلاش بر آن است تا یکی از ساده‌ترین و در عین حال کاراترین راه‌های ممکن برای ایجاد برنامه‌های چند زبانه با استفاده از تکنولوژی WPF آموزش داده شود.

مروری بر روش‌های موجود

همواره روش‌های مختلفی برای پیاده سازی یک ایده در دنیای نرم افزار وجود دارد که هر روش را می‌توان بر حسب نیاز مورد استفاده قرار داد. در برنامه‌های مبتنی بر WPF معمولا از دو روش عمده برای این منظور استفاده می‌شود:

1-استفاده از فایل‌های resx

در این روش که برای Win App نیز استفاده می‌شود، اطلاعات مورد نیاز برای هر زبان به شکل جدول هایی دارای کلید و مقدار در داخل یک فایل .resx نگهداری می‌شود و در زمان اجرای برنامه بر اساس انتخاب کاربر اطلاعات زبان مورد نظر از داخل فایل resx خوانده شده و نمایش داده می‌شود. یکی از ضعف هایی که این روش در عین ساده بودن دارد این است که همه اطلاعات مورد نیاز داخل assembly اصلی برنامه قرار می‌گیرد و امکان افزودن زبان‌های جدید بدون تغییر دادن برنامه اصلی ممکن نخواهد بود.

2-استفاده از فایل‌های csv که به فایل‌های dll تبدیل می‌شوند

در این روش با استفاده از ابزارهای موجود در کامپایلر WPF برای هر کنترل یک property به نام Uid ایجاد شده و مقدار دهی می‌شود. سپس با ابزار دیگری (که جزو ابزارهای کامپایلر محسوب نمی‌شود) از فایل csproj پروژه یک خروجی اکسل با فرمت csv ایجاد می‌شود که شامل Uidهای کنترل‌ها و مقادیر آن‌ها است. پس از ترجمه متون مورد نظر به زبان مقصد با کمک ابزار دیگری فایل اکسل مورد نظر به یک net assembly تبدیل می‌شود و داخل پوشه ای با نام culture استاندارد ذخیره می‌شود. (مثلا برای زبان فارسی نام پوشه fa-IR خواهد بود). زمانی که برنامه اجرا می‌شود بر اساس culture ای که در سیستم عامل انتخاب شده است و در صورتی که برای آن culture فایل dll ای موجود باشد، زبان مربوط به آن culture را load خواهد کرد. با وجود این که این روش مشکل روش قبلی را ندارد و بیشتر با ویژگی‌های WPF سازگار است اما پروسه ای طولانی برای انجام کارها دارد و به ازای هر تغییری باید کل مراحل هر بار تکرار شوند. همچنین مشکلاتی در نمایش برخی زبان‌ها (از جمله فارسی) در این روش مشاهده شده است.

روش سوم!

روش سوم اما کاملا بر پایه WPF و در اصطلاح WPF-Native می‌باشد. ایده از آنجا ناشی شده است که برای ایجاد skin در برنامه‌های WPF استفاده می‌شود. در ایجاد برنامه‌های Skin-Based به این شیوه عمل می‌شود که skinهای مورد نظر به صورت style هایی در داخل ResourceDictionary ها قرار می‌گیرند. سپس آن ResourceDictionary به شکل dll کامپایل می‌شود. در برنامه اصلی نیز همه کنترل‌ها style هایشان را به شکل dynamic resource از داخل یک ResourceDictionary مشخص شده load می‌کنند. حال کافی است برای تغییر skin فعلی، ResourceDictionary مورد نظر از dll مشخص load شود و ResourceDictionary ای که در حال حاضر در برنامه از آن استفاده می‌شود با ResourceDictionary ای که load شده جایگزین شود. کنترل‌ها مقادیر جدید را از ResourceDictionary جدید به شکل کاملا خودکار دریافت خواهند کرد.

به سادگی می‌توان از این روش برای تغییر زبان برنامه نیز استفاده کرد با این تفاوت که این بار، به جای Style ها، Stringهای زبان‌های مختلف را درون resourceها نگهداری خواهیم کرد.

یک مثال ساده

در این قسمت نحوه پیاده سازی این روش با ایجاد یک نمونه برنامه ساده که دارای دو زبان انگلیسی و فارسی خواهد بود آموزش

داده می‌شود.

ابتدا یک پروژه WPF Application در Visual Studio 2010 ایجاد کنید. در MainWindow سه کنترل Button قرار دهید و یک ComboBox که قرار است زبان‌های موجود را نمایش دهد و با انتخاب یک زبان، نوشته‌های درون Buttonها متناسب با آن تغییر خواهند کرد.

توجه داشته باشید که برای Buttonها نباید به صورت مستقیم مقداری به Content شان داده شود. زیرا مقدار مورد نظر از داخل ResourceDictionary که خواهیم ساخت به شکل dynamic گرفته خواهد شد. پس در این مرحله یک ResourceDictionary به پروژه اضافه کرده و در آن resource هایی به شکل string ایجاد می‌کنیم. هر resource دارای یک Key می‌باشد که بر اساس آن، Button مورد نظر، مقدار آن Resource را load خواهد کرد. فایل ResourceDictionary را Culture_en-US.xaml نامگذاری کنید و مقادیر مورد نظر را به آن اضافه نمایید.

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:system="clr-namespace:System;assembly=mscorlib">
    <system:String x:Key="button1">Hello!</system:String>
    <system:String x:Key="button2">How Are You?</system:String>
    <system:String x:Key="button3">Are You OK?</system:String>

</ResourceDictionary>
```

دقت کنید که namespace ای که کلاس string در آن قرار دارد به فایل xaml اضافه شده است و پیشوند system به آن نسبت داده شده است.

با افزودن یک ResourceDictionary به پروژه، آن ResourceDictionary به MergedDictionary کلاس App اضافه می‌شود. بنابراین فایل App.xaml به شکل زیر خواهد بود:

```
<Application x:Class="BeRMOoDA.WPF.LocalizationSample.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             StartupUri="MainWindow.xaml">
    <Application.Resources>

        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="Culture_en-US.xaml"/>
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
```

```
    </Application.Resources>
</Application>
```

برای اینکه بتوانیم محتوای Buttonهای موجود را به صورت داینامیک و در زمان اجرای برنامه، از داخل Resourceها بگیریم، از DynamicResource استفاده می‌کنیم.

```
<Button Content="{DynamicResource ResourceKey=button1}" />
<Button Content="{DynamicResource ResourceKey=button2}" />
<Button Content="{DynamicResource ResourceKey=button3}" />
```

بسیار خوب! اکنون باید شروع به ایجاد یک ResourceDictionary برای زبان فارسی کنیم و آن را به صورت یک فایل dll کامپایل نماییم.

برای این کار یک پروژه جدید در قسمت WPF از نوع User control ایجاد می‌کنیم و نام آن را Culture_fa-IR_Farsi قرار می‌دهیم. لطفا شیوه نامگذاری را رعایت کنید چرا که در ادامه به آن نیاز خواهیم داشت.

پس از ایجاد پروژه فایل UserControl1.xaml را از پروژه حذف کنید و یک ResourceDictionary با نام Culture_fa-IR.xaml اضافه کنید. محتوای آن را پاک کنید و محتوای فایل Culture_en-US.xaml را از پروژه قبلی به صورت کامل در فایل جدید کپی کنید. دو فایل باید ساختار کاملا یکسانی از نظر key برای Resourceهای موجود داشته باشند. حالا زمان ترجمه فرا رسیده است! رشته‌های دلخواه را ترجمه کنید و پروژه را build نمایید.

پس از ترجمه فایل Culture_fa-IR.xaml به شکل زیر خواهد بود:

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:system="clr-namespace:System;assembly=mscorlib">
    <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary Source="Culture_fa-IR_Farsi.xaml"/>
    </ResourceDictionary.MergedDictionaries>
    <system:String x:Key="button1">سلام!</system:String>
    <system:String x:Key="button2">حالت چطوره؟</system:String>
    <system:String x:Key="button3">خوبی؟</system:String>
</ResourceDictionary>
```

خروجی این پروژه یک فایل با نام Culture_fa-IR_Farsi.dll خواهد بود که حاوی یک ResourceDictionary برای زبان فارسی می‌باشد.

در ادامه میخواهیم راهکاری ارئه دهیم تا بتوان فایل‌های dll مربوط به زبان‌ها را در زمان اجرای برنامه اصلی، load کرده و نام زبان‌ها را در داخل ComboBox ای که داریم نشان دهیم. سپس با انتخاب هر زبان در ComboBox، محتوای Buttonها بر اساس زبان انتخاب شده تغییر کند.

برای سهولت کار، نام فایل‌ها را به گونه ای انتخاب کردیم که بتوانیم ساده‌تر به این هدف برسیم. نام هر فایل از سه بخش تشکیل شده است:

```
Culture_[standard culture notation]_[display name for this culture].dll
```

یعنی اگر فایل Culture_fa-IR_Farsi.dll را در نظر بگیریم، Culture نشان دهنده این است که این فایل مربوط به یک culture می‌باشد. fa-IR نمایش استاندارد culture برای کشور ایران و زبان فارسی است و Farsi هم مقداری است که می‌خواهیم در ComboBox برای این زبان نمایش داده شود.

پوشه ای با نام Languages در کنار فایل اجرایی برنامه اصلی ایجاد کنید و فایل Culture_fa-IR_Farsi.dll را درون آن کپی کنید. تصمیم داریم همه dllهای مربوط به زبان‌ها را داخل این پوشه قرار دهیم تا مدیریت آن‌ها ساده‌تر شود.

برای مدیریت بهتر فایل‌های مربوط به زبان‌ها یک کلاس با نام CultureAssemblyModel خواهیم ساخت که هر instance از آن نشانگر یک فایل زبان خواهد بود. یک کلاس با این نام به پروژه اضافه کنید و propertyهای زیر را در آن تعریف نمایید:

```
public class CultureAssemblyModel
    {
        //the text will be displayed to user as language name (like Farsi)
        public string DisplayText { get; set; }
        //name of .dll file (like Culture_fa-IR_Farsi.dll)
        public string Name { get; set; }
        //standar notation of this culture (like fa-IR)
        public string Culture { get; set; }
        //name of resource dictionary file inside the loaded .dll (like Culture_fa-IR.xaml)
        public string XamlFileName { get; set; }
    }
```

اکنون باید لیست cultureهای موجود را از داخل پوشه languages خوانده و نام آنها را در ComboBox نمایش دهیم.

برای خواندن لیست cultureهای موجود، لیستی از CultureAssmeblyModelها ایجاد کرده و با استفاده از متد LoadCultureAssmeblies، آن را پر می‌کنیم.

```
//will keep information about loaded assemblies
public List<CultureAssemblyModel> CultureAssemblies { get; set; }

//loads assmeblies in languages folder and adds their info to list
 void LoadCultureAssemblies()
 {
      //we should be sure that list is empty before adding info (do u want to add some cultures more
than one? of course u dont!)
      CultureAssemblies.Clear();
      //creating a directory represents applications directory\languages
      DirectoryInfo dir = new
DirectoryInfo(System.IO.Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location) +
"\\languages");
      //getting all .dll files in the language folder and its sub dirs. (who knows? maybe someone keeps
each culture file in a seperate folder!)
      var assemblies = dir.GetFiles("*.dll", SearchOption.AllDirectories);
      //for each found .dll we will create a model and set its properties and then add to list  for
(int i = 0; i < assemblies.Count(); i++)
      {

                string name = assemblies[i].Name;

                CultureAssemblyModel model = new CultureAssemblyModel() { DisplayText = name.Split('.',
'_')[2], Culture = name.Split('.', '_')[1], Name = name  , XamlFileName =name.Substring(0,
name.LastIndexOf(".")) + ".xaml" };
                CultureAssemblies.Add(model);
      }
 }
```

پس از دریافت اطلاعات cultureهای موجود، زمان نمایش آن‌ها در ComboBox است. این کار بسیار ساده است، تنها کافی است ItemsSource آن را با لیستی از CultureAssmeblyModelها که ساختیم، مقدار دهی کنیم.

```
comboboxLanguages.ItemsSource = CultureAssemblies;
```

البته لازم به ذکر است که برای نمایش فقط نام هر CultureAssemblyModel در ComboBox، باید ItemTemplate مناسبی برای ComboBox ایجاد کنیم. در مثال ما ItemTemplate به شکل زیر خواهد بود:

```
<ComboBox HorizontalAlignment="Left" Margin="10" VerticalAlignment="Top" MinWidth="100"
Name="comboboxLanguages">
            <ComboBox.ItemTemplate>
                <DataTemplate>
                    <Label Content="{Binding DisplayText}"/>
                </DataTemplate>
            </ComboBox.ItemTemplate>
</ComboBox>
```

توجه داشته باشید که با وجود اینکه فقط نام را در ComboBox نشان می‌دهیم، اما باز هم هر آیتم از ComboBox یک instance از نوع CultureAssemblyModel می‌باشد.

در مرحله بعد، قرار است متدی بنویسیم که اطلاعات زبان انتخاب شده را گرفته و با جابجایی ResourceDictionary ها، زبان برنامه را تغییر دهیم.
متدی با نام LoadCulture در کلاس App ایجاد می‌کنیم که یک CultureAssemblyModel به عنوان ورودی دریافت کرده و ResourceDictionary داخل آن را load می‌کند و آن را با ResourceDictionary فعلی موجود در App.xaml جابجا می‌نماید.
با این کار، Button هایی که قبلا مقدار Content خود را از Resourceهای موجود دریافت می‌کردند، اکنون از Resourceهای جابجا شده خواهند گرفت و به این ترتیب زبان انتخاب شده بر روی برنامه اعمال می‌شود.

```
//loads selected culture
 public void LoadCulture(CultureAssemblyModel culture)
 {
     //creating a FileInfo object represents .dll file of selected cultur
     FileInfo assemblyFile = new FileInfo("languages\\" + culture.Name);
     //loading .dll into memory as a .net assembly
     var assembly = Assembly.LoadFile(assemblyFile.FullName);
     //getting .dll file name
     var assemblyName = assemblyFile.Name.Substring(0, assemblyFile.Name.LastIndexOf("."));
     //creating string represents structure of a pack uri (something like this:
/{myassemblyname;component/myresourcefile.xaml}
     string packUri = string.Format(@"/{0};component/{1}", assemblyName, culture.XamlFileName);
     //creating a pack uri
     Uri uri = new Uri(packUri, UriKind.Relative);
     //now we have created a pack uri that represents a resource object in loaded assembly
     //and its time to load that as a resource dictionary (do u remember that we had resource
dictionary in culture assemblies? don't u?)
     var dic = Application.LoadComponent(uri) as ResourceDictionary;
     dic.Source = uri;
     //here we will remove current merged dictionaries in our resource dictionary and add recently-
loaded resource dictionary as e merged dictionary
     var mergedDics = this.Resources.MergedDictionaries;
     if (mergedDics.Count > 0)
          mergedDics.Clear();
     mergedDics.Add(dic);
 }
```

برای ارسال زبان انتخاب شده به این متد، باید رویداد SelectionChanged را برای ComboBox مدیریت کنیم:

```
void comboboxLanguages_SelectionChanged(object sender, SelectionChangedEventArgs e)
 {
     var selectedCulture = (CultureAssemblyModel)comboboxLanguages.SelectedItem;
     App app = Application.Current as App;
     app.LoadCulture(selectedCulture);
 }
```

کار انجام شد!
از مزیت‌های این روش می‌توان به WPF-Native بودن، سادگی در پیاده سازی، قابلیت load کردن هر زبان جدیدی در زمان اجرا بدون نیاز به کوچک‌ترین تغییر در برنامه اصلی و همچنین پشتیبانی کامل از نمایش زبان‌های مختلف از جمله فارسی اشاره کرد.

نظرات خوانندگان

نویسنده: رضا
تاریخ: ۱۳۹۱/۰۶/۰۹ ۱۵:۲۸

آیا استفاده زیاد از این Dynamic Resourceها مشکلی در Performance برنامه ایجاد نمیکند؟

نویسنده: امیر اویسی
تاریخ: ۱۳۹۱/۰۶/۰۹ ۱۸:۳۵

Dynamic Resourceها در مقایسه با Static Resourceها دارای performance کمتری هستند اما در مواردی که گرفتن مقدار از Resourceها در زمان اجرا انجام می‌گیرد، باید از Dynamic Resourceها استفاده کرد.
در کل تفاوت Performance در کاربردهای این چنین آنقدر نیست که موجب نگرانی باشد.

نویسنده: S.Roshan
تاریخ: ۱۳۹۱/۱۲/۰۵ ۲۱:۲۰

سلام. ممنون از مقاله مفیدتون.
امااااا....

```
if (mergedDics.Count > 0)   mergedDics.Clear();
```

متاسفانه این خط کد ، کل دیکشنریهای ادغامی رو پاک میکنه.
در صورتیکه ممکنه ما یه سری ریسورس دیگه مرتبط با بخشهای دیگه‌ی برنامه داشته باشیم، که نیازی به پاک کردنشون نباشه.
به نظرم باید این کد تغییر کنه ، فقط ریسورس مربوط به زبان برنامه رو پاک کنه.

فقط نمیدونم چه جوری ؟
بی زحمت کدشو بنویسید . ممنون

نویسنده: بهزاد دات نت
تاریخ: ۱۳۹۲/۱۰/۰۹ ۱۱:۴۵

با تشکر از آموزش خوبتون. میخواستم بدونم Direction صفحات برنامه رو چطور مدیریت کنم. مثلا برای انتخاب زبان فارسی راست به چپ و انگلیسی چپ به راست بشه به صورت خودکار؟ با سپاس

نویسنده: امیر اویسی
تاریخ: ۱۳۹۲/۱۲/۱۴ ۱۹:۴۹

سلام
با عرض پوزش به خاطر تاخیر زیاد در ارسال پاسخ باید عرض کنم که اجباری به حذف کلیه ResourceDictionaryها نیست. شما میتوانید با استفاده از متد ResourceDictionary.Remove یک ResourceDictionary به خصوص را با استفاده از Key آن از لیست MergedDictionaryها حذف کنید.

نویسنده: امیر اویسی
تاریخ: ۱۳۹۲/۱۲/۱۴ ۱۹:۵۸

برای همه تنظیماتی که نیاز دارید در زمان load شدن یک زبان خاص بر روی برنامه اعمال شود میتوانید در داخل فایل Resource آن زبان مقدار مورد نظر را با استفاده از یک کلید به عنوان Resource تعریف کنید و در برنامه خودتان مقدار مورد نیاز را از همان

Resource بخوانید.

مثلا با فرض اینکه میخواهیم با انتخاب زبان فارسی، برنامه ما راست به چپ شود، میتوانید یک Resource از نوع FrameworkElement.FlowDirection با کلید Direction در داخل ResourceDictionary زبان فارسی ایجاد کنید و مقدار مورد نظر را به آن اختصاص دهید. سپس در کنترل هایی که نیاز دارید راست به چپ شوند، مقدار FlowDirection آن‌ها را به صورت DynamicResource به همین Resource ای که تعریف کردید مقدار دهی کنید.

عنوان: آشنایی و استفاده از WCF Data Services در Visualstudio 2012
نویسنده: مجتبی کاویانی
تاریخ: ۱۳۹۱/۱۰/۱۳ ۱۷:۰
آدرس: www.dotnettips.info
گروه‌ها: WPF, Web Service, OData, WCF Data Services

مقدمه:

WCF Data Services جزئی از NET Framework . است که امکان ایجاد سرویس دهنده‌های با قرارداد OData را به روی وب یا Intranet با استفاده از REST مهیا می‌سازد. OData از داده هایی که با Url آدرس پذیر هستند استفاده می‌نماید. دسترسی و تغییر داده‌ها با استفاده از استاندارد HTTP و کلمات GET، PUT، POST و DELETE صورت می‌پذیرد. برای اینکه درک بهتری داشته باشید به یک مثال می‌پردازیم.

ایجاد یک برنامه سرویس دهنده WCF Data Service در VisualStudio 2012

یک ASP.NET Web Application با نام NorthwindService ایجاد نمایید و بر روی پروژه راست کلیک کنید و از منوی Add گزینه New Item را انتخاب نمایید از پنجره باز شده از دسته Data گزینه ADO.NET Entity Data Model را انتخاب و نام ان را Northwind بگذارید.

از پنجره باز شده Generate from Database را انتخاب و با انتخاب کانکشن از نوع Sql Server Compact 4 اتصال به فایل Northwind.sdf را انتخاب تا کلاس‌های لازم تولید شود.

برای تولید data service بر روی پروژه راست کلیک کنید و از منوی Add گزینه New Item را انتخاب نمایید از پنجره باز شده گزینه WCF Data Service را انتخاب و نام آن را Northwind.svc بگذارید. کد زیر خودکار تولید می‌شود

```
public class Northwind : DataService< /* TODO: put your data source class name here */ >
    {
        // This method is called only once to initialize service-wide policies.
        public static void InitializeService(DataServiceConfiguration config)
        {
            // TODO: set rules to indicate which entity sets and service operations are visible,
updatable, etc.
            // Examples:
            // config.SetEntitySetAccessRule("MyEntityset", EntitySetRights.AllRead);
            // config.SetServiceOperationAccessRule("MyServiceOperation", ServiceOperationRights.All);
            config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V3;
        }
    }
```

برای دسترسی به موجودیت‌های Northwind بجای عبارت put your data source نام مدل را تایپ کنید

```
public class Northwind : DataService<NorthwindEntities>
```

برای فعال کردن دسترسی به منابع data source متغیر config کلاس DataServiceConfiguration را بصورت زیر تنظیم نمایید. تابع SetEntitySetAccessRule با گرفتن نام موجودیت و نحوه دسترسی امکان استفاده از این موجودیت را با استفاده از WCF Data Service فراهم می‌نمایید. مثلا در زیر امکان دسترسی به موجودیت Orders را با امکان خواندن همه، نوشتن ادقامی و جایگزین فراهم نموده است.

```
config.SetEntitySetAccessRule("Orders", EntitySetRights.AllRead
     | EntitySetRights.WriteMerge
     | EntitySetRights.WriteReplace );
config.SetEntitySetAccessRule("Customers", EntitySetRights.AllRead);
```

اگر بخواهیم امکان خواندن همه موجودیت‌ها را فراهم کنیم از کد زیر می‌توانیم استفاده نمایید که * به معنای همه موجودیت‌های data model می‌باشد

```
config.SetEntitySetAccessRule("*", EntitySetRights.AllRead);
```

دسترسی به WCF Data Service بوسیله مرورگر وب

برای دسترسی به وب سرویس برنامه را اجرا نمایید تا آدرس http://localhost:8358/Northwind.svc مشخصات وب سرویس را نمایش دهد

```
<service xmlns="http://www.w3.org/2007/app" xmlns:atom="http://www.w3.org/2005/Atom"
xml:base="http://localhost:8358/Northwind.svc/">
<workspace>
<atom:title>Default</atom:title>
<collection href="Categories">
<atom:title>Categories</atom:title>
</collection>
<collection href="Customers">
<atom:title>Customers</atom:title>
</collection>
<collection href="Employees">
<atom:title>Employees</atom:title>
</collection>
<collection href="Order_Details">
<atom:title>Order_Details</atom:title>
</collection>
<collection href="Orders">
<atom:title>Orders</atom:title>
</collection>
<collection href="Products">
<atom:title>Products</atom:title>
</collection>
<collection href="Shippers">
<atom:title>Shippers</atom:title>
</collection>
<collection href="Suppliers">
<atom:title>Suppliers</atom:title>
</collection>
</workspace>
</service>
```

حال اگر آدرس را به http://localhost:8358/Northwind.svc/ Products وارد نمایید لیست کالاها بصورت Atom xml قابل دسترس می‌باشد.

ایجاد یک برنامه گیرنده WCF Data Service در Visual Studio 2012

بر روی Solution پروژه جاری راست کلیک و از منوی Add گزینه New Project را انتخاب و یک پروژه از نوع WPF Application با نام NorthwindClient ایجاد نمایید.

در پنجره MainWindow مانند کد زیر از یک Combobox و DataGrid برای نمایش اطلاعات استفاده نمایید

```
<Window x:Class="MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Northwind Orders" Height="335" Width="425"
        Name="OrdersWindow" Loaded="Window1_Loaded">
    <Grid Name="orderItemsGrid">
        <ComboBox DisplayMemberPath="Order_ID" ItemsSource="{Binding}"
                  IsSynchronizedWithCurrentItem="true"
                  Height="23" Margin="92,12,198,0" Name="comboBoxOrder" VerticalAlignment="Top"/>
        <DataGrid ItemsSource="{Binding Path=Order_Details}"
                  CanUserAddRows="False" CanUserDeleteRows="False"
                  Name="orderItemsDataGrid" Margin="34,46,34,50"
                  AutoGenerateColumns="False">
            <DataGrid.Columns>
                <DataGridTextColumn  Header="Product" Binding="{Binding Product_ID, Mode=OneWay}" />
                <DataGridTextColumn  Header="Quantity" Binding="{Binding Quantity, Mode=TwoWay}" />
                <DataGridTextColumn  Header="Price" Binding="{Binding UnitPrice, Mode=TwoWay}" />
                <DataGridTextColumn  Header="Discount" Binding="{Binding Discount, Mode=TwoWay}" />
            </DataGrid.Columns>
        </DataGrid>
        <Label Height="28" Margin="34,12,0,0" Name="orderLabel" VerticalAlignment="Top"
               HorizontalAlignment="Left" Width="65">Order:</Label>
        <StackPanel Name="Buttons" Orientation="Horizontal" HorizontalAlignment="Right"
                    Height="40" Margin="0,257,22,0">
```

```
            <Button Height="23" HorizontalAlignment="Right" Margin="0,0,12,12"
                Name="buttonSave" VerticalAlignment="Bottom" Width="75"
                    Click="buttonSaveChanges_Click">Save Changes
            </Button>
            <Button Height="23" Margin="0,0,12,12"
                Name="buttonClose" VerticalAlignment="Bottom" Width="75"
                    Click="buttonClose_Click">Close</Button>
        </StackPanel>
    </Grid>
</Window>
```

برای ارجاع به wcf data service بر روی پروژه راست کلیک و گزینه Add Service Reference را انتخاب نمایید در پنجره باز شده گزینه Discover را انتخاب تا سرویس را یافته و نام Namespase را Northwind بگذارید.

حال مانند کد زیر یک شی از مدل NorthwindEntities با آدرس وب سرویس ایجاد نموده ایم و نتیجه کوئری با استفاده از کلاس DataServiceCollection به DataContext گرید انتصاب داده ایم که البته پیش فرض آن آشنایی با DataBinding در WPF است.

```
private NorthwindEntities context;
        private string customerId = "ALFKI";
        private Uri svcUri = new Uri("http://localhost:8358/Northwind.svc");

        private void Window1_Loaded(object sender, RoutedEventArgs e)
        {
            try
            {
                context = new NorthwindEntities(svcUri);
                var ordersQuery = from o in context.Orders.Expand("Order_Details")
                                  where o.Customers.Customer_ID == customerId
                                  select o;
                DataServiceCollection<Orders> customerOrders = new
DataServiceCollection<Orders>(ordersQuery);
                this.orderItemsGrid.DataContext = customerOrders;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString());
            }
        }
```

با صدا زدن تابع SaveChanges مدل می‌توانید تغییرات را در پایگاه داده ذخیره نمایید.

```
private void buttonSaveChanges_Click(object sender, RoutedEventArgs e)
{
    try
    {
        context.SaveChanges();
    }
    catch (DataServiceRequestException ex)
    {
        MessageBox.Show(ex.ToString());

    }
}
```

برنامه را اجرا نمایید تا خروجی کار را مشاهده نمایید. مقادیر Quantity را تغییر دهید و دکمه Save Changes را انتخاب تا تغییرات دخیره شود.

در اینجا در یک برنامه ویندوزی استفاده از WCF Data Service را تست نمودیم اما براحتی به همین شیوه در یک برنامه وب نیز قابل استفاده است.

نظرات خوانندگان

نویسنده: حامد
تاریخ: ۱۳۹۱/۱۰/۱۵ ۲۳:۳

با درود
لطفاً درباره‌ی امنیت استفاده از این روش هم توضیحاتی بنویسید.
مخصوصاً امنیت از نوع Message، آیا می‌توان این نوع امنیت را در کنار WCF Data Service داشت؟

با سپاس

نویسنده: ابوالفضل رجب پور
تاریخ: ۱۳۹۲/۱۰/۲۳ ۱۴:۶

سلام
در صورتی که بخوایم سطح دسترسی به داده بر اساس کاربران داشته باشیم، چطور باید انجام بدیم؟ چون اینجا همه چیز انگاری خودکار و بدون واسطه انجام میشه

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۰/۲۳ ۱۴:۵۱

Securing WCF Data Services

OData and Authentication - Part 7 - Forms Authentication

عنوان: **بارگذاری UserControl در WPF به کمک الگوی MVVM**
نویسنده: شاهین کیاست
تاریخ: ۱۳۹۱/۱۰/۲۷ ۳:۳۵
آدرس: www.dotnettips.info
برچسب‌ها: MVVM, WPF

در نرم افزارهای تحت ویندوز روشها و سلیقه‌های متفاوتی برای چینش فرمها ، منوها و دیگر اجزای برنامه وجود دارد. در یک نرم افزار اتوماسیون اداری که فرمهای ورود اطلاعات زیادی دارد فضای کافی برای نمایش همه‌ی فرم‌ها به کاربر نیست. یکی از روش هایی که می‌تواند به کار رود تقسیم قسمت‌های مختلف نرم افزار در Viewهای جداگانه است. این کار استفاده‌ی مجدد از قسمت‌های مختلف و نگهداری کد را سهولت می‌بخشد.

الگوی متداولی که در نرم افزارهای WPF و Silverlight استفاده می‌شود الگوی MVVM است. (این الگو در جاوااسکریپت هم به سبب Stateful بودن استفاده می‌شود.) قبلا مطالب زیادی در این سایت جهت آموزش و توضیح این الگوی منتشر شده است.
فرض کنید نرم افزار از چند بخش تشکیل شده :

صفحه‌ی اصلی
منو
یک صفحه‌ی خوش آمدگویی
صفحه‌ی ورود و نمایش اطلاعات

می توان اجزا و تعریف هر یک از این قسمت‌ها را در یک UserControl قرار داد و در زمان مناسب آن را بارگذاری کرد.
سوالی که مطرح است بارگذاری UserControlها به کمک الگوی MVVM چگونه است ؟
کدهای XAML صفحه‌ی اصلی :

```
<Window x:Class="TwoViews.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        Title="MVVM Light View Switching"
        d:DesignHeight="300"
        d:DesignWidth="300"
        DataContext="{Binding Main,
                              Source={StaticResource Locator}}"
        ResizeMode="NoResize"
        SizeToContent="WidthAndHeight"
        mc:Ignorable="d">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>

        <ContentControl Content="{Binding CurrentViewModel}" />

        <DockPanel Grid.Row="1" Margin="5">
            <Button Width="75"
                    Height="23"
                    Command="{Binding SecondViewCommand}"
                    Content="Second View"
                    DockPanel.Dock="Right" />
            <Button Width="75"
                    Height="23"
                    Command="{Binding FirstViewCommand}"
                    Content="First View"
                    DockPanel.Dock="Left" />
        </DockPanel>
    </Grid>
</Window>
```

2 دکمه در صفحه‌ی اصلی وجود دارد ، یکی از آنها وظیفه‌ی بارگذاری View اول و دیگری وظیفه‌ی بارگذاری View دوم را دارد ، این دکمه‌ها نقش منو را در یک نرم افزار واقعی به عهده دارند.

کدهای View-Model گره خورده (به کمک الگوی ViewModolLocator) به View اصلی :

```
/// This is our MainViewModel that is tied to the MainWindow via the
    /// ViewModelLocator class.
    /// </summary>
    public class MainViewModel : ViewModelBase
    {
        /// <summary>
        /// Static instance of one of the ViewModels.
        /// </summary>
        private static readonly SecondViewModel SecondViewModel = new SecondViewModel();
        /// <summary>
        /// Static instance of one of the ViewModels.
        /// </summary>
        private static readonly FirstViewModel FirstViewModel = new FirstViewModel();
        /// <summary>
        /// The current view.
        /// </summary>
        private ViewModelBase _currentViewModel;
        /// <summary>
        /// Default constructor.  We set the initial view-model to 'FirstViewModel'.
        /// We also associate the commands with their execution actions.
        /// </summary>
        public MainViewModel()
        {
            CurrentViewModel = FirstViewModel;
            FirstViewCommand = new RelayCommand(ExecuteFirstViewCommand);
            SecondViewCommand = new RelayCommand(ExecuteSecondViewCommand);
        }
        /// <summary>
        /// The CurrentView property.  The setter is private since only this
        /// class can change the view via a command.  If the View is changed,
        /// we need to raise a property changed event (via INPC).
        /// </summary>
        public ViewModelBase CurrentViewModel
        {
            get { return _currentViewModel; }
            set
            {
                if (_currentViewModel == value)
                    return;
                _currentViewModel = value;
                RaisePropertyChanged("CurrentViewModel");
            }
        }
        /// <summary>
        /// Simple property to hold the 'FirstViewCommand' - when executed
        /// it will change the current view to the 'FirstView'
        /// </summary>
        public ICommand FirstViewCommand { get; private set; }
        /// <summary>
        /// Simple property to hold the 'SecondViewCommand' - when executed
        /// it will change the current view to the 'SecondView'
        /// </summary>
        public ICommand SecondViewCommand { get; private set; }
        /// <summary>
        /// Set the CurrentViewModel to 'FirstViewModel'
        /// </summary>
        private void ExecuteFirstViewCommand()
        {
            CurrentViewModel = FirstViewModel;
        }
        /// <summary>
        /// Set the CurrentViewModel to 'SecondViewModel'
        /// </summary>
        private void ExecuteSecondViewCommand()
        {
            CurrentViewModel = SecondViewModel;
        }
    }
```

این ViewModel از کلاس پایه‌ی چارچوب MVVM Light مشتق شده است. Commandها جهت Handle کردن کلیک دکمه‌ها هستند .
نکته‌ی اصلی این ViewModel پراپرتی CurrentViewModel می‌باشد. این پراپرتی به ویژگی Content کنترل ContentControl مقید (Bind) شده است. با کلیک شدن روی دکمه‌ها View مورد نظر به کاربر نمایش داده می‌شود.
WPF از کجا می‌داند کدام View را به ازای ViewModel خاص render کند ؟
در فایل App.xaml یک سری DataTemplate تعریف شده است :

```
<Application.Resources>
        <vm:ViewModelLocator x:Key="Locator" d:IsDataSource="True" />
        <!--
            We define the data templates here so we can apply them across the
            entire application.

            The data template just says that if our data type is of a particular
            view-model type, then render the appropriate view.  The framework
            takes care of this dynamically.  Note that the DataContext for
            the underlying view is already set at this point, so the
            view (UserControl), doesn't need to have it's DataContext set
            directly.
        -->
        <DataTemplate DataType="{x:Type vm:SecondViewModel}">
            <views:SecondView />
        </DataTemplate>
        <DataTemplate DataType="{x:Type vm:FirstViewModel}">
            <views:FirstView />
        </DataTemplate>
    </Application.Resources>
```

به کمک این DataTemplateها مشخص شده اگر نوع داده‌ی ما از یک نوع View-Model خاص می‌باشد View مناسب را به ازای آن Render کند. با تعریف DataTemplateها در App.Xaml می‌توان از آنها در سطح نرم افزار استفاده کرد. می‌توان DataTemplateها را جهت خلوت کردن App.xaml به Resource دیگری انتقال داد.

دریافت مثال : TwoViews.zip
منبع مثال

نظرات خوانندگان

نویسنده: افشار محبی
تاریخ: ۱۳۹۱/۱۰/۲۷ ۱۹:۲۰

کاربرد DataTemplate را نمی‌دانستم. اینجا یاد گرفتم. ممنون.

نویسنده: سعید
تاریخ: ۱۳۹۱/۱۰/۲۸ ۱۴:۲۳

علت خاصی داره که viewmodelهای دوم و اول رو به صورت static readonly تعریف کردید؟ اگر تعداد viewmodelها زیاد شد برنامه به مشکلات مصرف زیاد حافظه بر نمی‌خوره؟ شاید استفاده از خواص lazy در اینجا مناسب‌تر باشه. یا اینکه این وهله سازی فقط در زمان نیاز انجام بشه.

عنوان: **معماری لایه بندی نرم افزار #1**
نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۱/۱۲/۲۸ ۱۸:۵
آدرس: www.dotnettips.info
گروه‌ها: Design patterns, SoC, Separation of Concerns, ASP.Net, Domain Driven Design, DDD, SOLID Principals, C#, MVC, WPF, N-Layer Architecture

طراحی یک معماری خوب و مناسب یکی از عوامل مهم تولید یک برنامه کاربردی موفق می‌باشد. بنابراین انتخاب یک ساختار مناسب به منظور تولید برنامه کاربردی بسیار مهم و تا حدودی نیز سخت است. در اینجا یاد خواهیم گرفت که چگونه یک طراحی مناسب را انتخاب نماییم. همچنین روش‌های مختلف تولید برنامه‌های کاربردی را که مطمئنا شما هم از برخی از این روشها استفاده نمودید را بررسی می‌نماییم و مزایا و معایب آن را نیز به چالش می‌کشیم.

ضد الگو (Antipattern) – رابط کاربری هوشمند (Smart UI)

با استفاده از Visual Studio یا به اختصار VS ، می‌توانید برنامه‌های کاربردی را به راحتی تولید نمایید. طراحی رابط کاربری به آسانی عمل کشیدن و رها کردن (Drag & Drop) کنترل‌ها بر روی رابط کاربری قابل انجام است. همچنین در پشت رابط کاربری (Code Behind) تمامی عملیات مربوط به مدیریت رویدادها، دسترسی به داده ها، منطق تجاری و سایر نیازهای برنامه کاربردی، کد نویسی خواهند شد. مشکل این نوع کدنویسی بدین شرح است که تمامی نیازهای برنامه در پشت رابط کاربری قرار می‌گیرند و موجب تولید کدهای تکراری، غیر قابل تست، پیچیدگی کدنویسی و کاهش قابلیت استفاده مجدد از کد می‌گردد.

به این روش کد نویسی Smart UI می‌گویند که موجب تسهیل تولید برنامه‌های کاربردی می‌گردد. اما یکی از مشکلات عمده‌ی این روش، کاهش قابلیت نگهداری و پشتیبانی و عمر کوتاه برنامه‌های کاربردی می‌باشد که در برنامه‌های بزرگ به خوبی این مشکلات را حس خواهید کرد.

از آنجایی که تمامی برنامه نویسان مبتدی و تازه کار، از جمله من و شما در روزهای اول برنامه نویسی، به همین روش کدنویسی می‌کردیم، لزومی به ارائه مثال در رابطه با این نوع کدنویسی نمی‌بینم.

تفکیک و جدا سازی اجزای برنامه کاربردی (Separating Your Concern)

راه حل رفع مشکل Smart UI ، لایه بندی یا تفکیک اجزای برنامه از یکدیگر می‌باشد. لایه بندی برنامه می‌تواند به شکل‌های مختلفی صورت بگیرد. این کار می‌تواند توسط تفکیک کدها از طریق فضای نام (Namespace) ، پوشه بندی فایلهای حاوی کد و یا جداسازی کدها در پروژه‌های متفاوت انجام شود. در شکل زیر نمونه ای از معماری لایه بندی را برای یک برنامه کاربردی بزرگ می‌بینید.

به منظور پیاده سازی یک برنامه کاربردی لایه بندی شده و تفکیک اجزای برنامه از یکدیگر، مثالی را پیاده سازی خواهیم کرد. ممکن است در این مثال با مسائل جدید و شیوه‌های پیاده سازی جدیدی مواجه شوید که این نوع پیاده سازی برای شما قابل درک نباشد. اگر کمی صبر پیشه نمایید و این مجموعه‌ی آموزشی را پیگیری کنید، تمامی مسائل نامانوس با جزئیات بیان خواهند شد و درک آن برای شما ساده خواهد گشت. قبل از شروع این موضوع را هم به عرض برسانم که علت اصلی این نوع پیاده سازی، انعطاف پذیری بالای برنامه کاربردی، پشتیبانی و نگهداری آسان، قابلیت تست پذیری با استفاده از ابزارهای تست، پیاده سازی پروژه بصورت تیمی و تقسیم بخشهای مختلف برنامه بین اعضای تیم و سایر مزایای فوق العاده آن می‌باشد.

1- Visual Studio را باز کنید و یک Solution خالی با نام SoCPatterns.Layered ایجاد نمایید.

· جهت ایجاد Solution خالی، پس از انتخاب New Project ، از سمت چپ گزینه Other Project Types و سپس Visual Studio Solutions را انتخاب نمایید. از سمت راست گزینه Blank Solution را انتخاب کنید.

2- بر روی Solution کلیک راست نموده و از گزینه Add > New Project یک پروژه Class Library با نام SoCPatterns.Layered.Repository ایجاد کنید.

-3 با استفاده از روش فوق سه پروژه Class Library دیگر با نامهای زیر را به Solution اضافه کنید:

SoCPatterns.Layered.Model

SoCPatterns.Layered.Service

SoCPatterns.Layered.Presentation

-4 با توجه به نیاز خود یک پروژه دیگر را باید به Solution اضافه نمایید. نوع و نام پروژه در زیر لیست شده است که شما باید با توجه به نیاز خود یکی از پروژه‌های موجود در لیست را به Solution اضافه کنید.

(Windows Forms Application (SoCPatterns.Layered.WinUI

(WPF Application (SoCPatterns.Layered.WpfUI

(ASP.NET Empty Web Application (SoCPatterns.Layered.WebUI

(ASP.NET MVC 4 Web Application (SoCPatterns.Layered.MvcUI

-5 بر روی پروژه SoCPatterns.Layered.Repository کلیک راست نمایید و با انتخاب گزینه Add Reference به پروژه‌ی SoCPatterns.Layered.Model ارجاع دهید.

-6 بر روی پروژه SoCPatterns.Layered.Service کلیک راست نمایید و با انتخاب گزینه Add Reference به پروژه‌های SoCPatterns.Layered.Model و SoCPatterns.Layered.Repository ارجاع دهید.

-7 بر روی پروژه SoCPatterns.Layered.Presentation کلیک راست نمایید و با انتخاب گزینه Add Reference به پروژه‌های SoCPatterns.Layered.Model و SoCPatterns.Layered.Service ارجاع دهید.

-8 بر روی پروژه‌ی UI خود به عنوان مثال SoCPatterns.Layered.WebUI کلیک راست نمایید و با انتخاب گزینه Add Reference به پروژه‌های SoCPatterns.Layered.Service ، SoCPatterns.Layered.Repository ، SoCPatterns.Layered.Model و SoCPatterns.Layered.Presentation ارجاع دهید.

-9 بر روی پروژه‌ی UI خود به عنوان مثال SoCPatterns.Layered.WebUI کلیک راست نمایید و با انتخاب گزینه Set as StartUp Project پروژه‌ی اجرایی را مشخص کنید.

-10 بر روی Solution کلیک راست نمایید و با انتخاب گزینه Add > New Solution Folder پوشه‌های زیر را اضافه نموده و پروژه‌های مرتبط را با عمل Drag & Drop در داخل پوشه‌ی مورد نظر قرار دهید.

1. UI
§ SoCPatterns.Layered.WebUI

2. Presentation Layer
§ SoCPatterns.Layered.Presentation

3. Service Layer
§ SoCPatterns.Layered.Service

4. Domain Layer
§ SoCPatterns.Layered.Model

5. Data Layer
§ SoCPatterns.Layered.Repository

توجه داشته باشید که پوشه بندی برای مرتب سازی لایه‌ها و دسترسی راحت‌تر به آنها می‌باشد.

پیاده سازی ساختار لایه بندی برنامه به صورت کامل انجام شد. حال به پیاده سازی کدهای مربوط به هر یک از لایه‌ها و بخش‌ها می‌پردازیم و از لایه Domain شروع خواهیم کرد.

نظرات خوانندگان

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۱/۱۲/۲۸ ۲۰:۲

مباحثی از این دست بسیار مفید و ضروری است و به شدت استقبال می‌کنم از شروع این سری مقالات. البته پیش‌تر هم مطالبی از این دست در سایت ارائه شده است که امیدوارم این سری مقالات بتونه تا حدی پراکندگی مطالب مربوطه را از بین ببرد. فقط لطف بفرمایید در این سری مقالات مرز بندی مشخصی برای برخی مفاهیم در نظر داشته باشید. به عنوان مثال گاهی در یک مقاله مفهوم Repository معادل مفهوم لایه سرویس در مقاله دیگر است. یا Domain Model مرز مشخصی با View Model داشته باشد. همچنین بحث‌های خوبی مهندس نصیری عزیز در مورد عدم نیاز به ایجاد Repository در مفهوم متداول در هنگام استفاده از EF داشتند که در رفرنس‌های معتبر دیگری هم مشاهده می‌شود. لطفاً در این مورد نیز بحث بیشتری با مرز بندی مشخص داشته باشید.

نویسنده: حسن
تاریخ: ۱۳۹۱/۱۲/۲۸ ۲۲:۵

آیا صرفا تعریف چند ماژول مختلف برنامه را لایه بندی می‌کند و ضمانتی است بر لایه بندی صحیح، یا اینکه استفاده از الگوهای MVC و MVVM می‌توانند ضمانتی باشند بر جدا سازی حداقل لایه نمایشی برنامه از لایه‌های دیگر، حتی اگر تمام اجزای یک برنامه داخل یک اسمبلی اصلی قرار گرفته باشند؟

نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۱/۱۲/۲۹ ۰:۵۳

این سری مقالات جمع بندی کامل معماری لایه بندی نرم افزار است. پس از پایان مقالات یک پروژه کامل رو با معماری منتخب پیاده سازی میکنم تا تمامی شک و شبهات برطرف بشه. در مورد مرزبندی لایه‌ها هم صحبت می‌کنم و مفهوم هر کدام را دقیقا توضیح میدم.

نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۱/۱۲/۲۹ ۰:۵۹

اگر مقاله فوق رو با دقت بخونید متوجه میشید که MVC و MVVM در لایه UI پیاده سازی میشن. البته در MVC لایه Model رو به Domain Model و Repository در برخی مواقع لایه Controller رو در لایه Presentation قرار میدن. در MVVM نیز لایه Model در Domain Model و Repository و لایه View Model نیز در لایه Presentation قرار میگیره. همچنین View Modelها نیز در لایه Service قرار میگیرن.
در مورد ماژول بندی هم اگر در مقاله خونده باشید میتونید لایه‌ها رو از طریق پوشه ها، فضای نام و یا پروژه‌ها از هم جدا کنید

نویسنده: حسن
تاریخ: ۱۳۹۱/۱۲/۲۹ ۱۰:۱۴

شما در مطلبتون با ضدالگو شروع کردید و عنوان کردید که روش code behind یک سری مشکلاتی رو داره. سؤال من هم این بود که آیا صرفا تعریف چند ماژول جدید می‌تواند ضمانتی باشد بر رفع مشکل code behind یا اینکه با این ماژول‌ها هم نهایتا همان مشکل قبل پابرجا است یا می‌تواند پابرجا باشد.

ضمن اینکه تعریف شما از لایه دقیقا چی هست؟ به نظر فقط تعریف یک اسمبلی در اینجا لایه نام گرفته.

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۱/۱۲/۲۹ ۱۱:۵۸

صحبت شما کاملاً صحیح است و صرفاً با ماژولار کردن به معماری چند لایه نمی‌رسیم. اما نویسنده مقاله نیز چنین نگفته و در پایان مقاله بحث پایان **ساختار** چند لایه است و نه پایان پروژه. این قسمت اول این سری مقالات است و قطعاً در هنگام پیاده سازی کدهای هر لایه مباحثی مطرح خواهد شد تا تضمین مفهوم مورد نظر شما باشد.

نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۱/۱۲/۲۹ ۱۲:۳۸

با تشکر از دوست عزیزم جناب آقای آرمان فرقانی با توضیحی که دادند.
یکی از دلایل این شیوه کد نویسی امکان تست نویسی برای هر یک از لایه‌ها و همچنین استقلال لایه‌ها از هم دیگه هست که هر لایه بتونه بدون وجود لایه‌ی دیگه تست بشه. ماژولار کردنه ممکنه مشکل Smart UI رو حل کنه و ممکنه حل نکنه. بستگی به شیوه کد نویسی داره.

نویسنده: بهروز
تاریخ: ۱۳۹۱/۱۲/۲۹ ۱۳:۱۱

وقتی نظرات زیر مطلب شما رو میخونم میفهمم که نیاز به این سری آموزشی که دارید ارائه میدید چقدر زیاد احساس میشه فقط می‌خواستم بگم بر سر این مبحثی که دارید ارائه می‌دید اختلاف بین علما زیاد است!(حتی در عمل و در شرکت‌های نرم افزاری که تا به حال دیدم چه برسد در سطح آموزش...)
امیدوارم این حساسیت رو در نظر بگیرید و همه ما پس از مطالعه این سری آموزشی به فهم مشترک و یکسانی در مورد مفاهیم موجود برسیم
فکر می‌کنم وجود یک پروژه برای دست یافتن به این هدف هم ضروری باشد
باز هم تشکر

نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۱/۱۲/۲۹ ۱۳:۵۹

من هم وقتی کار بر روی این معماری رو شروع کردم با مشکلات زیادی روبرو بودم و خیلی از مسائل برای من هم نامانوس و غیر قابل هضم بود. ولی بعد از اینکه چند پروژه نرم افزاری رو با این معماری پیاده سازی کردم فهم بیشتری نسبت به اون پیدا کردم و خیلی از مشکلات موجود رو با دقت بالا و با در نظر گرفتن تمامی الگوها رفع کردم. امیدوارم این حس مشترک بوجود بیاد. ولی دلیل اصلی ایجاد تکنولوژی‌ها و معماری‌های جدید اختلاف نظر بین علماست. این اختلاف نظر در اکثر مواقع میتونه مفید باشه.
ممنون دوست عزیز

نویسنده: مسعود مشهدی
تاریخ: ۱۳۹۲/۰۱/۰۴ ۱۸:۳۳

با سلام
بابت مطالبتون سپاسگذارم
همون طور که خودتون گفتید نظرات و شیوه‌های متفاوتی در نوع لایه بندی‌ها وجود داره.
در مقام مقایسه لایه بندی زیر چه وجه اشتراک و تفاوتی با لایه بندی شما داره.

```
Application.Web
Application.Manager

Application.DAL

Application.DTO

Application.Core
```

Application.Common

با تشکر

Business Layer یا Domain Model

پیاده سازی را از منطق تجاری یا Business Logic آغاز می‌کنیم. در روش کد نویسی Smart UI ، منطق تجاری در Code Behind قرار می‌گرفت اما در روش لایه بندی، منطق تجاری و روابط بین داده‌ها در Domain Model طراحی و پیاده سازی می‌شوند. در مطالب بعدی راجع به Domain Model و الگوهای پیاده سازی آن بیشتر صحبت خواهم کرد اما بصورت خلاصه این لایه یک مدل مفهومی از سیستم می‌باشد که شامل تمامی موجودیت‌ها و روابط بین آنهاست.

الگوی Domain Model جهت سازماندهی پیچیدگی‌های موجود در منطق تجاری و روابط بین موجودیت‌ها طراحی شده است.

شکل زیر مدلی را نشان می‌دهد که می‌خواهیم آن را پیاده سازی نماییم. کلاس Product موجودیتی برای ارائه محصولات یک فروشگاه می‌باشد. کلاس Price جهت تشخیص قیمت محصول، میزان سود و تخفیف محصول و همچنین استراتژی‌های تخفیف با توجه به منطق تجاری سیستم می‌باشد. در این استراتژی همکاران تجاری از مشتریان عادی تفکیک شده اند.

Domain Model را در پروژه SoCPatterns.Layered.Model پیاده سازی می‌کنیم. بنابراین به این پروژه یک Interface به نام IDiscountStrategy را با کد زیر اضافه نمایید:

```
public interface IDiscountStrategy
{
    decimal ApplyExtraDiscountsTo(decimal originalSalePrice);
}
```

علت این نوع نامگذاری Interface فوق، انطباق آن با الگوی Strategy Design Pattern می‌باشد که در مطالب بعدی در مورد این الگو بیشتر صحبت خواهم کرد. استفاده از این الگو نیز به این دلیل بود که این الگو مختص الگوریتم هایی است که در زمان اجرا قابل انتخاب و تغییر خواهند بود.

توجه داشته باشید که معمولا نام Design Pattern انتخاب شده برای پیاده سازی کلاس را بصورت پسوند در انتهای نام کلاس ذکر می‌کنند تا با یک نگاه، برنامه نویس بتواند الگوی مورد نظر را تشخیص دهد و مجبور به بررسی کد نباشد. البته به دلیل تشابه برخی از الگوها، امکان تشخیص الگو، در پاره ای از موارد وجود ندارد و یا به سختی امکان پذیر است.

الگوی Strategy یک الگوریتم را قادر می‌سازد تا در داخل یک کلاس کپسوله شود و در زمان اجرا به منظور تغییر رفتار شی، بین رفتارهای مختلف سوئیچ شود.

حال باید دو کلاس به منظور پیاده سازی روال تخفیف ایجاد کنیم. ابتدا کلاسی با نام TradeDiscountStrategy را با کد زیر به پروژه SoCPatterns.Layered.Model اضافه کنید:

```
public class TradeDiscountStrategy : IDiscountStrategy
{
    public decimal ApplyExtraDiscountsTo(decimal originalSalePrice)
    {
        return originalSalePrice * 0.95M;
    }
}
```

سپس با توجه به الگوی Null Object کلاسی با نام NullDiscountStrategy را با کد زیر به پروژه SoCPatterns.Layered.Model اضافه کنید:

```
public class NullDiscountStrategy : IDiscountStrategy
{
    public decimal ApplyExtraDiscountsTo(decimal originalSalePrice)
    {
        return originalSalePrice;
    }
}
```

از الگوی Null Object زمانی استفاده می‌شود که نمی‌خواهید و یا در برخی مواقع نمی‌توانید یک نمونه (Instance) معتبر را برای یک کلاس ایجاد نمایید و همچنین مایل نیستید که مقدار Null را برای یک نمونه از کلاس برگردانید. در مباحث بعدی با جزئیات بیشتری در مورد الگوها صحبت خواهم کرد.

با توجه به استراتژی‌های تخفیف کلاس Price را ایجاد کنید. کلاسی با نام Price را با کد زیر به پروژه SoCPatterns.Layered.Model اضافه کنید:

```
public class Price
{
    private IDiscountStrategy _discountStrategy = new NullDiscountStrategy();
    private decimal _rrp;
    private decimal _sellingPrice;
    public Price(decimal rrp, decimal sellingPrice)
    {
        _rrp = rrp;
        _sellingPrice = sellingPrice;
    }
    public void SetDiscountStrategyTo(IDiscountStrategy discountStrategy)
    {
        _discountStrategy = discountStrategy;
    }
    public decimal SellingPrice
    {
        get { return _discountStrategy.ApplyExtraDiscountsTo(_sellingPrice); }
```

```
    }
    public decimal Rrp
    {
        get { return _rrp; }
    }
    public decimal Discount
    {
        get {
            if (Rrp > SellingPrice)
                return (Rrp - SellingPrice);
            else
                return 0;
        }
    }
    public decimal Savings
    {
        get{
            if (Rrp > SellingPrice)
                return 1 - (SellingPrice / Rrp);
            else
                return 0;
        }
    }
}
```

کلاس Price از نوعی Dependency Injection به نام Setter Injection در متد SetDiscountStrategyTo استفاده نموده است که استراتژی تخفیف را برای کالا مشخص می‌نماید. نوع دیگری از Dependency Injection با نام Constructor Injection وجود دارد که در مباحث بعدی در مورد آن بیشتر صحبت خواهم کرد.

جهت تکمیل لایه Model ، کلاس Product را با کد زیر به پروژه SoCPatterns.Layered.Model اضافه کنید:

```
public class Product
{
    public int Id {get; set;}
    public string Name { get; set; }
    public Price Price { get; set; }
}
```

موجودیت‌های تجاری ایجاد شدند اما باید روشی اتخاذ نمایید تا لایه Model نسبت به منبع داده ای بصورت مستقل عمل نماید. به سرویسی نیاز دارید که به کلاینت‌ها اجازه بدهد تا با لایه مدل در اتباط باشند و محصولات مورد نظر خود را با توجه به تخفیف اعمال شده برای رابط کاربری برگردانند. برای اینکه کلاینت‌ها قادر باشند تا نوع تخفیف را مشخص نمایند، باید یک نوع شمارشی ایجاد کنید که به عنوان پارامتر ورودی متد سرویس استفاده شود. بنابراین نوع شمارشی CustomerType را با کد زیر به پروژه SoCPatterns.Layered.Model اضافه کنید:

```
public enum CustomerType
{
    Standard = 0,
    Trade = 1
}
```

برای اینکه تشخیص دهیم کدام یک از استراتژی‌های تخفیف باید بر روی قیمت محصول اعمال گردد، نیاز داریم کلاسی را ایجاد کنیم تا با توجه به CustomerType تخفیف مورد نظر را اعمال نماید. کلاسی با نام DiscountFactory را با کد زیر ایجاد نمایید:

```
public static class DiscountFactory
{
```

```
    public static IDiscountStrategy GetDiscountStrategyFor
        (CustomerType customerType)
    {
        switch (customerType)
        {
            case CustomerType.Trade:
                return new TradeDiscountStrategy();
            default:
                return new NullDiscountStrategy();
        }
    }
}
```

در طراحی کلاس فوق از الگوی Factory استفاده شده است. این الگو یک کلاس را قادر می‌سازد تا با توجه به شرایط، یک شی معتبر را از یک کلاس ایجاد نماید. همانند الگوهای قبلی، در مورد این الگو نیز در مباحث بعدی بیشتر صحبت خواهم کرد.

لایه‌ی سرویس با برقراری ارتباط با منبع داده ای، داده‌های مورد نیاز خود را بر می‌گرداند. برای این منظور از الگوی Repository استفاده می‌کنیم. از آنجایی که لایه Model باید مستقل از منبع داده ای عمل کند و نیازی به شناسایی نوع منبع داده ای ندارد، جهت پیاده سازی الگوی Repository از Interface استفاده می‌شود. یک Interface به نام IProductRepository را با کد زیر به پروژه SoCPatterns.Layered.Model اضافه کنید:

```
public interface IProductRepository
{
    IList<Product> FindAll();
}
```

الگوی Repository به عنوان یک مجموعه‌ی در حافظه (In-Memory Collection) یا انباره ای از موجودیت‌های تجاری عمل می‌کند که نسبت به زیر بنای ساختاری منبع داده ای کاملا مستقل می‌باشد.

کلاس سرویس باید بتواند استراتژی تخفیف را بر روی مجموعه ای از محصولات اعمال نماید. برای این منظور باید یک Collection سفارشی ایجاد نماییم. اما من ترجیح می‌دهم از Extension Methods برای اعمال تخفیف بر روی محصولات استفاده کنم. بنابراین کلاسی به نام ProductListExtensionMethods را با کد زیر به پروژه SoCPatterns.Layered.Model اضافه کنید:

```
public static class ProductListExtensionMethods
{
    public static void Apply(this IList<Product> products,
                                     IDiscountStrategy discountStrategy)
    {
        foreach (Product p in products)
        {
            p.Price.SetDiscountStrategyTo(discountStrategy);
        }
    }
}
```

الگوی Separated Interface تضمین می‌کند که کلاینت از پیاده سازی واقعی کاملا نامطلع می‌باشد و می‌تواند برنامه نویس را به سمت Abstraction و Dependency Inversion به جای پیاده سازی واقعی سوق دهد.

حال باید کلاس Service را ایجاد کنیم تا از طریق این کلاس، کلاینت با لایه Model در ارتباط باشد. کلاسی به نام ProductService را با کد زیر به پروژه SoCPatterns.Layered.Model اضافه کنید:

```
public class ProductService
{
    private IProductRepository _productRepository;
    public ProductService(IProductRepository productRepository)
    {
        _productRepository = productRepository;
    }
    public IList<Product> GetAllProductsFor(CustomerType customerType)
    {
        IDiscountStrategy discountStrategy =
                                DiscountFactory.GetDiscountStrategyFor(customerType);
        IList<Product> products = _productRepository.FindAll();
        products.Apply(discountStrategy);
        return products;
    }
}
```

در اینجا کدنویسی منطق تجاری در Domain Model به پایان رسیده است. همانطور که گفته شد، لایه‌ی Business یا همان Domain Model به هیچ منبع داده ای خاصی وابسته نیست و به جای پیاده سازی کدهای منبع داده ای، از Interface ها به منظور برقراری ارتباط با پایگاه داده استفاده شده است. پیاده سازی کدهای منبع داده ای را به لایه‌ی Repository واگذار نمودیم که در بخش‌های بعدی نحوه پیاده سازی آن را مشاهده خواهید کرد. این امر موجب می‌شود تا لایه Model درگیر پیچیدگی‌ها و کد نویسی‌های منبع داده ای نشود و بتواند به صورت مستقل و فارغ از بخشهای مختلف برنامه تست شود. لایه بعدی که می‌خواهیم کد نویسی آن را آغاز کنیم، لایه‌ی Service می‌باشد.

در کد نویسی‌های فوق از الگوهای طراحی (Design Patterns) متعددی استفاده شده است که به صورت مختصر در مورد آنها صحبت کردم. اصلا جای نگرانی نیست، چون در مباحث بعدی به صورت مفصل در مورد آنها صحبت خواهم کرد. در ضمن، ممکن است روال یادگیری و آموزش بسیار نامفهوم باشد که برای فهم بیشتر موضوع، باید کدها را بصورت کامل تست نموده و مثالهایی را پیاده سازی نمایید.

نظرات خوانندگان

نویسنده: سینا کردی
تاریخ: ۱۳۹۱/۱۲/۳۰ ۴:۱۰

سلام
ممنون از شما این بخش هم کامل و زیبا بود
ولی کمی فشرده بود
لطفا اگر ممکن هست در مورد معماری‌ها و الگوها و بهترین‌های آنها کمی توضیح دهید یا منبعی معرفی کنید تا این الگوها و معماری برای ما بیشتر مفهوم بشه
من در این زمینه تازه کارم و از شما میخواهم که من رو راهنمایی کنید که چه مقدماتی در این زمینه‌ها نیاز دارم
باز هم ممنون.

نویسنده: علي
تاریخ: ۱۳۹۱/۱۲/۳۰ ۹:۱۲

در همین سایت مباحث الگوهای طراحی و Refactoring مفید هستند.

و یا الگوهای طراحی Agile رو هم در اینجا می‌تونید پیگیری کنید.

نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۱/۱۲/۳۰ ۱۱:۳۸

فشردگی این مباحث بخاطر این بود که میخواستم فعلا یک نمونه پروژه رو آموزش بودم تا یک شمای کلی از کاری که می‌خواهیم انجام بدیم رو ببینید. در مباحث بعدی این مباحث رو بازتر می‌کنم. خود من برای مطالعه و جمع بندی این مباحث منابع زیادی رو مطالعه کردم. واقعا برای بعضی مباحث نمیشه به یک منبع اکتفا کرد.

نویسنده: محسن.د
تاریخ: ۱۳۹۱/۱۲/۳۰ ۱۷:۱

بسیار عالی

آیا فراخوانی مستقیم تابع SetDiscountStrategyTo کلاس Price در تابع الحاقی Apply از نظر کپسوله سازی مورد اشکال نیست ؟ بهتر نیست که برای خود کلاس Product یک تابع پیاده سازی کنیم که در درون خودش تابع Price.SetDiscountStrategyTo را فراخوانی کند و به این شکل کلاس‌های بیرونی رو از تغییرات درونی کلاس Product مستقل کنیم ؟

نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۱/۱۲/۳۰ ۱۸:۱

دوست عزیزم. متد Apply یک Extension Method برای IList<Product> است. اگر این متد تعریف نمی‌شد شما باید در کلاس سرویس حلقه foreach رو قرار می‌دادید. البته با این حال در قسمت هایی از طراحی کلاسها که الگوهای طراحی را زیر سوال نمی‌برد و تست پذیری را دچار مشکل نمی‌کند، طراحی سلیقه ای است. مقاله من هم آیه‌ی نازل شده نیست که دستخوش تغییرات نشود. شما می‌توانید با سلیقه و دید فنی خود تغییرات مورد نظر رو اعمال کنید. ولی اگر نظر من را بخواهید این طراحی مناسب‌تر است.

نویسنده: رضا عرب
تاریخ: ۱۳۹۲/۰۱/۰۹ ۱۴:۴۵

خسته نباشید، واقعا ممنونم آقای خوشبخت، لطفا به نگارش این دست مطالب مرتبط با طراحی ادامه دهید، زمینه بکریه که کمتر عملی به آن پرداخته شده و این نوع نگارش شما فراتر از یک معرفیه که واقعا جای تشکر داره.

نویسنده: f.tahan36
تاریخ: ۱۳۹۲/۰۲/۲۹ ۱۷:۱۰

با سلام

تفاوت factory با design factory در چیست؟ (با مثال کد)

و virtual کردن یک تابع معمولی با virtual کردن تابع سازنده چه تفاوتی دارد؟

با تشکر

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۳۰ ۰:۴۰

از همون رندهایی هستی که تمرین کلاسیت رو آوردی اینجا؟! :)

عنوان: **نحوه نمایش تمام آیکون‌های تعریف شده در یک قلم در WPF**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۰۱ ۱۰:۵۵
آدرس: www.dotnettips.info
برچسب‌ها: WPF

سال نو مبارک! به امید روزهایی شاد، سلامت و پر برکت.

پیرو مطلب قلم‌هایی حاوی آیکون که خصوصا در برنامه‌های مترو بیشتر مرسوم شده‌اند، شاید بد نباشد کار برنامه Character Map ویندوز را با WPF شبیه سازی کنیم.

ابتدا Model و ViewModel این برنامه را درنظر بگیرید:

```
namespace CrMap.Models
{
    public class Symbol
    {
        public char Character { set; get; }
        public string CharacterCode { set; get; }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Windows.Media;
using CrMap.Models;

namespace CrMap.ViewModels
{
    public class CrMapViewModel
    {
        public IList<Symbol> Symbols { set; get; }
        public int GridRows { set; get; }
        public int GridCols { set; get; }

        public CrMapViewModel()
        {
            fillDataSource();
        }

        private void fillDataSource()
        {
            Symbols = new List<Symbol>();
            GridCols = 15;

            var fontFamily = new FontFamily(new Uri("pack://application:,,,/"), "/Fonts/#whhglyphs");

            GlyphTypeface glyph = null;
            foreach (var typeface in fontFamily.GetTypefaces())
            {
                if (typeface.TryGetGlyphTypeface(out glyph) && (glyph != null))
                    break;
            }

            if (glyph == null)
                throw new InvalidOperationException("Couldn't find a GlyphTypeface.");

            GridRows = (glyph.CharacterToGlyphMap.Count / GridCols) + 1;

            foreach (var item in glyph.CharacterToGlyphMap)
            {
                var index = item.Key;
                Symbols.Add(new Symbol
                {
                    Character = Convert.ToChar(index),
                    CharacterCode = string.Format("&#x{0:X}", index)
                });
            }
        }
    }
}
```

توضیحات:

یک سری قابلیت جالب در WPF برای استخراج اطلاعات قلم‌ها وجود دارند که در فضای نام System.Windows.Media اسمبلی PresentationCore.dll قرار گرفته‌اند. برای نمونه پس از وهله سازی FontFamily بر اساس یک قلم مدفون شده در برنامه، امکان استخراج تعداد گلیف‌های موجود در این قلم وجود دارد که نحوه انجام آن‌را در متد fillDataSource ملاحظه می‌کنید. این اطلاعات استخراج شده و لیست Symbols برنامه را تشکیل می‌دهند. در نهایت برای نمایش این اطلاعات، از ترکیب ItemsControl و UniformGrid استفاده خواهیم کرد:

```
<Window x:Class="CrMap.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:vm="clr-namespace:CrMap.ViewModels"
        Title="MainWindow" WindowStartupLocation="CenterScreen" WindowState="Maximized"
        Height="350" Width="525">
    <Window.Resources>
        <vm:CrMapViewModel x:Key="vmCrMapViewModel" />
    </Window.Resources>
    <ScrollViewer VerticalScrollBarVisibility="Visible">
        <ItemsControl
                      DataContext="{StaticResource vmCrMapViewModel}"
                      ItemsSource="{Binding Symbols}"
                      Name="MainItemsControl"
                      VerticalAlignment="Top"
                      HorizontalAlignment="Center"
                      Grid.Column="0" Grid.Row="1" Grid.ColumnSpan="4">
            <ItemsControl.ItemsPanel>
                <ItemsPanelTemplate>
                    <UniformGrid
                        HorizontalAlignment="Center"
                        VerticalAlignment="Center"
                        Columns="{Binding GridCols}"
                        Rows="{Binding GridRows}">
                    </UniformGrid>
                </ItemsPanelTemplate>
            </ItemsControl.ItemsPanel>
            <ItemsControl.ItemTemplate>
                <DataTemplate>
                    <ContentControl>
                        <Border BorderBrush="SlateGray"
                            HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
                            BorderThickness="1" CornerRadius="3" Margin="3">
                            <StackPanel Margin="3" Orientation="Vertical">
                                <TextBlock
                                    HorizontalAlignment="Center"
                                    VerticalAlignment="Center"
                                    FontFamily="Fonts/#whhglyphs"
                                    Foreground="DarkRed"
                                    FontSize="17"
                                    Text="{Binding Character}" />
                                <TextBlock
                                    HorizontalAlignment="Center"
                                    VerticalAlignment="Center"
                                    Text="{Binding CharacterCode}" />
                            </StackPanel>
                        </Border>
                    </ContentControl>
                </DataTemplate>
            </ItemsControl.ItemTemplate>
        </ItemsControl>
    </ScrollViewer>
</Window>
```

دریافت مثال این مطلب

CrMap.zip

نظرات خوانندگان

نویسنده: Petek
تاریخ: ۱۳۹۲/۰۱/۲۶ ۱۷:۵۷

با سلام مهندس
مهندس آیا امکانش هست که در پروژه‌های WPF خودمون یه فونت آیکون تعریف کنیم و آیکون‌های مورد نظرمون رو درش قرار بدیم و این فونت رو به پروژه Embed کرده و استفاده کنیم اگر این عمل شدنی هست ممنون میشم راهنمایی کنید که با چه ابزاری و به چه صورت این کار رو میشه انجام داد . با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۲۶ ۱۸:۷

ابزار فونت ادیتور خوب هست. مثلا یک نمونه در اینجا

عنوان: **نحوه استخراج آیکون‌های یک قلم در WPF**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۰۲ ۸:۵
آدرس: www.dotnettips.info
برچسب‌ها: WPF

مطلب « نحوه نمایش تمام آیکون‌های تعریف شده در یک قلم در WPF » را در نظر بگیرید. **سؤال:** اگر در یک برنامه تنها به تعدادی از این آیکون‌ها یا گلیف‌ها نیاز بود آیا می‌توان این‌ها را به صورت مجزا استخراج و استفاده کرد؟

پاسخ: بلی. همان کلاس FontFamily موجود در اسمبلی PresentationCore.dll، امکان تبدیل یک گلیف را به معادل هندسی آن نیز دارد. در ادامه کدهای آن‌را مرور خواهیم کرد:

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Windows;
using System.Windows.Media;
using CrMap.Models;

namespace CrMap.ViewModels
{
    public class CrMapViewModel
    {
        public IList<Symbol> Symbols { set; get; }
        public int GridRows { set; get; }
        public int GridCols { set; get; }

        public CrMapViewModel()
        {
            fillDataSource();
        }

        private void fillDataSource()
        {
            Symbols = new List<Symbol>();
            GridCols = 15;

            var fontFamily = new FontFamily(new Uri("pack://application:,,,/"), "/Fonts/#whhglyphs");

            GlyphTypeface glyph = null;
            Typeface glyphTypeface = null;
            foreach (var typeface in fontFamily.GetTypefaces())
            {
                if (typeface.TryGetGlyphTypeface(out glyph) && (glyph != null))
                {
                    glyphTypeface = typeface;
                    break;
                }
            }

            if (glyph == null)
                throw new InvalidOperationException("Couldn't find a GlyphTypeface.");

            GridRows = (glyph.CharacterToGlyphMap.Count / GridCols) + 1;

            foreach (var item in glyph.CharacterToGlyphMap)
            {
                var index = item.Key;
                Symbols.Add(new Symbol
                {
                    Character = Convert.ToChar(index),
                    CharacterCode = string.Format("&#x{0:X}", index)
                });

                saveToFile(glyphTypeface, index);
            }
        }

        private static void saveToFile(Typeface glyphTypeface, int index)
        {
            var formattedText = new FormattedText(
                                    textToFormat: Convert.ToChar(index).ToString(),
                                    culture: new CultureInfo("en-us"),
                                    flowDirection: FlowDirection.LeftToRight,
                                    typeface: glyphTypeface,
                                    emSize: 20,
```

```
                                        foreground: Brushes.Black);
            var geometry = formattedText.BuildGeometry(new Point(0, 0));
            var path = geometry.GetFlattenedPathGeometry();
            File.WriteAllText(index + ".path", path.ToString());
        }
    }
}
```

در اینجا تنها متد saveToFile در مقایسه با قسمت قبل افزوده شده است.

شیء FormattedText دارای متدی است به نام BuildGeometry که اطلاعات یک گلیف را تبدیل به معادل هندسی آن می‌کند. سپس توسط GetFlattenedPathGeometry معادل Path آن‌را می‌توان بدست آورد. برای مثال اگر پس از اجرای این مثال، به فایل path.48 تولیدی آن مراجعه کنیم، چنین خروجی را می‌توان مشاهده کرد:

```
F1M5,7.47150993347168L5,17.2566661834717 5.732421875,19.0242443084717 7.5,19.7566661834717
12.5,19.7566661834717 13.69140625,19.4441661834717 5,7.47150993347168z
M7.5,4.75666618347168L6.30859375,5.06916618347168 15,17.0418224334717
15,7.25666618347168 14.267578125,5.48908805847168 12.5,4.75666618347168
7.5,4.75666618347168z M7.5,2.25666618347168L12.5,2.25666618347168
14.4189453125,2.62287712097168 16.03515625,3.72150993347168 17.1337890625,5.33772134780884
17.5,7.25666618347168 17.5,17.2566661834717 17.1337890625,19.1756114959717
16.03515625,20.7918224334717 14.4189453125,21.8904552459717 12.5,22.2566661834717
7.5,22.2566661834717 5.5810546875,21.8904552459717 3.96484375,20.7918224334717
2.8662109375,19.1756114959717 2.5,17.2566661834717 2.5,7.25666618347168 2.8662109375,5.33772134780884
3.96484375,3.72150993347168 5.5810546875,2.62287712097168 7.5,2.25666618347168z
```

که برای استفاده از اطلاعات آن در WPF می‌توان نوشت:

```
<Path Stroke="DarkRed" Fill="Black" Data="F1M5,7.47150993347168L5,17.2566661834717
              5.732421875,19.0242443084717 7.5,19.7566661834717 12.5,19.7566661834717
13.69140625,19.4441661834717
              5,7.47150993347168z M7.5,4.75666618347168L6.30859375,5.06916618347168 15,17.0418224334717
              15,7.25666618347168 14.267578125,5.48908805847168 12.5,4.75666618347168
7.5,4.75666618347168z
              M7.5,2.25666618347168L12.5,2.25666618347168 14.4189453125,2.62287712097168
              16.03515625,3.72150993347168 17.1337890625,5.33772134780884 17.5,7.25666618347168
              17.5,17.2566661834717 17.1337890625,19.1756114959717 16.03515625,20.7918224334717
              14.4189453125,21.8904552459717 12.5,22.2566661834717 7.5,22.2566661834717
              5.5810546875,21.8904552459717 3.96484375,20.7918224334717 2.8662109375,19.1756114959717
              2.5,17.2566661834717 2.5,7.25666618347168 2.8662109375,5.33772134780884
              3.96484375,3.72150993347168
              5.5810546875,2.62287712097168 7.5,2.25666618347168z" />
```

Service Layer

نقش لایه‌ی سرویس این است که به عنوان یک مدخل ورودی به برنامه کاربردی عمل کند. در برخی مواقع این لایه را به عنوان لایه‌ی Facade نیز می‌شناسند. این لایه، داده‌ها را در قالب یک نوع داده ای قوی (Strongly Typed) به نام View Model ، برای لایه‌ی Presentation فراهم می‌کند. کلاس View Model یک Strongly Typed محسوب می‌شود که نماهای خاصی از داده‌ها را که متفاوت از دید یا نمای تجاری آن است، بصورت بهینه ارائه می‌نماید. در مورد الگوی View Model در مباحث بعدی بیشتر صحبت خواهم کرد.

الگوی Facade یک Interface ساده را به منظور کنترل دسترسی به مجموعه ای از Interface ها و زیر سیستم‌های پیچیده ارائه می‌کند. در مباحث بعدی در مورد آن بیشتر صحبت خواهم کرد.

کلاسی با نام ProductViewModel را با کد زیر به پروژه SoCPatterns.Layered.Service اضافه کنید:

```
public class ProductViewModel
{
    Public int ProductId {get; set;}
    public string Name { get; set; }
    public string Rrp { get; set; }
    public string SellingPrice { get; set; }
    public string Discount { get; set; }
    public string Savings { get; set; }
}
```

برای اینکه کلاینت با لایه‌ی سرویس در تعامل باشد باید از الگوی Request/Response Message استفاده کنیم. بخش Request توسط کلاینت تغذیه می‌شود و پارامترهای مورد نیاز را فراهم می‌کند. کلاسی با نام ProductListRequest را با کد زیر به پروژه SoCPatterns.Layered.Service اضافه کنید:

```
using SoCPatterns.Layered.Model;

namespace SoCPatterns.Layered.Service
{
    public class ProductListRequest
    {
        public CustomerType CustomerType { get; set; }
    }
}
```

در شی Response نیز بررسی می‌کنیم که درخواست به درستی انجام شده باشد، داده‌های مورد نیاز را برای کلاینت فراهم می‌کنیم و همچنین در صورت عدم اجرای صحیح درخواست، پیام مناسب را به کلاینت ارسال می‌نماییم. کلاسی با نام ProductListResponse را با کد زیر به پروژه SoCPatterns.Layered.Service اضافه کنید:

```
public class ProductListResponse
{
    public bool Success { get; set; }
```

```
    public string Message { get; set; }
    public IList<ProductViewModel> Products { get; set; }
}
```

به منظور تبدیل موجودیت Product به ProductViewModel ، به دو متد نیاز داریم، یکی برای تبدیل یک Product و دیگری برای تبدیل لیستی از Product . شما می‌توانید این دو متد را به کلاس Product موجود در Domain Model اضافه نمایید، اما این متدها نیاز واقعی منطق تجاری نمی‌باشند. بنابراین بهترین انتخاب، استفاده از Extension Method ها می‌باشد که باید برای کلاس Product و در لایه‌ی سرویس ایجاد نمایید. کلاسی با نام ProductMapperExtensionMethods را با کد زیر به پروژه SoCPatterns.Layered.Service اضافه کنید:

```
public static class ProductMapperExtensionMethods
{
    public static ProductViewModel ConvertToProductViewModel(this Model.Product product)
    {
        ProductViewModel productViewModel = new ProductViewModel();
        productViewModel.ProductId = product.Id;
        productViewModel.Name = product.Name;
        productViewModel.RRP = String.Format("{0:C}", product.Price.RRP);
        productViewModel.SellingPrice = String.Format("{0:C}", product.Price.SellingPrice);
        if (product.Price.Discount > 0)
            productViewModel.Discount = String.Format("{0:C}", product.Price.Discount);
        if (product.Price.Savings < 1 && product.Price.Savings > 0)
            productViewModel.Savings = product.Price.Savings.ToString("#%");
        return productViewModel;
    }
    public static IList<ProductViewModel> ConvertToProductListViewModel(
        this IList<Model.Product> products)
    {
        IList<ProductViewModel> productViewModels = new List<ProductViewModel>();
        foreach(Model.Product p in products)
        {
            productViewModels.Add(p.ConvertToProductViewModel());
        }
        return productViewModels;
    }
}
```

حال کلاس ProductService را جهت تعامل با کلاس سرویس موجود در Domain Model و به منظور برگرداندن لیستی از محصولات و تبدیل آن به لیستی از ProductViewModel ، ایجاد می‌نماییم. کلاسی با نام ProductService را با کد زیر به پروژه SoCPatterns.Layered.Service اضافه کنید:

```
public class ProductService
{
    private Model.ProductService _productService;
    public ProductService(Model.ProductService ProductService)
    {
        _productService = ProductService;
    }
    public ProductListResponse GetAllProductsFor(
        ProductListRequest productListRequest)
    {
        ProductListResponse productListResponse = new ProductListResponse();
        try
        {
            IList<Model.Product> productEntities =
                _productService.GetAllProductsFor(productListRequest.CustomerType);
            productListResponse.Products = productEntities.ConvertToProductListViewModel();
            productListResponse.Success = true;
        }
        catch (Exception ex)
        {
            // Log the exception…
            productListResponse.Success = false;
            // Return a friendly error message
```

```
            productListResponse.Message = ex.Message;
        }
        return productListResponse;
    }
}
```

کلاس Service تمامی خطاها را دریافت نموده و پس از مدیریت خطا، پیغامی مناسب را به کلاینت ارسال می‌کند. همچنین این لایه محل مناسبی برای Log کردن خطاها می‌باشد. در اینجا کد نویسی لایه سرویس به پایان رسید و در ادامه به کدنویسی Data Layer می‌پردازیم.

Data Layer

برای ذخیره سازی محصولات، یک بانک اطلاعاتی با نام Shop01 ایجاد کنید که شامل جدولی به نام Product با ساختار زیر باشد:

Product

Column Name	Data Type	Allow Nulls
ProductId	int	☐
ProductName	nvarchar(50)	☐
Rrp	smallmoney	☐
SellingPrice	smallmoney	☐
		☐

برای اینکه کدهای بانک اطلاعاتی را سریعتر تولید کنیم از روش Linq to SQL در Data Layer استفاده می‌کنم. برای این منظور یک Data Context برای Linq to SQL به این لایه اضافه می‌کنیم. بر روی پروژه SoCPatterns.Layered.Repository کلیک راست نمایید و گزینه Add > New Item را انتخاب کنید. در پنجره ظاهر شده و از سمت چپ گزینه Data و سپس از سمت راست گزینه Linq to SQL Classes را انتخاب نموده و نام آن را Shop.dbml تعیین نمایید.

از طریق پنجره Server Explorer به پایگاه داده مورد نظر متصل شوید و با عمل Drag & Drop جدول Product را به بخش Design کشیده و رها نمایید.

اگر به یاد داشته باشید، در لایه Model برای برقراری ارتباط با پایگاه داده از یک Interface به نام IProductRepository استفاده نمودیم. حال باید این Interface را پیاده سازی نماییم. کلاسی با نام ProductRepository را با کد زیر به پروژه SoCPatterns.Layered.Repository اضافه کنید:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using SoCPatterns.Layered.Model;

namespace SoCPatterns.Layered.Repository
{
    public class ProductRepository : IProductRepository
    {
        public IList<Model.Product> FindAll()
        {
            var products = from p in new ShopDataContext().Products
                           select new Model.Product
                           {
                               Id = p.ProductId,
                               Name = p.ProductName,
                               Price = new Model.Price(p.Rrp, p.SellingPrice)
                           };
            return products.ToList();
        }
    }
}
```

در متد FindAll ، با استفاده از دستورات Linq to SQL ، لیست تمامی محصولات را برگرداندیم. کدنویسی لایه‌ی Data هم به پایان رسید و در ادامه به کدنویسی لایه‌ی Presentation و UI می‌پردازیم.

Presentation Layer

به منظور جداسازی منطق نمایش (Presentation) از رابط کاربری (User Interface) ، از الگوی Model View Presenter یا همان MVP استفاده می‌کنیم که در مباحث بعدی با جزئیات بیشتری در مورد آن صحبت خواهم کرد. یک Interface با نام IProductListView را با کد زیر به پروژه SoCPatterns.Layered.Presentation اضافه کنید:

```
using SoCPatterns.Layered.Service;

public interface IProductListView
{
    void Display(IList<ProductViewModel> Products);
    Model.CustomerType CustomerType { get; }
    string ErrorMessage { set; }
}
```

این Interface توسط Web Form های ASP.NET و یا Win Form ها باید پیاده سازی شوند. کار با Interface ها موجب می‌شود تا تست View ها به راحتی انجام شوند. کلاسی با نام ProductListPresenter را با کد زیر به پروژه SoCPatterns.Layered.Presentation اضافه کنید:

```
using SoCPatterns.Layered.Service;

namespace SoCPatterns.Layered.Presentation
{
    public class ProductListPresenter
    {
        private IProductListView _productListView;
        private Service.ProductService _productService;
        public ProductListPresenter(IProductListView ProductListView,
            Service.ProductService ProductService)
        {
            _productService = ProductService;
            _productListView = ProductListView;
        }
        public void Display()
        {
            ProductListRequest productListRequest = new ProductListRequest();
            productListRequest.CustomerType = _productListView.CustomerType;
            ProductListResponse productResponse =
                _productService.GetAllProductsFor(productListRequest);
            if (productResponse.Success)
            {
                _productListView.Display(productResponse.Products);
            }
            else
            {
                _productListView.ErrorMessage = productResponse.Message;
            }
        }
    }
}
```

کلاس Presenter وظیفه‌ی واکشی داده ها، مدیریت رویدادها و بروزرسانی UI را دارد. در اینجا کدنویسی لایه‌ی Presentation به پایان رسیده است. از مزایای وجود لایه‌ی Presentation این است که تست نویسی مربوط به نمایش داده‌ها و تعامل بین کاربر و سیستم به سهولت انجام می‌شود بدون آنکه نگران دشواری Unit Test نویسی Web Form ها باشید. حال می‌توانید کد نویسی مربوط به UI را انجام دهید که در ادامه به کد نویسی در Win Forms و Web Forms خواهیم پرداخت.

نظرات خوانندگان

نویسنده: محسن
تاریخ: ۱۳۹۲/۰۱/۰۲ ۱۸:۲۹

ممنون از زحمات شما.
چند سؤال و نظر:
- با تعریف الگوی مخزن به چه مزیتی دست پیدا کردید؟ برای مثال آیا هدف این است که کدهای پیاده سازی آن، با توجه به وجود اینترفیس تعریف شده، شاید روزی با مثلا NHibernate تعویض شود؟ در عمل متاسفانه حتی پیاده سازی LINQ این‌ها هم متفاوت است و من تابحال در عمل ندیدم که ORM یک پروژه بزرگ رو عوض کنند. یعنی تا آخر و تا روزی که پروژه زنده است با همان انتخاب اول سر می‌کنند. یعنی شاید بهتر باشه قسمت مخزن و همچنین سرویس یکی بشن.
- چرا لایه سرویس تعریف شده از یک یا چند اینترفیس مشتق نمی‌شود؟ اینطوری تهیه تست برای اون ساده‌تر میشه. همچنین پیاده سازی‌ها هم وابسته به یک کلاس خاص نمی‌شن چون از اینترفیس دارن استفاده می‌کنند.
- این اشیاء Request و Response هم در عمل به نظر نوعی ViewModel هستند. درسته؟ اگر اینطوره بهتر یک مفهوم کلی دنبال بشه تا سردرگمی‌ها رو کمتر کنه.

یک سری نکته جانبی هم هست که می‌تونه برای تکمیل بحث جالب باشه:
- مثلا الگوی Context per request بجای نوشتن new ShopDataContext بهتر استفاده بشه تا برنامه در طی یک درخواست در یک تراکنش و اتصال کار کنه.
- در مورد try/catch و استفاده از اون بحث زیاد هست. خیلی‌ها توصیه می‌کنن که یا اصلا استفاده نکنید یا استفاده از اون‌ها رو به بالاترین لایه برنامه موکول کنید تا این وسط کرش یک قسمت و بروز استثناء در اون، از ادامه انتشار صدمه به قسمت‌های بعدی جلوگیری کنه.

نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۲/۰۱/۰۲ ۲۳:۳۵

محسن عزیز. از شما ممنونم که به نکته‌های ظریفی اشاره کردید.

در سری مقالات اولیه فقط دارم یک دید کلی به کسایی میدم که تازه دارن با این مفاهیم آشنا میشن. این پروژه اولیه دستخوش تغییرات زیادی میشه. در واقع محصول نهایی این مجموعه مقالات بر پایه همین نوع لایه بندی ولی بادید و طراحی مناسب‌تر خواهد بود.
در مورد ORM هم من با چند Application سروکار داشتم که در روال توسعه بخش‌های جدید رو بنا به دلایلی با ORM یا DB متفاوتی توسعه داده اند. غیر از این موضوع، حتی بخشهایی از مدل، سرویس و یا مخزن رو در پروژه‌های دیگری استفاده کرده اند. همچنین برخی از نکات مربوط به تفکیک لایه‌ها به منظور تست پذیری راحت‌تر رو هم در نظر بگیرید.
در مورد اشیا Request و Response هم باید خدمتتان عرض کنم که برای درخواست و پاسخ به درخواست استفاده می‌شوند که چون پروژه ای که مثال زدم کوچک بوده ممکنه کاملا درکش نکرده باشید. ما کلاسهای Request و Response متعددی در پروژه داریم که ممکنه خیلی از اونها فقط از یک View Model استفاده کنن ولی پارامترهای ارسالی یا دریافتی آنها متفاوت باشد.
در مورد try...catch هم من با شما کاملا موافقم. به دلیل هزینه ای که دارد باید در آخرین سطح قرار بگیرد. در این مورد ما میتونیم اونو به Presentation و یا در MVC به Controller منتقل کنیم.
در مورد DbContext هم هنوز الگویی رو معرفی نکردم. در واقع هنوز وارد جزئیات لایه‌ی Data نشدم. در مورد اون اگه اجازه بدی بعدا صحبت میکنم.

نویسنده: ایلیا
تاریخ: ۱۳۹۲/۰۱/۰۳ ۰:۴۳

آقای خوشبخت خداقوت.

مرسی از مطالب خوبتون.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۰۳ ۰:۴۸

لطفا برای اینکه نظرات حالت فنی‌تر و غنای بیشتری پیدا کنند، از ارسال پیام‌های تشکر خودداری کنید. برای ابراز احساسات و همچنین تشکر، لطفا از گزینه رای دادن به هر مطلب که ذیل آن قرار دارد استفاده کنید.
این مطلب تا این لحظه 76 بار دیده شده، اما فقط 4 رای دارد. لطفا برای ابراز تشکر، امتیاز بدهید. ممنون.

نویسنده: محسن
تاریخ: ۱۳۹۲/۰۱/۰۳ ۱:۰

- من در عمل تفاوتی بین لایه مخزن و سرویس شما مشاهده نمی‌کنم. یعنی لایه مخزن داره GetAll می‌کنه، بعد لایه سرویس هم داره همون رو به یک شکل دیگری بر می‌گردونه. این تکرار کد نیست؟ این دو یکی نیستند؟

عموما در منابع لایه مخزن رو به صورت روکشی برای دستورات مثلا EF یا LINQ to SQL معرفی می‌کنند. فرضشون هم این است که این روش ما رو از تماس مستقیم با ORM برحذر می‌داره (شاید فکر می‌کنند ایدز می‌گیرند اگر مستقیم کار کنند!). ولی عرض کردم این روکش در واقعیت فقط شاید با EF یا L2S قابل تعویض باشه نه با ORMهای دیگر با روش‌های مختلف و بیشتر یک تصور واهی هست که جنبه عملی نداره. بیشتر تئوری هست بدون پایه تجربه دنیای واقعی. ضمن اینکه این روکش باعث میشه نتونید از خیلی از امکانات ORM مورد استفاده درست استفاده کنید. مثلا ترکیب کوئری‌ها یا روش‌های به تاخیر افتاده و امثال این.

- پس در عمل شما Request ViewModel و Response ViewModel تعریف کردید.

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۰۳ ۱۲:۲۷

سپاس از سری مطالبی که منتشر می‌کنید.
-پیشنهادی که من دارم اینه که لایه‌ی Repository حذف شود ، همانطور که در مطالب قبلی ذکر شده DbSet در Entity Framework همان پیاده سازی الگوی مخزن هست و ایجاد Repository جدید روی آن یک Abstraction اضافه هست. در نتیجه اگر Repository حذف شود همه‌ی منطق‌ها مانند GetBlaBla به Service منتقل می‌شود.
-یک پیشنهاد دیگر اینکه استفاده از کلمه‌ی New در Presentation Layer را به حداقل رساند و همه جا نیاز مندی‌ها را به صورت وابستگی به کلاس‌های استفاده کننده تزریق شود تا در زمان نوشتن تست‌ها همه‌ی اجزاء قابل تعویض با Mock objects باشند.

نویسنده: افشین
تاریخ: ۱۳۹۲/۰۱/۰۶ ۱۱:۱۵

لطفا دمو یا سورس برنامه رو هم قرار بدید که یادگیری و آموزش سریعتر انجام بشه.

ممنون

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۱/۱۰ ۰:۱۱

با سلام از کار بزرگی که دارین می‌کنین سپاس
یک سوال؟
جای الگوی Unit Of Work در این پروژه کجا میشه؟

در این پست جناب آقای مهندس نصیری در لایه سرویس الگوی واحد کار را پیاده کرده اند، با توجه به وجود الگوی Repository

در پروژه شما ممنون میشم شرح بیشتری بدین که جایگاه پیاده سازی الگو واحد کار با توجه به مزایایی که دارد در کدام لایه است؟

نویسنده: رام
تاریخ: ۱۳۹۲/۰۱/۱۶ ۵:۲۹

محسن جان، چیزی که من از این الگو در مورد واکشی و نمایش داده‌ها برداشت میکنم اینه:

کلاس‌های لایه مخزن با دریافت دستور از لایه سرویس آبجکت مدل مربوطه را پر میکنند و به بالا (لایه سرویس) پاس میدند.

بعد

در لایه سرویس نمونه‌ی مدل مربوطه به ویـــــومدل متناظر باهاش تبدیل میشه و به لایه بالاتر فرستاده میشه

بنابراین
کار در **"لایه مخزن"** روی **"مدل ها"** انجام میگیره
و
کار در **"لایه سرویس"** روی **"ویـــــومدل ها"** انجام میشه

نتیجه: لایه سرویس هدف دیگری را نسبت به لایه مخزن دنبال میکند و این هدف آنقدر بزرگ و مهم هست که برایش یه لایه مجزا در نظر گرفته بشه

نویسنده: رام
تاریخ: ۱۳۹۲/۰۱/۱۶ ۵:۴۹

شاهین جان، من با حذف لایه مخزن مخالف هستم. زیرا:

ما لایه ای به نام "لایه مخزن" را میسازیم تا در نهایت کلیه متدهایی که برای حرف زدن با داده هامون را نیاز داریم داشته باشیم. حالا این اطلاعات ممکنه از پایگاه داده یا جاهای دیگه جمع آوری بشوند (و الزاما توسط EF قابل دسترسی و ارائه نباشند)

همچنین گاهی نیاز هست که بر مبنای چند متد که EF به ما میرسونه (مثلا چند SP) یک متد کلی‌تر را تعریف کنیم (چند فراخوانی را در یک متد مثلا متد X در لایه مخزن انجام دهیم) و در لایه بالاتر آن متد را صدا بزنیم (بجای نوشتن و تکرار پاپی همه کدهای نوشت شده در متد X)

علاوه بر این در لایه مخزن میشه چند ORM را هم کنار هم دید (نه فقط EF) که همونطور که آقای خوشبخت در کامنت‌ها نوشتند گاهی نیاز میشه.

بنابراین:
من وجود لایه مخزن را ضروری میدونم.

(فراموش نکنیم که هدف از این آموزش تعریف یک الگوی معماری مناسب برای پروژه‌های بزرگ هست و الا بدون خیلی از اینها هم میشه برنامه ساخت. همونطور که اکثرا بدون این ساختارها و خیلی ساده‌تر میسازند)

نویسنده: محسن
تاریخ: ۱۳۹۲/۰۱/۱۶ ۹:۳

- بحث آقای شاهین و من در مورد مثال عینی بود که زده شد. در مورد کار با ORM که کدهاش دقیقا ارائه شده. این روش قابل نقد و رد است.

شما الان اومدی یک بحث انتزاعی کلی رو شروع کردید. بله. اگر ORM رو کنار بگذارید مثلا می‌رسید به ADO.NET (یک نمونه که خیلی‌ها در این سایت حداقل یکبار باهاش کار کردن). این افراد پیش از اینکه این مباحث مطرح باشن برای خودشون لایه DAL داشتند و تمام جزئیات ADO.NET رو کپسوله کرده بودن در اون. حالا با اومدن ORMها این لایه DAL کنار رفته چون خود ORM هست که کپسوله کننده ADO.NET است. همین‌ها هم یک لایه دیگر داشتند به نام BLL که از لایه DAL استفاده می‌کرد برای پیاده سازی منطق تجاری برنامه. این لایه الان اسمش شده لایه سرویس.

یعنی تمام مواردی رو که عنوان کردید در مورد ADO.NET صدق می‌کنه. یکی اسمش رو می‌ذاره DAL شما اسمش رو گذشتید Repository. ولی این مباحث ربطی به یک ORM تمام عیار که کپسوله کننده ADO.NET است ندارد.

- ترکیب چند SP در لایه مخزن انجام نمیشه. چیزی رو که عنوان کردید یعنی پیاده سازی منطق تجاری و این مورد باید در لایه سرویس باشه. اگر از ADO.NET استفاده میشه، می‌تونیم با استفاده از DAL جزئیات دسترسی به SP رو مخفی و ساده‌تر کنیم با کدی یک دست‌تر در تمام برنامه. اگر از EF استفاده می‌کنیم، باز همین ساده سازی در طی فراخوانی فقط یک متد انجام شده. بنابراین بهتر است وضعیت و سطح لایه‌ای رو که داریم باهاش کار می‌کنیم خوب بررسی و درک کنیم.

- می‌تونید در عمل در بین پروژه‌های سورس باز و معتبر موجود فقط یک نمونه رو به من ارائه بدید که در اون از 2 مورد ORM مختلف همزمان استفاده شده باشه؟ این مورد یعنی سؤ مدیریت. یعنی پراکندگی و انجام کاری بسیار مشکل مثلا یک نمونه: ORMها لایه‌ای دارند به نام سطح اول کش که مثلا در EF اسمش هست Trackig API. این لایه فقط در حین کار با Context همون ORM کار می‌کنه. اگر دو مورد رو با هم مخلوط کنید، قابل استفاده نیست، ترکیب پذیر نیستند. از این دست باز هم هست مثلا در مورد نحوه تولید پروکسی‌هایی که برای lazy loading تولید می‌کنند و خیلی از مسایل دیگری از این دست. ضمن اینکه مدیریت چند Context فقط در یک لایه خودش یعنی نقض اصل تک مسئولیتی کلاس‌ها.

نویسنده: محسن
تاریخ: ۱۳۹۲/۰۱/۱۶ ۹:۱۵

سعی نکنید انتزاعی بحث کنید. چون در این حالت این حرف می‌تونه درست باشه یا حتی نباشه. اگر از ADO.NET استفاده می‌کنید، درسته. اگر از EF استفاده می‌کنید غلط هست. لازم هست منطق کار با ADO.NET رو یک سطح کپسوله کنیم. چون از تکرار کد جلوگیری می‌کنه و نهایتا به یک کد یک دست خواهیم رسید. لازم نیست اعمال یک ORM رو در لایه‌ای به نام مخزن کپسوله کنیم، چون خودش کپسوله سازی ADO.NET رو به بهترین نحوی انجام داده. برای نمونه در همین مثال عینی بالا به هیچ مزیتی نرسیدیم. فقط یک تکرار کد است. فقط بازی با کدها است.

نویسنده: رام
تاریخ: ۱۳۹۲/۰۱/۱۶ ۱۶:۴۶

من منظور شما را خوب متوجه میشم ولی حرفام یه بحث انتزاعی نیست چون پروژه عملی زیر دستم دارم که توی اون هم با پر کردن View Model کار میکنم.

مشکل از اینجا شروع میشه که شما فکر میکنید همیشه مدل ای که در EF ساختید را باید بدون تغییر در ساختارش به پوسته برنامه برسونید و از پوسته هم دقیقا نمونه ای از همون را بگیرید و به لایه‌های پایین بفرستید ولی یکی از مهمترین کارهای View Model اینه که این قانون را از این سفتی و سختی در بیاره چون خیلی مواقع هست که شما در پوسته برنامه به شکل دیگه ای از داده‌ها (متفاوت با اونچه در Model تعریف کردید و EF باهاش کار میکنه) نیاز دارید. مثلا فیلد تاریخ از نوع DateTime در Model و نوع String در پوسته و یا حتی اضافه و کم کردن فیلدهای یک Model و ایجاد ساختارهای متفاوتی از اون برای عملیات‌های Select, Update و Delete. لذا لایه سرویس قرار نیست فقط همون کار لایه مخزن را تکرار کنه (به قول شما GetAll). بلکه در زمان لزوم تغییرات لازم که نام بردم را هم رووش اعمال میکنه (که به نظر من آقای خوشبخت هم به خوبی از کلمه Convert در لایه سرویس استفاده کردند.)

اما بحث اینکه ما در لایه مخزن روی EF یک سطح کپسوله میسازیم جای گفتگو داره هرچند من در اون مورد هم با وجد لایه مخزن بیشتر موافقم تا گفتگوی مستقیم لایه سرویس با چیزی مثل EF

نتیجه: فرقی نمیکنه شما از Asp.Net استفاده میکنید یا هر ORM مورد نظرتون. کلاس‌های مدل باید در ارتباط با لایه بالاتر خودشون به ویـــو مدل تبدیل بشند و در این الگو این کار در لایه سرویس انجام میشه.

نویسنده: محسن
تاریخ: ۱۳۹۲/۰۱/۱۶ ۱۷:۱۰

- پیاده سازی الگوی مخزن در عمل (بر اساس بحث فعلی که در مورد کار با ORMها است) به صورت کپسوله سازی ORM در همه جا مطرح میشه و اینکار اساسا اشتباه هست. چون هم شما رو محروم می‌کنه از قابلیت‌های پیشرفته ORM و هم ارزش افزوده‌ای رو به همراه نداره. دست آخر می‌بینید در لایه مخزن GetAll دارید در لایه سرویس هم GetAll دارید. این مساله هیچ مزیتی نداره. یک زمانی در ADO.NET برای GetAll کردن باید کلی کد شبیه به کدهای یک ORM نوشته می‌شد. خود ORM الان اومده این‌ها رو کپسوله کرده و لایه‌ای هست روی اون. اینکه ما مجددا یک پوسته روی این بکشیم حاصلی نداره بجز تکرار کد. عده‌ای عنوان می‌کنند که حاصل اینکار امکان تعویض ORM رو ممکن می‌کنه ولی این‌ها هم بعد از یک مدت تجربه با ORMهای مختلف به این نتیجه می‌رسند که ای بابا! حتی پیاده سازی LINQ این ORMها یکی نیست چه برسه به قابلیت‌های پیشرفته‌ای که در یکی هست در دوتای دیگر نیست (واقع بینی، بجای بحث تئوری محض).

- اینکه این تبدیلات (پر کردن ViewModel از روی مدل) هم می‌تونه و بهتره که (نه الزاما) در لایه سرویس انجام بشه، نتیجه مناسبی هست.

UI

در نهایت نوبت به طراحی و کدنویسی UI می‌رسد تا بتوانیم محصولات را به کاربر نمایش دهیم. اما قبل از شروع باید موضوعی را یادآوری کنم. اگر به یاد داشته باشید، در کلاس ProductService موجود در لایه‌ی Domain ، از طریق یکی از روشهای الگوی Dependency Injection به نام Constructor Injection ، فیلدی از نوع IProductRepository را مقداردهی نمودیم. حال زمانی که بخواهیم نمونه ای را از ProductService ایجاد نماییم، باید به عنوان پارامتر ورودی سازنده، شی ایجاد شده از جنس کلاس ProductRepository موجود در لایه Repository را به آن ارسال نماییم. اما از آنجایی که می‌خواهیم تفکیک پذیری لایه‌ها از بین نرود و UI بسته به نیاز خود، نمونه مورد نیاز را ایجاد نموده و به این کلاس ارسال کند، از ابزارهایی برای این منظور استفاده می‌کنیم. یکی از این ابزارها StructureMap می‌باشد که یک Inversion of Control Container یا به اختصار IoC Container نامیده می‌شود. با Inversion of Control در مباحث بعدی بیشتر آشنا خواهید شد. StructureMap ابزاری است که در زمان اجرا، پارامترهای ورودی سازنده‌ی کلاسهایی را که از الگوی Dependency Injection استفاده نموده اند و قطعا پارامتر ورودی آنها از جنس یک Interface می‌باشد را، با ایجاد شی مناسب مقداردهی می‌نماید.

به منظور استفاده از StructureMap در Visual Studio 2012 باید بر روی پروژه UI خود کلیک راست نموده و گزینه‌ی Manage NuGet Packages را انتخاب نمایید. در پنجره ظاهر شده و از سمت چپ گزینه‌ی Online و سپس در کادر جستجوی سمت راست و بالای پنجره واژه‌ی structuremap را جستجو کنید. توجه داشته باشید که باید به اینترنت متصل باشید تا بتوانید Package مورد نظر را نصب نمایید. پس از پایان عمل جستجو، در کادر میانی structuremap ظاهر می‌شود که می‌توانید با انتخاب آن و فشردن کلید Install آن را بر روی پروژه نصب نمایید.

جهت آشنایی بیشتر با NuGet و نصب آن در سایر نسخه‌های Visual Studio می‌توانید به لینک‌های زیر رجوع کنید:

1. آشنایی با NuGet قسمت اول

2. آشنایی با NuGet قسمت دوم

3. Installing NuGet

کلاسی با نام BootStrapper را با کد زیر به پروژه UI خود اضافه کنید:

```
using StructureMap;
using StructureMap.Configuration.DSL;
using SoCPatterns.Layered.Repository;
using SoCPatterns.Layered.Model;

namespace SoCPatterns.Layered.WebUI
{
    public class BootStrapper
    {
        public static void ConfigureStructureMap()
        {
            ObjectFactory.Initialize(x => x.AddRegistry<ProductRegistry>());
        }
    }
```

```
    public class ProductRegistry : Registry
    {
        public ProductRegistry()
        {
            For<IProductRepository>().Use<ProductRepository>();
        }
    }
}
```

ممکن است یک WinUI ایجاد کرده باشید که در این صورت به جای فضای نام SoCPatterns.Layered.WebUI از SoCPatterns.Layered.WinUI استفاده نمایید.

هدف کلاس BootStrapper این است که تمامی وابستگی‌ها را توسط StructureMap در سیستم Register نماید. زمانی که کدهای کلاینت می‌خواهند به یک کلاس از طریق StructureMap دسترسی داشته باشند، Structuremap وابستگی‌های آن کلاس را تشخیص داده و بصورت خودکار پیاده سازی واقعی (Concrete Implementation) آن کلاس را، براساس همان چیزی که ما برایش تعیین کردیم، به کلاس تزریق می‌نماید. متد ConfigureStructureMap باید در همان لحظه ای که Application آغاز به کار می‌کند فراخوانی و اجرا شود. با توجه به نوع UI خود یکی از روالهای زیر را انجام دهید:

در WebUI :

فایل Global.asax را به پروژه اضافه کنید و کد آن را بصورت زیر تغییر دهید:

```
namespace SoCPatterns.Layered.WebUI
{
    public class Global : System.Web.HttpApplication
    {
        protected void Application_Start(object sender, EventArgs e)
        {
            BootStrapper.ConfigureStructureMap();
        }
    }
}
```

در WinUI :

در فایل Program.cs کد زیر را اضافه کنید:

```
namespace SoCPatterns.Layered.WinUI
{
    static class Program
    {
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
```

BootStrapper.ConfigureStructureMap();

```
Application.Run(new Form1());
        }
    }
}
```

سپس برای طراحی رابط کاربری، با توجه به نوع UI خود یکی از روالهای زیر را انجام دهید:

در WebUI :

صفحه Default.aspx را باز نموده و کد زیر را به آن اضافه کنید:

```
<asp:DropDownList AutoPostBack="true" ID="ddlCustomerType" runat="server">
    <asp:ListItem Value="0">Standard</asp:ListItem>
    <asp:ListItem Value="1">Trade</asp:ListItem>
</asp:DropDownList>
<asp:Label ID="lblErrorMessage" runat="server" ></asp:Label>
<asp:Repeater ID="rptProducts" runat="server" >
    <HeaderTemplate>
        <table>
            <tr>
                <td>Name</td>
                <td>RRP</td>
                <td>Selling Price</td>
                <td>Discount</td>
                <td>Savings</td>
            </tr>
            <tr>
                <td colspan="5"><hr /></td>
            </tr>
    </HeaderTemplate>
    <ItemTemplate>
            <tr>
                <td><%# Eval("Name") %></td>
                <td><%# Eval("RRP")%></td>
                <td><%# Eval("SellingPrice") %></td>
                <td><%# Eval("Discount") %></td>
                <td><%# Eval("Savings") %></td>
            </tr>
    </ItemTemplate>
    <FooterTemplate>
        </table>
    </FooterTemplate>
</asp:Repeater>
```

در WinUI :

فایل Form1.Designer.cs را باز نموده و کد آن را بصورت زیر تغییر دهید:

```
#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.cmbCustomerType = new System.Windows.Forms.ComboBox();
    this.dgvProducts = new System.Windows.Forms.DataGridView();
    this.colName = new System.Windows.Forms.DataGridViewTextBoxColumn();
    this.colRrp = new System.Windows.Forms.DataGridViewTextBoxColumn();
    this.colSellingPrice = new System.Windows.Forms.DataGridViewTextBoxColumn();
    this.colDiscount = new System.Windows.Forms.DataGridViewTextBoxColumn();
    this.colSavings = new System.Windows.Forms.DataGridViewTextBoxColumn();
    ((System.ComponentModel.ISupportInitialize)(this.dgvProducts)).BeginInit();
    this.SuspendLayout();
    //
    // cmbCustomerType
    //
    this.cmbCustomerType.DropDownStyle =
System.Windows.Forms.ComboBoxStyle.DropDownList;
    this.cmbCustomerType.FormattingEnabled = true;
    this.cmbCustomerType.Items.AddRange(new object[] {
        "Standard",
        "Trade"});
    this.cmbCustomerType.Location = new System.Drawing.Point(12, 90);
```

```
    this.cmbCustomerType.Name = "cmbCustomerType";
    this.cmbCustomerType.Size = new System.Drawing.Size(121, 21);
    this.cmbCustomerType.TabIndex = 3;
    //
    // dgvProducts
    //
    this.dgvProducts.ColumnHeadersHeightSizeMode =
System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize;
    this.dgvProducts.Columns.AddRange(new System.Windows.Forms.DataGridViewColumn[] {
    this.colName,
    this.colRrp,
    this.colSellingPrice,
    this.colDiscount,
    this.colSavings});
    this.dgvProducts.Location = new System.Drawing.Point(12, 117);
    this.dgvProducts.Name = "dgvProducts";
    this.dgvProducts.Size = new System.Drawing.Size(561, 206);
    this.dgvProducts.TabIndex = 2;
    //
    // colName
    //
    this.colName.DataPropertyName = "Name";
    this.colName.HeaderText = "Product Name";
    this.colName.Name = "colName";
    this.colName.ReadOnly = true;
    //
    // colRrp
    //
    this.colRrp.DataPropertyName = "Rrp";
    this.colRrp.HeaderText = "RRP";
    this.colRrp.Name = "colRrp";
    this.colRrp.ReadOnly = true;
    //
    // colSellingPrice
    //
    this.colSellingPrice.DataPropertyName = "SellingPrice";
    this.colSellingPrice.HeaderText = "Selling Price";
    this.colSellingPrice.Name = "colSellingPrice";
    this.colSellingPrice.ReadOnly = true;
    //
    // colDiscount
    //
    this.colDiscount.DataPropertyName = "Discount";
    this.colDiscount.HeaderText = "Discount";
    this.colDiscount.Name = "colDiscount";
    //
    // colSavings
    //
    this.colSavings.DataPropertyName = "Savings";
    this.colSavings.HeaderText = "Savings";
    this.colSavings.Name = "colSavings";
    this.colSavings.ReadOnly = true;
    //
    // Form1
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
    this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
    this.ClientSize = new System.Drawing.Size(589, 338);
    this.Controls.Add(this.cmbCustomerType);
    this.Controls.Add(this.dgvProducts);
    this.Name = "Form1";
    this.Text = "Form1";
    ((System.ComponentModel.ISupportInitialize)(this.dgvProducts)).EndInit();
    this.ResumeLayout(false);
}
#endregion
private System.Windows.Forms.ComboBox cmbCustomerType;
private System.Windows.Forms.DataGridView dgvProducts;
private System.Windows.Forms.DataGridViewTextBoxColumn colName;
private System.Windows.Forms.DataGridViewTextBoxColumn colRrp;
private System.Windows.Forms.DataGridViewTextBoxColumn colSellingPrice;
private System.Windows.Forms.DataGridViewTextBoxColumn colDiscount;
private System.Windows.Forms.DataGridViewTextBoxColumn colSavings;
```

سپس در Code Behind ، با توجه به نوع UI خود یکی از روالهای زیر را انجام دهید:

در WebUI :

وارد کد نویسی صفحه Default.aspx شده و کد آن را بصورت زیر تغییر دهید:

```
using System;
using System.Collections.Generic;
using SoCPatterns.Layered.Model;
using SoCPatterns.Layered.Presentation;
using SoCPatterns.Layered.Service;
using StructureMap;

namespace SoCPatterns.Layered.WebUI
{
    public partial class Default : System.Web.UI.Page, IProductListView
    {
        private ProductListPresenter _productListPresenter;
        protected void Page_Init(object sender, EventArgs e)
        {
            _productListPresenter = new
ProductListPresenter(this,ObjectFactory.GetInstance<Service.ProductService>());
            this.ddlCustomerType.SelectedIndexChanged +=
                delegate { _productListPresenter.Display(); };
        }
        protected void Page_Load(object sender, EventArgs e)
        {
            if(!Page.IsPostBack)
                _productListPresenter.Display();
        }
        public void Display(IList<ProductViewModel> products)
        {
            rptProducts.DataSource = products;
            rptProducts.DataBind();
        }
        public CustomerType CustomerType
        {
            get { return (CustomerType) int.Parse(ddlCustomerType.SelectedValue); }
        }
        public string ErrorMessage
        {
            set
            {
                lblErrorMessage.Text =
                    String.Format("<p><strong>Error:</strong><br/>{0}</p>", value);
            }
        }
    }
}
```

در WinUI :

وارد کدنویسی Form1 شوید و کد آن را بصورت زیر تغییر دهید:

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using SoCPatterns.Layered.Model;
using SoCPatterns.Layered.Presentation;
using SoCPatterns.Layered.Service;
using StructureMap;

namespace SoCPatterns.Layered.WinUI
{
    public partial class Form1 : Form, IProductListView
    {
        private ProductListPresenter _productListPresenter;
        public Form1()
        {
            InitializeComponent();
            _productListPresenter =
                new ProductListPresenter(this, ObjectFactory.GetInstance<Service.ProductService>());
            this.cmbCustomerType.SelectedIndexChanged +=
                delegate { _productListPresenter.Display(); };
            dgvProducts.AutoGenerateColumns = false;
```

```
            cmbCustomerType.SelectedIndex = 0;
        }
        public void Display(IList<ProductViewModel> products)
        {
            dgvProducts.DataSource = products;
        }
        public CustomerType CustomerType
        {
            get { return (CustomerType)cmbCustomerType.SelectedIndex; }
        }
        public string ErrorMessage
        {
            set
            {
                MessageBox.Show(
                    String.Format("Error:{0}{1}", Environment.NewLine, value));
            }
        }
    }
}
```

با توجه به کد فوق، نمونه ای را از کلاس ProductListPresenter ، در لحظه‌ی نمونه سازی اولیه‌ی کلاس UI ، ایجاد نمودیم. با استفاده از متد ObjectFactory.GetInstance مربوط به StructureMap ، نمونه ای از کلاس ProductService ایجاد شده است و به سازنده‌ی کلاس ProductListPresenter ارسال گردیده است. در مورد Structuremap در مباحث بعدی با جزئیات بیشتری صحبت خواهم کرد. پیاده سازی معماری لایه بندی در اینجا به پایان رسید.

اما اصلا نگران نباشید، شما فقط پرواز کوتاه و مختصری را بر فراز کدهای معماری لایه بندی داشته اید که این فقط یک دید کلی را به شما در مورد این معماری داده است. این معماری هنوز جای زیادی برای کار دارد، اما در حال حاضر شما یک Applicaion با پیوند ضعیف (Loosely Coupled) بین لایه‌ها دارید که دارای قابلیت تست پذیری قوی، نگهداری و پشتیبانی آسان و تفکیک پذیری قدرتمند بین اجزای آن می‌باشد. شکل زیر تعامل بین لایه‌ها و وظایف هر یک از آنها را نمایش می‌دهد.

نظرات خوانندگان

نویسنده: حامد
تاریخ: ۱۳۹۲/۰۱/۰۳ ۱۴:۲۹

ممنون از مقاله خوبتون
به نظر شما امکانش هست برای این معماری یک generator بسازیم به طوری که فقط ما تمام جداول دیتابیس و رابطه‌ی آنها را بسازیم و بعد این generator از روی اون تمام لایه‌ها را بر اساس آن بسازه و بعد ما صرفا جاهایی که نیاز به جزییات داره را کامل کنیم
آیا نمونه ای از این برنامه‌ها هست که این معماری یا معماری‌های مشابه را بسازه؟

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۰۳ ۱۵:۳۱

اگر با T4 آشنایی داشته باشید بر اساس هر قالبی می‌توانید کد تولید کنید.

نویسنده: حامد
تاریخ: ۱۳۹۲/۰۱/۰۳ ۱۶:۳۶

متاسفانه آشنایی ندارم میشه یه توضیح مختصر بدین و یا منبع معرفی کنید

نویسنده: محسن
تاریخ: ۱۳۹۲/۰۱/۰۳ ۲۳:۵۹

چون اینجا بحث طراحی مطرح شده یک اصل رو در برنامه‌های وب باید رعایت کرد:

هیچ وقت متن خطای حاصل رو به کاربر نمایش ندید (از لحاظ امنیتی). فقط به ذکر عبارت خطایی رخ داده بسنده کنید. خطا رو مثلا توسط ELMAH لاگ کنید برای بررسی بعدی برنامه نویس.

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۰۴ ۱:۲۰

http://codepanic.blogspot.com/2012/03/t4-enum.html

نویسنده: M.Q
تاریخ: ۱۳۹۲/۰۱/۰۴ ۲۲:۱۵

دوست عزیز غیر از ELMAH ابزار دیگری برای لاگ گیری از خطاها وجود دارد که قابل اعتماد باشد؟

همچنین اگر ابزاری جهت لاگ گیری از عملیات کاربران (CRUD => حالا R خیلی مهم نیست) می‌شناسید معرفی نمائید.

با سپاس

نویسنده: محسن
تاریخ: ۱۳۹۲/۰۱/۰۵ ۰:۳۳

متد auditFields مطرح شده در مطلب ردیابی اطلاعات این سایت برای مقصود شما مناسب است.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۱/۱۲ ۱۴:۲۳

سلام با تشکر از شما
من نفهمیدم که توی ASP.NET MVC شما چگونه از الگوی MVP استفاده کردین؟
ظاهرا مثال این قسمت هم توی پست وجود نداره، اگر اشتباه می‌کنم لطفا تصحیح بفرمایید.

نویسنده: علي
تاریخ: ۱۳۹۲/۰۱/۱۲ ۱۶:۳

مثال وب فرم هست. post back و page load داره.

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۱۲ ۱۶:۴

اگر توجه کنید از الگوی MVP در Web Forms استفاده شده و نه در MVC.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۱/۱۲ ۱۸:۳۰

آقای کیاست و علی آقا
می‌دونم که پروژه چی هست، یکی از UIهای ما قرار بود MVC باشه خواستم بدونم چطور می‌خوان استفاده کنن، اینجا (در این پست) که می‌دونم ASP.NET Web form هست و در MVC می‌دونم که Page_Load و.. وجود نداره سوال من چیز دیگه بود دوستان

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۱۲ ۱۸:۴۴

شما گفتید:

سلام با تشکر از شما
من نفهمیدم که توی ASP.NET MVC شما چگونه از الگوی MVP استفاده کردین؟
ظاهرا مثال این قسمت هم توی پست وجود نداره، اگر اشتباه می‌کنم لطفا تصحیح بفرمایید.
با خواندن کامنت شما برداشت کردم شما تصور کردید کدهای پست جاری مربوط به تکنولوژی ASP.NET MVC هست.

به نظر نویسنده هنوز برای MVC و WPF مثال‌ها را ایجاد نکرده و توضیح نداده اند.
اما برای استفاده از این نوع معماری در MVC کار خاصی لازم نیست انجام شود. همانطور که قبلا در مثال‌های آقای نصیری دیده ایم کافی است Service Layer در Controller مدل مناسب را تغذیه کند و برای View فراهم کند.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۱/۱۲ ۱۹:۲۶

من هم با توجه به مثال آقای نصیری و استفاده از الگوی کار گیج شدم، این معماری یک لایه Repository دارد، من الگوی کار توی این لایه پیاده کردم، با پیاده سازی در این لایه به نظر میاد لایه سرویس کاربردش از دست میده توی پست‌های قبل هم از آقای خوشبخت سوال کردم اما طاهرا هنوز وقت نکردن پاسخ بدن.

مورد دوم اینکه در این پست الگوی کار شرح داده شده و پیاده سازی شده، و در این پست گفته شده " *حین استفاده از EF code first، الگوی واحد کار، همان DbContext است و الگوی مخزن، همان DbSetها. ضرورتی به ایجاد یک لایه محافظ اضافی بر روی*

این‌ها وجود ندارد. " با توجه به این مسائل کلا مسائل قاطی کردم متاسفانه آقای نصیری هم سرشون شلوغ و درگیر دوره ها است، که بحثی بر سر این معماری بشه.

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۱۲ ۲۰:۴۶

روشی که در مثال آقای نصیری گفته شده با روش این سری مقالات کمی متفاوت هست.
در آنجا از روکش اضافه برای Repository استفاده نشده همچنین از الگوی واحد کار استفاده شده.
به علاوه این سری مقالات ممکن است هنوز تکمیل نشده باشند.
به نظر من هر کس با توجه به میزان اطلاعاتی که دارد و درکی که از الگوها دارد با مقایسه‌ی روش‌ها و مقالات می‌تواند تصمیم بگیرد چه معماری به کار بگیرد.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۱/۱۲ ۲۱:۳

حرف شما کاملا متین هست

من قبلا معماری سه لایه کار می‌کردم، که نمونه اون توی همین سایت بخش پروژه ها گذاشتم، اما الان با EF , MVC کمی به مشکل بر خوردم و درست نتونستم تا حالا لایه‌های مورد نظر برای خودم در پروژه‌ها تفکیک کنم، این معماری به نظرم جالب اومد، خواستم که الگوی کار هم توی اون به کار ببرم که به مشکل بر خوردم (چون درک درستی از الگوی کار پیدا نکردم یا شایدم کلا دارم اشتباه می‌کنم). البته به قول شما شاید این معماری هنوز تکمیل نشده پروژه اش، در هر صورت از پاسخ‌های شما متشکرم.

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۱۲ ۲۱:۷

خواهش می‌کنم.
فقط جهت یادآوری مثال روش آقای نصیری با پوشش MVC و EF قابل دریافت است.

نویسنده: ابوالفضل روشن ضمیر
تاریخ: ۱۳۹۲/۰۱/۱۷ ۱:۴۵

سلام
با تشکر فروان از شما ...
اگر امکان داره این مثال که در قالی یک پروژه نوشته شده برای دانلود قرار دهید ...تا بهتر بتوانیم برنامه را تجزیه و تحلیل کنیم ....ممنون

نویسنده: فرشید علی اکبری
تاریخ: ۱۳۹۲/۰۱/۱۹ ۱۵:۵۳

با سلام و تشکر از زحمات کلیه دوستان
با زحمتی که آقای خوشبخت تا اینجا کشیدن فکر کنم در صورتیکه خودمون مقاله مربوطه به این پروژه رو قدم به قدم بخونیم وطراحی کینم خیلی بهتر متوجه میشیم تا اینکه اونو آماده دانلود کنیم. من با این روش پیش رفتم و برای ایجاد اون با step by step کردن مراحلش حدود 45 دقیقه وقت گذاشتم ولی درصد یادگیریش خیلی بالاتر بود تا گرفتن فایل آماده...
درضمن لازمه بگم که بخاطر رفع شک وشبهه در سرعت پردازش وبالا اومدن اطلاعات، من تست این روش رو با تعداد 155 هزار رکورد انجام دادم که کمتر از سه ثانیه برام لود شد... باوجودیکه کامپوننت‌های دات نت بار مختلفی رو هم روی فرمم قرار دادم که بیشتر به اهمیت لود اطلاعاتم در پروژه و فرمهای واقعی پی ببرم.
سئوال اینکه :

به نظر شما ما می‌تونیم روی این لایه‌ها الگوی واحد کار رو هم ایجاد کنیم یا نه؟ اصلا ضرورتی داره ؟

نویسنده: علیرضا کیانی مقدم
تاریخ: ۱۳۹۲/۰۱/۲۸ ۱۲:۸

با تشکر از نویسنده مقاله و اهتمام ایشان به بررسی دقیق مفاهیم ،
از آنجا که flexible و reusable بودن برنامه‌ها را نمی‌توان نادیده گرفت تا آنجا که این تفکیک پذیری خود به مسئله ای بغرنج تبدیل نشده و تکرر داده‌ها و پاس دادن غیر ضرور آنها را موجب نشود تلاش در این باره مفید خواهد بود .
امروزه توسعه دهنده گان به سمت کم کردن لایه‌های فرسایشی و حذف پیچیدگی‌های غیر ضرور قدم بر می‌دارند. خلق عبارات لامبادا در دات نت و delegate ها نمونه هایی از تلاش بشر برنامه نویس در این باره است .

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۰۲/۰۹ ۹:۱۲

سلام
Business Rule-1ها و validation-2ها در کجای این معماری اعمال میشوند؟

منظور از DomainService چیست؟

ممکنه منابع بیشتری معرفی نمایید؟
ممنون.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۰۹ ۱۶:۲۶

منبع برای مطالعه بیشتر

عنوان: **مقیدسازی (DataBinding) در WPF زمانی که دسترسی به DataContext وجود ندارد**
نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۰۳/۰۲ ۱۴:۵
آدرس: www.dotnettips.info
برچسب‌ها: MVVM, WPF

در WPF و Silverlight می‌توان با استفاده از مقید سازی (DataBinding) کنترل‌ها را به منبع‌های داده متصل کرد. این منابع به چند شیوه مختلف مانند استفاده مستقیم از خصوصیت Source قابل دسترسی هستند. یکی از این روش ها، ارث بری از DataContext نزدیک‌ترین والد است.

همانطور که گفته شد DataContext هر کنترل، توسط تمامی فرزندان آن قابل دسترسی است. اما در بعضی مواقع، زمانیکه کنترل فرزند، بخشی از visual یا logical tree نباشند، دسترسی به DataContext وجود ندارد.

برای مثال زمانی که نیاز است خصوصیت ItemsSource مربوط به یک به لیستی خارج از ItemsSource کنترل DataGrid DataGridTemplateColumn مثلا به لیستی درون ViewModel مربوط به Window در مثال زیر مقید شود، به صورت معمول باید به این صورت عمل کرد:

ViewModel :

```
public List<People> ComboBoxDataSource{get; set;}
```

XAML :

```
<Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow"
        x:Name="this">
    <Grid>
        <DataGrid ItemsSource="{Binding DataCollection}">
            <DataGrid.Columns>
                <DataGridComboBoxColumn ItemsSource="{Binding DataContext.ComboBoxDataSource,
ElementName=this}"/>
            </DataGrid.Columns>
        </DataGrid>
    </Grid>
</Window>
```

با اینکه همه چیز درست به نظر می‌رسد اما در عمل هیچ اتصالی صورت نمی‌گیرد و در پنجره Output ویژوال استادیو خطای زیر مشاهده می‌شود:

```
System.Windows.Data Error: 2 : Cannot find governing FrameworkElement or FrameworkContentElement for
target element.
BindingExpression:Path=ComboBoxDataSource; DataItem=null;
target element is 'DataGridComboBoxColumn' (HashCode=17334644); target property is 'ItemsSource' (type
'IEnumerable')
```

این خطا مشخص میکند که WPF نمیتواند تشخیص بدهد که کدام FrameWorkElement قرار است از DataContext استفاده کند؛ چرا که همانطور که قبلا عنوان شد DataGridTemplateColumn بخشی از visual یا logical tree نیست.

برای مشکل فوق در صورتیکه خصوصیت مورد نظر، یک خصوصیت از فرزندان کنترل باشد، از طریق استایل‌ها می‌توان مشکل را حل کرد. برای مثال به جای ItemSource مربوط به DataGridComboBoxColumn می‌توان خصوصیت ItemSource کنترل ComboBox درون آن را تنظیم کرد.

```
<DataGridComboBoxColumn DisplayMemberPath="FirstName">
        <DataGridComboBoxColumn.EditingElementStyle>
              <Style TargetType="ComboBox">
                     <Setter Property="ItemsSource" Value="{Binding DataContext.ComboBoxDataSource ,
ElementName=this}"/>
                 </Style>
         </DataGridComboBoxColumn.EditingElementStyle>
   </DataGridComboBoxColumn>
```

اما در صورتیکه نیاز باشد یک خصوصیت از خود DataGridComboBoxColumn مانند Visibility مقید سازی شود، روش بالا کارساز نخواهد بود. برای حل مشکل فوق میتوان از کلاس‌های Freezable استفاده کرد؛ چرا که این کلاسها می‌توانند از DataContext ارث بری کنند حتی زمانیکه بخشی از visual یا logical tree نباشند. برای این کار می‌توان کلاس زیر را ایجاد کرد:

```
public class DataBindingHelper : Freezable
    {
        protected override Freezable CreateInstanceCore()
        {
            return new DataBindingHelper();
        }
        public object Data
        {
            get { return (object)GetValue(DataProperty); }
            set { SetValue(DataProperty, value); }
        }

        public static readonly DependencyProperty DataProperty =
            DependencyProperty.Register("Data", typeof(object), typeof(DataBindingHelper), new
UIPropertyMetadata(null));
    }
```

و یک نمونه از آن را در Resourceهای DataGrid ساخت:

```
<DataGrid.Resources>
       <local:DataBindingHelper x:Key="bindingHelper"Data="{Binding}"/>
</DataGrid.Resources>
```

و هنگام مقید سازی خصوصیت Visibility مربوط به DataGridComboBoxColumn، از نمونه ساخته شده به عنوان Source استفاده نمود.

```
<DataGridComboBoxColumn Visibility="{Binding Data.IsVisible,Converter={StaticResource
visibilityConverter},Source={StaticResource bindingHelper}}"/>
```

عنوان: **فراخوانی یک متداز یک کنترل WPF از XAML**
نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۰۳/۱۲ ۲۳:۰
آدرس: www.dotnettips.info
برچسب‌ها: WPF

در بعضی مواقع نیاز است که یک متد از یک کنترل درون XAML فراخوانی شود. برای مثال لازم است یکی از متدهای یک کنترل در یک استایل فراخوانی شود. یکی از روش‌های انجام این کار استفاده از خصوصیت‌های پیوست شده (AttachedPropery) است. شیوه‌ی کار به این صورت است که یک خصوصیت از نوع Bool ایجاد می‌کنیم. هنگامیکه مقدار این خصوصیت تغییر کند یک رویه فراخوانی می‌شود که کار فراخوانی متد مورد نظر را انجام میدهد:

```
public class SelectAllBehavior
    {
        public static bool GetSelectAll(TextBoxBase target)
        {
            return (bool)target.GetValue(SelectAllProperty);
        }

        public static void SetSelectAll(TextBoxBase target, bool value)
        {
            target.SetValue(SelectAllProperty, value);
        }

        public static readonly DependencyProperty SelectAllProperty =
DependencyProperty.RegisterAttached("SelectAll", typeof(bool), typeof(SelectAllBehavior), new
UIPropertyMetadata(false, OnSelectAllPropertyChanged));

        static void OnSelectAllPropertyChanged(DependencyObject o, DependencyPropertyChangedEventArgs
e)
        {
            ((TextBoxBase)o).SelectAll();
        }
    }
```

برای استفاده از کلاس فوق درون یک استایل باید به شکل زیر عمل کرد:

```
<Style TargetType="{x:Type TextBoxBase}" >
    <Style.Triggers>
        <Trigger Property="IsFocused" Value="True">
            <Setter Property="x: SelectAllBehavior.SelectAll" Value="True"/>
        </Trigger>
    </Style.Triggers>
</Style>
```

در تکه کد بالا ،هنگامی که خصوصیت IsFocused مربوط به کنترل TextBox برابر True می‌شود، یعنی Focus روی TextBox قرار می‌گیرد،مقدار خصوصیت پیوست شده نیز برابر True می‌شود که همانطور که گفته شد باعث فراخوانی OnSelectAllPropertyChanged می‌شود.

تا اینجا فراخوانی یک متد از کنترل از طریق استایل توضیح داده شد، همانطور که در عنوان مطلب آورده شده است. اما اگر بخواهید مثال فوق را به درستی اجرا کنید یعنی هنگام کلیک روی TextBox متن درون آن انتخاب شود، نیاز به اضافه کردن کدهای دیگری وجود دارد. چرا که به صورت پیش فرض زمانی که MouseLeftButtonUp اجرا می‌شود در صورتی که حالت متن به صورت انتخاب شده باشد، از حالت انتخاب خارج میشود. این بار برای اینکار از خصوصیت‌های پیوست شده (AttachedPropery) برای کنترل رویداد PreviewMouseLeftButtonDown استفاده میکنیم و هنگام فشرده شدن کلید سمت چپ موس رویدادهای Click و بقیه LeftMouseButtonDownها را غیرفعال می‌کنیم.

```
public class PreviewMouseLeftButtonDownBehavior
{
    public static readonly DependencyProperty PreviewMouseLeftButtonDownProperty =
DependencyProperty.RegisterAttached("PreviewMouseLeftButtonDown"
                  , typeof(bool?)
                  , typeof(PreviewMouseLeftButtonDownBehavior)
                  , new UIPropertyMetadata(null, OnPreviewMouseLeftButtonDown)
```

```
                  );

    public static void SetPreviewMouseLeftButtonDown(DependencyObject target, bool? value)
    {
        target.SetValue(PreviewMouseLeftButtonDownProperty, value);
    }
    public static object GetPreviewMouseLeftButtonDown(DependencyObject target)
    {
        return target.GetValue(PreviewMouseLeftButtonDownProperty);
    }

    public static void OnPreviewMouseLeftButtonDown(DependencyObject obj,
DependencyPropertyChangedEventArgs e)
    {
        var control = obj as Control;
        if (control == null)
            return;
        if (e.NewValue != null && e.OldValue == null)
            control.PreviewMouseLeftButtonDown += control_PreviewMouseLeftButtonUp;
        else if ((e.NewValue == null) && (e.OldValue != null))
        {
            control.PreviewMouseLeftButtonDown -= control_PreviewMouseLeftButtonUp;
        }
    }
    static void control_PreviewMouseLeftButtonUp(object sender,
System.Windows.Input.MouseButtonEventArgs e)
    {
        var tb = (sender as TextBox);
        if (tb != null)
        {
            if (!tb.IsKeyboardFocusWithin)
            {
                e.Handled = true;
                tb.Focus();
            }
        }
    }
}
```

و استایل را به صورت زیر تغییر میدهیم:

```
<Style TargetType="{x:Type TextBoxBase}" >
    <Setter Property="x:PreviewMouseLeftButtonDownBehavior.IsPreviewMouseLeftButtonDown" Value="True"/>
    <Style.Triggers>
        <Trigger Property="IsKeyboardFocusWithin" Value="True">
            <Setter Property="InputLanguageManager.InputLanguage" Value="fa-ir" />
        </Trigger>
        <Trigger Property="IsFocused" Value="True">
            <Setter Property="xamlServices:SelectAllTextBoxBehavior.SelectAll" Value="True"/>
        </Trigger>
    </Style.Triggers>
</Style>
```

در این مثال با Focus رو هر TextBox که استایل فوق را به کار گرفته باشد، متن در حالت انتخاب شده قرار میگیرد. (البته این مشروط به این است که نیاز نباشد رویداد PreviewMouseLeftButtonDown کنترل مورد نظر درون برنامه مقیدسازی(Bind) شود.)

عنوان: آشنایی با Catel MVVM Frameowork
نویسنده: محمّد زارع
تاریخ: ۱۳۹۲/۰۳/۱۳ ۱۶:۱۵
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF, Catel

در این مقاله به بررسی اولیه فریمورک Catel و برخی ویژگی‌های آن خواهیم پرداخت.
همانطور که می‌دانید فریمورک‌های متعددی برای MVVM به وجود آمده اند، مانند MVVM Light یا Caliburn و Chinch و ... که هر کدام از آن‌ها دارای ویژگی هایی می‌باشند اما Catel تنها یک فریمورک برای MVVM نیست بلکه دارای قسمت‌های دیگری مانند کنترل‌های اختصاصی و سرویس‌های متعدد و پرکاربرد و Extensionهای مفید و ... نیز می‌باشد که کار توسعه یک برنامه MVVM را فوق العاده لذتبخش می‌کند.
برای شروع کار با این فریمورک ابتدا بایستی قالب پروژه را از این آدرس دریافت نمایید . بعد از دریافت و نصب آن یک زیرگروه جدید به نام Catel به قسمت افزودن پروژه جدید اضافه خواهد شد که شامل قالب پروژه برای WPF و Silverlight و Windows Phone و Windows Store می باشد. در این قسمت گزینه WPF Application with Catel را انتخاب نمایید و پروژه را ایجاد کنید.
بعد از ایجاد پروژه نوبت به نصب بسته های nuget مورد نیاز Catel می رسد. تنها بسته مورد نیاز Catel.Extensions.Controls می باشد که به صورت خودکار بسته های Catel.MVVM و Catel.Core را نیز نصب خواهد کرد. البته بسته‌های دیگری مانند Catel.Extensions.Prism, Catel.Extensions.FluentValidation و Catel.Extensions.Data و Catel.Fody و ... نیز برای این فریمورک وجود دارد که در این مطلب به آن‌ها نیازی نداریم.
اکنون ساختار اصلی پروژه ما ایجاد شده است. در این ساختار پوشه‌های Views، Models و ViewModels به صورت پیش فرض وجود دارند. Catel برای برقراری ارتباط بین View و ViewModel از IViewModelLocator ، IViewLocator و یکسری قواعد نام گذاری پیروی میکند تا نیاز به رجیستر کردن تک تک ویوها و ویومدل‌ها به صورت دستی نباشد که البته این قواعد قابل تغییر و شخصی سازی هستند. قرارداد پیش فرض برای پروژه‌های کوچک ممکن است مناسب باشد ولی در پروژه‌های بزرگ نیاز به سفارشی سازی دارد که در قسمت‌های بعد به آن خواهیم پرداخت.

View و ViewModel:

برای ایجاد یک ViewModel جدید، باید از منوی Add New Item قسمت Catel گزینه ViewModel (Catel) را انتخاب نمایید. با توجه به code snippet های تهیه شده برای این فریمورک، کار تهیه ViewModelها فوق العاده سریع انجام می‌شود. به عنوان مثال برای اضافه کردن یک Command در ویومدل، از vmcommand و یا vmcommandwithcanexecute و برای ایجاد پروپرتی هم از vmprop و vmpropchanged میتوان استفاده نمود. همانطور که ملاحظه می‌کنید نام این snippetها کاملا واضح می‌باشد و نیاز به توضیح اضافی ندارند.
همینطور برای ایجاد یک View گزینه DataWindow (WPF with Catel) را انتخاب نمایید. ViewModelها در Catel از کلاس پایه ViewModelBase و Viewها نیز از کلاس DataWindow مشتق می‌شوند.
DataWindow یک Window پیشرفته با قابلیت هایی مانند افزودن خودکار دکمه‌های Ok / Cancel یا Ok / Cancel / Apply یا Close می‌باشد که می‌تواند باعث تسریع روند ایجاد Windowهای تکراری شود. اما اگر به هیچ کدام از این دکمه‌های ذکر شده نیاز نداشتید DataWindowMode.Custom را انتخاب می‌کنید. نشان دادن Validation در بالای پنجره به صورت popup نیز یکی دیگر از قابلیت‌های این Window پیشرفته است. البته DataWindow دارای overloadهای مختلفی است که می‌توانید به کمک آن ویژگی‌های ذکر شده را فعال یا غیر فعال کنید.
حال برای درک بهتر commandها و نحوه تعریف و بکارگیری آن‌ها یک command جدید در MainWindowViewModel با استفاده از vmcommand ایجاد کنید. مانند قطعه کد زیر:

```
public class MainWindowViewModel : ViewModelBase
    {
        public MainWindowViewModel()
            : base()
        {
            ShowPleaseWait = new Command(OnShowPleaseWaitExecute);
        }

        public override string Title { get { return "View model title"; } }

        public Command ShowPleaseWait { get; private set; }
        private void OnShowPleaseWaitExecute()
        {
            var pleaseWaitService = GetService<IPleaseWaitService>();
```

```
            pleaseWaitService.Show(() =>
            {
                Thread.Sleep(3000);
            });
        }
    }
```

در داخل بدنه این command از PleaseWaitService استفاده کردیم که در ادامه توضیح داده خواهد شد. در MainView نیز یک button اضافه کنید و پروپرتی Command آن را به صورت زیر تنظیم کنید:

```
<Button Margin="6"
                Command="{Binding ShowPleaseWait}"
                Content="Show PleaseWait!" />
```

اکنون با فشردن button کد داخل بدنه command اجرا خواهد شد.

سرویس ها:

کتابخانه Catel.MVVM دارای سرویس‌های مختلف و پرکاربردی می‌باشد که در ادامه به بررسی آن‌ها خواهیم پرداخت:

PleaseWaitService: از این سرویس برای نشان دادن یک loading به کاربر در حین انجام یک کار سنگین استفاده می‌شود و نحوه استفاده از آن به صورت زیر است:

```
var pleaseWaitService = GetService<IPleaseWaitService>();
pleaseWaitService.Show(() =>
{
        Thread.Sleep(3000);
});
```

UIVisualizerService: از این سرویس برای باز کردن پنجره‌های برنامه استفاده می‌شود. هر View در برنامه دارای یک ViewModel می باشد. برای باز کردن View ابتدا یک نمونه از ViewModel مربوطه را ایجاد میکنیم و با دادن viewmodel به متد Show یا ShowDialog پنجره مورد نظر را باز میکنیم.

```
var uiService = GetService<IUIVisualizerService>();
var viewModel = new AnotherWindowViewModel();
uiService.Show(viewModel);
```

OpenFileService: برای نشان دادن OpenFileDialog جهت باز کردن یک فایل در برنامه.

```
var openFileService = GetService<IOpenFileService>();
openFileService.Filter = "ZIP files (*.zip)|*.zip";
openFileService.IsMultiSelect = false;
openFileService.Title = "Open file";
if (openFileService.DetermineFile())
{
       // ?
}
```

SaveFileService: برای نشان دادن SaveFileDialog جهت ذخیره سازی.

```
var saveFileService = GetService<ISaveFileService>();
saveFileService.Filter = "ZIP files (*.zip)|*.zip";
saveFileService.FileName = "test";
saveFileService.Title = "Save file";
if (saveFileService.DetermineFile())
{
       // ?
```

```
}
```

ProcessService: برای اجرا کردن یک process. به عنوان مثال برای باز کردن ماشین حساب ویندوز به صورت زیر عمل می‌کنیم:

```
var processService = GetSetvice<IProcessService>();
processService.StartProcess(@"C:\Windows\System32\calc.exe");
```

SplashScreenService: برای نشان دادن SplashScreen در ابتدای برنامه هایی که سرعت بالا آمدن پایینی دارند.

```
var splashScreenService = GetService<ISplashScreenService>();
splashScreenService.Enqueue(new ActionTask("Creating the shell", OnCreateShell));
splashScreenService.Enqueue(new ActionTask("Initializing modules", OnInitializeModules));
splashScreenService.Enqueue(new ActionTask("Starting application", OnStartApplication));
```

MessageService: برای نشان دادن MessageBox به کاربر.

```
var messageService = GetService<IMessageService>();
if (messageService.Show("Are you sure?", "?", MessageButton.YesNo, MessageImage.Warning) ==
MessageResult.Yes)
{
	// ?
}
```

همانطور که ملاحظه کردید اکثر کارهای مورد نیاز یک پروژه با کمک سرویس‌های ارائه شده در این فریمورک به آسانی انجام می‌شود.

دریافت مثال و پروژه کامل این قسمت: TestApp.zip

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 Bootstrapper 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 پروژه‌های 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 پروژه‌های 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
نخاب ماژول
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	WpfApplication۱
TargetPath	E:\Workspace\Projects\XLIFFProject\XLIFFProject\bin\Release\WpfApplication۱.exe
ProjectPath	E:\Workspace\Projects\XLIFFProject\XLIFFProject\XLIFFProject.csproj
ProjectFileName	XLIFFProject.csproj
TargetExt	.exe
TargetFileName	WpfApplication۱.exe
DevEnvDir	C:\Program Files (x۸۶)\Microsoft Visual Studio ۱۰,۰\Common۷\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	x۸۶
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 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: **استفاده از ItemsControl جهت ساختن کنترل های پویا در WPF**
نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۰۴/۰۶ ۱۴:۵
آدرس: www.dotnettips.info
برچسب‌ها: WPF

از ItemsControl برای ارائه مجموعه ای از کنترل‌ها استفاده می‌شود،در اینجا قرار است از آن استفاده کنیم و یک کنترل پویا ایجاد کنیم.برای مثال در نظر بگیرید،قرار است یک DropDown Panel ایجاد کنیم و در جاهای مختلف برنامه کنترل‌های مختلفی را درون آن قرار بدهیم.برای ایجاد آن به صورت زیر عمل میکنیم:

```
<UserControl x:Class="MySystem.Common.Controls.DropDownPanel"
             …
             x:Name="This">
    <Grid>
        <ToggleButton x:Name="ShowPopupButton"/>
        <Popup
            PlacementTarget="{Binding ElementName=ShowPopupButton}"
            Placement="{Binding PopupPlacement, ElementName=this}"
            PopupAnimation="Slide"
            AllowsTransparency="True"
            Focusable="True"
            StaysOpen="False"
            IsOpen="{Binding IsChecked, ElementName=ShowPopupButton }">
            <Border Background="#FFE3EAF3" BorderThickness="1" Padding="2">
                <Grid>
                    <ItemsControl ItemsControl.ItemsSource="{Binding Path=PanelItems,ElementName=This
}">
                        <ItemsControl.ItemsPanel>
                            <ItemsPanelTemplate>
                                <Grid>

                                </Grid>
                            </ItemsPanelTemplate>
                        </ItemsControl.ItemsPanel>
                    </ItemsControl>
                </Grid>
            </Border>
        </Popup>
    </Grid>
</UserControl>
```

همانگونه که در کد بالا می‌بینید ،برای ایجاد DropDown Panel از یک ToggleButton و یک Popup که خصوصیت IsOpen آن به IsChecked مربوط به ToggleButton وصل شده است، استفاده کردیم و در قسمت بدنه کنترل به جای قراردان کنترل هایی که قرار است در آن نمایش داده شوند،از یک ItemsControl استفاده کردیم که خصوصیت ItemsSource آن به یک خصوصیت پیوست شده از نوع ObservableCollection<UIElement> در *Code Behind* ، مقید شده است.تعریف این خصوصیت پیوست شده به صورت زیر است:

```
public static readonly DependencyProperty PanelItemsProperty = DependencyProperty.Register("PanelItems"
            , typeof(ObservableCollection<UIElement>)
            , typeof(DropDownPanel)
            , new PropertyMetadata(new ObservableCollection<UIElement>()));
        public ObservableCollection<UIElement> PanelItems
        {
            get
            {
                return (ObservableCollection<UIElement>)GetValue(PanelItemsProperty);
            }
            set
            {
                SetValue(PanelItemsProperty, value);
            }
        }
```

برای استفاده از کنترل یک وهله از این کنترل را ایجاد می‌کنیم و کنترل هایی که قرار است در DropDown Panel نمایش داده شوندرا به PanelItems اضافه میکنیم:

```
<Window x:Class="MySystem.UI.View.Window1"
              ….
             xmlns:controls ="clr-
namespace:MySystem.Common.Controls;assembly=GoldAccountingSystem.Common.Controls">
    <Grid>
        <controls:DropDownPanel>
            <controls:DropDownPanel.PanelItems>
                    !--Put Controls Here--!
            </controls:DropDownPanel.PanelItems>
        </controls:DropDownPanel>
    </Grid>
</Window>
```

تا این مرحله کنترل مورد نظر را ایجاد و استفاده کردیم.اما یک مشکل وجود دارد،چنانچه از این کنترل چند بار در یک فرم استفاده شود، به درستی عمل نمیکند به اینصورت که فرزندان PanelItems تمام شی‌های ساخته شده از کنترل در یک فرم برابر هم و برابر مقداری می‌شود که برای آخرین کنترل قرارداده ایم. دلیل این امر این است که ما یکبار در هنگام تعریف خصوصیت PanelItems یک وهله از آن را به عنوان مقدار پیش فرض ایجاد کردیم و برای همه‌ی نمونه هایی از کنترل که در فرم قرار می‌گیرند از همان وهله استفاده می‌شود.

برای حل مشکل فوق یک کلاس از نوع ObservableCollection<UIElement> ایجاد کرده و هنگام ساختن کنترل در فرم از این کلاس برای وهله سازی مجدد از PanelItems استفاده می‌کنیم:

```
namespace MySystem.Common.Controls
{
    public class UIElementCollection : ObservableCollection<UIElement> { }
}
```

همانطور که گفته شد از کلاس ایجاد شده برای وهله سازی به صورت زیر استفاده می‌شود:

```
<Window x:Class="MySystem.UI.View.Window1"
              ….
             x:Name="This"
             xmlns:controls ="clr-
namespace:MySystem.Common.Controls;assembly=GoldAccountingSystem.Common.Controls">
    <Grid>
        <commonControls:DropDownPanel>
            <commonControls:DropDownPanel.PanelItems>
                <commonControls:UIElementCollection>
                    !--Put Controls Here--!
                </commonControls:UIElementCollection>
            </commonControls:DropDownPanel.PanelItems>
            </commonControls:DropDownPanel>
    </Grid>
</Window>
```

یک مثال ساده از توضیحات بالا را از آدرس روبرو می‌توانید دریافت نمایید: ItemsControlTest-a349625a156d4aeaaca288f000ae6d7a.rar

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۴/۰۶ ۱۸:۳۲

اگر امکانش باشه این مطلب رو به صورت یک مثال یا پروژه ساده قابل دریافت قرار بدید، امکان پیگیری اون بیشتر خواهد شد.
ممنون.

نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۰۴/۰۹ ۹:۷

سلام،ممنون از پیشنهادتون،یک مثال خیلی ساده به توضیحات بالا اضافه کردم.

عنوان: **ساخت فرم های generic در WPF و Windows Application**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۴/۰۹ ۸:۳۰
آدرس: www.dotnettips.info
برچسب‌ها: WPF, Generics

آیا می‌توان در یک پروژه های Windows App یا WPF، یک فرم پایه به صورت generic تعریف کنیم و سایر فرم‌ها بتوانند از آن ارث ببرند؟ در این پست به تشریح و بررسی این مسئله خواهیم پرداخت.

در پروژه هایی به صورت Smart UI کد نویسی شده اند و یا حتی قصد انجام پروژه با تکنولوژی‌های WPF یا Windows Application را دارید و نیاز دارید که فرم‌های خود را به صورت generic بسازید این مقاله به شما کمک خواهد کرد.

Windows Application#

یک پروژه از نوع Windows Application ایجاد می‌کنیم و یک فرم به نام FrmBase در آن خواهیم داشت. یک Label در فرم قرار دهید و مقدار Text آن را فرم اصلی قرار دهید.

در فرم مربوطه، فرم را به صورت generic تعریف کنید. به صورت زیر:

```
public partial class FrmBase<T> : Form where T : class
    {
        public FrmBase()
        {
            InitializeComponent();
        }
    }
```

بعد باید همین تغییرات را در فایل FrmBase.designer.cs هم اعمال کنیم:

```
partial class FrmBase<T> where T : class
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
        protected override void Dispose( bool disposing )
        {
            if ( disposing && ( components != null ) )
            {
                components.Dispose();
            }
            base.Dispose( disposing );
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.label1 = new System.Windows.Forms.Label();
            this.SuspendLayout();
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Location = new System.Drawing.Point(186, 22);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(51, 13);
            this.label1.TabIndex = 0;
            this.label1.Text = "فرم اصلی";
            //
            // FrmBase
```

```
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(445, 262);
            this.Controls.Add(this.label1);
            this.Name = "FrmBase";
            this.Text = "Form1";
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Label label1;
    }
```

یک فرم جدید بسازید و نام آن را FrmTest بگذارید. این فرم باید از FrmBase ارث ببرد. خب این کار را به صورت زیر انجام می‌دهیم:

```
public partial class FrmTest : FrmBase<String>
    {
        public FrmTest()
        {
            InitializeComponent();
        }
    }
```

پروژه را اجرا کنید. بدون هیچ گونه مشکلی برنامه اجرا می‌شود وفرم مربوطه را در حالت اجرا مشاهده خواهید کرد. اما اگر قصد باز کردن فرم FrmTest را در حالت design داشته باشید با خطای زیر مواجه خواهید شد:

با این که برنامه به راحتی اجرا می‌شود و خروجی آن قابل مشاهده است ولی امکان نمایش فرم در حالت design وجود ندارد. متاسفانه در Windows Appها برای تعریف فرم‌ها به صورت generic یا این مشکل روبرور هستیم. تنها راه موجود برای حل این مشکل استفاده از یک کلاس کمکی است. به صورت زیر:

```
public partial class FrmTest : FrmTestHelp
    {
        public FrmTest()
        {
            InitializeComponent();
        }
    }

    public class FrmTestHelp : FrmBase<String>
    {
    }
```

مشاهده می‌کنید که بعد از اعمال تغییرات بالا فرم FrmTest به راحتی Load می‌شود و در حالت designer هم می‌تونید از آن استفاده کنید.

WPF#

در پروژه‌های WPF ، راه حلی برای این مشکل در نظر گرفته شده است. در WPF، برای Window یا UserControl پایه نمی‌توان

Designer داشت. ابتدا باید فرم پایه را به صورت زیر ایجاد کنیم:

```
public class WindowBase<T> : Window where T : class
{
}
```

در این مرحله یک Window بسازید که از WindowBase ارث ببرد:

```
public partial class MainWindow: WindowBase<String>
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
```

در WPF باید تعاریف موجود برای Xaml و Code Behind یکی باشد. در نتیجه باید تغییرات زیر را در فایل Xaml نیز اعمال کنید:

```
<local:WindowBase x:Class="GenericWindows.MainWindow"

        x:TypeArguments="sys:String"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:GenericWindows"
        xmlns:sys="clr-namespace:System;assembly=mscorlib"
        Title="MainWindow" Height="350" Width="525">
    <Grid>

    </Grid>
</local:WindowBase>
```

همان طور که می‌بینید در ابتدای فایل به جای Window از local:WindowBase استفاده شده است. این نشان دهنده این است که فرم پایه برای این Window از نوع WindowBase است. برای مشخص کردن نوع generic هم می‌تونید از x:TypeArguments استفاده کنید که در این جا نوع آن را String انتخاب کردم.

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۴/۰۹ ۱۹:۶

با تشکر. لطفا یک مثال دنیای واقعی از این فرم جنریک بزنید.

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۴/۱۰ ۹:۳۷

یک مثال پیاده سازی شده رو می‌تونید (^) اینجا مشاهده کنید.

نویسنده: محمدی راوری
تاریخ: ۱۳۹۲/۰۵/۰۹ ۱۴:۲۴

با سلام و تشکر از آموزش ارائه شده
من در ساخت برنامه مشکلی نداشتم و اون رو ساختم اما در مرحله ای که لازم بود تا نمایش فرم ساخته شده را فعال کنم با مشکل برخوردم.
اگر ممکنه راهنمایی کنید؛ با سپاس فراوان.

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۵/۰۹ ۲۰:۲۶

مشکل در قسمت نمایش در حالت Design بوده است یا اجرا؟
اگر امکانش هست مشکل مربوطه را دقیق عنوان کنید.

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 ModuleIImpl 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

عنوان: **استفاده از F# در پروژه های WPF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۴/۱۷ ۸:۳۵
آدرس: www.dotnettips.info
برچسب‌ها: WPF, Programming, F#, FSharpx

در دوره F# این سایت (^) با نحوه کد نویسی و مفاهیم و مزایای این زبان آشنا شده اید. اما دانستن syntax یک زبان برای پیاده سازی یک پروژه کافی نیست و باید با تکنیک‌های مهم دیگر از این زبان آشنا شویم. همان طور که قبلا (فصل اول دوره F#) بیان شد Visual Studio به صورت Visual از پروژه‌های F# پشتیبانی نمی‌کند. یعنی امکان ایجاد یک پروژه WPF یا Windows Application یا حتی پروژه‌های تحت وب برای این زبان همانند زبان C# به صورت Visual در VS.Net تعبیه نشده است. حال چه باید کرد؟ آیا باید در این مواقع این گونه پروژه‌ها را با یک زبان دیگر نظیر C# ایجاد کنیم و از زبان F# در حل برخی مسائل محاسباتی و الگوریتمی استفاده کنیم. این اولین راه حلی است که به نظر می‌رسد. اما در حال حاضر افزونه هایی، توسط سایر تیم‌های برنامه نویسی تهیه شده اند که پیاده سازی و اجرای یک پروژه تحت ویندوز یا وب را به صورت کامل با زبان F# امکان پذیر می‌کنند. در این پست به بررسی یک مثال از پروژه WPF به کمک این افزونه‌ها می‌پردازیم.

نکته : آشنایی با کد نویسی و مفاهیم F# برای درک بهتر مطالب توصیه می‌شود.

معرفی پروژه FSharpX

پروژه FSharpx یک پروژه متن باز است که توسط یک تیم بسیار قوی از برنامه نویسان F# در حال توسعه می‌باشد. این پروژه شامل چندین زیر پروژه و بخش است که هر بخش آن برای یکی از مباحث دات نت در F# تهیه و توسعه داده می‌شود.

این قسمت‌ها عبارتند از :

FSharpx.Core : شامل مجموعه ای کامل از توابع عمومی، پرکاربرد و ساختاری است که برای این زبان توسعه داده شده اند و با تمام زبان‌های دات نت سازگاری دارند؛

FSharpx.Http : استفاده از F# در برنامه نویسی مدل Http؛

FSharpx.TypeProvider : این پروژه خود شامل چندین بخش است که در این جا چند مورد از آن‌ها را عنوان می‌کنم:

FSharpx.TypeProviders.AppSetting : متد خواندن و نوشتن (setter و getter) را برای فایل‌های تنظیمان پروژه (Application Setting File) فراهم می‌کند.

FSharpx.TypeProviders.Vector : برای محاسبات با ساختارهای برداری استفاده می‌شود.

FSharpx.TypeProviders.Machine : برای دسترسی و اعمال تغییرات در رجیستری و فایل‌های سیستمی استفاده می‌شود.

FSharpx.TypeProviders.Xaml : با استفاده از این افزونه می‌توانیم از فایل‌های Xaml، در پروژه‌های F# استفاده کنیم و WPF Designer نرم افزار VS.Net هم برای این زبان قابل استفاده خواهد شد.

FSharpx.TypeProviders.Regex : امکان استفاده از عبارات با قاعده را در این پروژه فراهم می‌کند.

یک مثال از عبارات با قاعده:

```
type PhoneRegex = Regex< @"(?<AreaCode>^\d{3})-(?<PhoneNumber>\d{3}-\d{4}$)">

PhoneRegex.IsMatch "425-123-2345"
|> should equal true

PhoneRegex().Match("425-123-2345").CompleteMatch.Value
|> should equal "425-123-2345"

PhoneRegex().Match("425-123-2345").PhoneNumber.Value
|> should equal "123-2345"
```

شروع پروژه

ایتدا یک پروژه از نوع F# Console Application ایجاد کنید. از قسمت Project Properties (بر روی پروژه کلیک راست کنید و گزینه Properties را انتخاب کنید) نوع پروژه را به Windows Application تغییر دهید(قسمت Out Put Type). اسمبلی‌های زیر را به پروژه ارجاع دهید:

PresentationCore

PresentationFramework

WindowBase

System.Xaml

با استفاده از پنجره Package Manager Console دستور نصب زیر را اجرا کنید(آخرین نسخه این پکیج 1.8.31 و حجم آن کمتر از یک مگابایت است):

```
PM> Install-Package FSharpx.TypeProviders.Xaml
```

حال یک فایل Xaml به پروژه اضافه کنید و کدهای زیر را در آن کپی کنید:

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="WPF F# Sample By Masoud Pakdel" Height="350" Width="525">
    <Grid Name="MainGrid">
        <StackPanel Name="StackPanel1" Margin="50">
            <Button Name="Button1">Who are you?</Button>
        </StackPanel>
    </Grid>
</Window>
```

کدهای بالا کاملا واضح است و نیاز به توضیح دیده نمی‌شود. اما اگر دقت کنید می‌بینید که این فایل، فایل Code Behind ندارد. برای این کار باید یک فایل جدید از نوع F# Source File ایجاد کنید. بهتر است که فایل جدید شما همنام با همین فایل باشد. پسوند این فایل fs است. حال کدهای زیر را در آن کپی کنید:

```
open System
open System.Windows
open System.Windows.Controls
open FSharpx

type MainWindow = XAML<"MainWindow.xaml">

let loadWindow() =
    let window = MainWindow()
    window.Button1.Click.Add(fun _ ->
        MessageBox.Show("Masoud Pakdel")
        |> ignore)
    window.Root

[<STAThread>]
(new Application()).Run(loadWindow())
|> ignore
```

نوع XAML استفاده شده که به صورت generic است در فضای نام FSharpx تعبیه شده است و این اجازه را می‌دهد که یک فایل F# بتواند برای مدیریت یک فایل Xaml استفاده شود.برای مثال می‌توانید به اشیا و خواص موجود در فایل Xaml دسترسی داشته باشید. در اینجا دیگر خبری از متد InitializeComponent موجود در سازنده کلاس CodeBehind پروژه‌های C# نیست. این تعاریف و آماده سازی کامپوننت‌ها به صورت توکار در نوع XAML موجود در FSharpx انجام می‌شود.

در تابع loadWindow یک نمونه از کلاس MainWindow ساخته می‌شود و برای button1 آن رویداد کلیک تعریف می‌کنیم. دستورات زیر معادل دستورات شروع برنامه در فایل program پروژه‌های C# است.

```
[<STAThread>]
(new Application()).Run(loadWindow())
|> ignore
```

پروژه را اجرا کنید و بر روی تنهای Button موجود در صفحه، کلیک کنید و پیغام مورد نظر را مشاهده خواهید کرد. به صورت زیر:

عنوان: Markup Extensions در XAML
نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۰۵/۰۷ ۲۰:۵
آدرس: www.dotnettips.info
برچسب‌ها: WPF, XAML

Markup Extensionها برای مواردی استفاده می‌شوند که قرار است مقداری غیر از یک مقدار ثابت و یک نوع قابل شناسایی در XAML برای یک value تنظیم شود. تمام مواردی در XAML که درون {} قرا می‌گیرند همان Markup Extensionها هستند. مانند Binding و یا StaticResoiurces.
علاوه بر Markup Extensionهای از پیش تعریف شده در XAML، می‌توان Markup Extensionهای شخصی را نیز تولید کرد. در واقع به زبان ساده‌تر Markup Extension برای تولید ساده‌ی داده‌های پیچیده در XAML استفاده می‌شوند.

لازم به ذکر است کهMarkup Extension ها می‌توانند به دو صورت Attribute Usage، درون {} :

```
"{Binding path=something,Mode=TwoWay}”
```

و یا Property Element Usage (همانند سایر Element هایWPF) درون <> استفاده شوند:

```
<Binding Path="Something" Mode="TwoWay"></Binding>
```

برای تعریف یک Markup Extension، یک کلاس ایجاد می‌کنیم که از Markup Extensions ارث بری می‌کند. این کلاس یک Abstract Method به نام ProvideValue دارد که باید پیاده سازی شود. این متد مقدار خصوصیتی که Markup Extensions را فراخوانی کرده به صورت یک Object بر می‌گرداند که یکبار در زمان Load برای خصوصیت مربوطه‌اش تنظیم می‌شود.

```
public abstract Object ProvideValue(IServiceProvider serviceProvider)
```

همانطورکه ملاحظه می‌کنید ProvideValue یک پارامتر IServiceProvider دارد که ازطریق آن می‌توان به IProvideValueTarget دسترسی داشت. ازاین Interface برای گرفتن اطلاعات کنترل(TargetObject) و خصوصیتی (TargetProperty) که فراخوانی را انجام داده در صورت لزوم استفاده می‌شود.

```
var target = serviceProviderGetService(typeof(IProvideValueTarget))as IProvideValueTarget;
var host = targetTargetObject as FrameworkElement;
```

Markup Extensionها می‌توانند پارامترهای ورودی داشته باشند:

```
public class ValueExtension : MarkupExtension
{
  public ValueExtension () { }
  public ValueExtension (object value1)
  {
    Value1 = value1;
  }
   public object Value1 { get; set; }
   public override object ProvideValue(IServiceProvider serviceProvider)
   {
     return Value1;
   }
}
```

و برای استفاده در فایل Xaml:

```
<TextBox  Text="{app:ValueExtension test}" ></TextBox>
```

و یا می‌توان خصوصیت هایی ایجاد کرد و از آنها برای ارسال مقادیر به آن استفاده کرد:

```
<TextBox  Text="{app:ValueExtension Value1=test}" ></TextBox>
```

تا اینجا موارد کلی برای تعریف و استفاده از Markup Extensions گفته شد. در ادامه یک مثال کاربردی می‌آوریم. برای مثال در نظر بگیرید که نیاز دارید DataType مربوط به یک DataTemplate را برابر یک کلاس Generic قرار بدهید:

```
public class EntityBase
{
   public int Id{get;set}
}

public class MyGenericClass<TType> where TType : EntityBase
{
   public int Id{get;set}
   public string Test{  get;set; }

In XAML:

<DataTemplate DataType="{app:GenericType ؟}">
```

برای انجام این کار یک Markup Extensions به صورت زیر ایجاد می‌کنیم که Type را به عنوان ورودی گرفته و یک نمونه از کلاس Generic ایجاد می‌کند:

```
public class GenericType : MarkupExtension
{
  private readonly Type _of;
  public GenericType(Type of)
  {
     _of = of;
  }
  public override object ProvideValue(IServiceProvider serviceProvider)
  {
      return typeof(MyGenericClass<>)MakeGenericType(_of);
 }
}
```

و برای استفاده از آن یک نمونه از MarkupExtension ایجاد شده ایجاد کرده و نوع Generic را برای آن ارسال می‌کنیم:

```
<DataTemplate DataType="{app:GenericType app:EntityBase}">
```

این یک مثال ساده از استفاده از Markup Extensions است. هنگام کار با WPF می‌توان استفاده‌های زیادی از این مفهوم داشت، برای مثال زمانی که نیاز است ItemsSource یک Combobox از Descriptionهای یک Enum پر شود می‌توان به راحتی با نوشتن یک Markup Extensions ساده این عمل و کارهای مشابه زیادی را انجام داد.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۰۸ ۲۳:۵۰

یک مثال جالب در این مورد

DelayBinding: a custom WPF Binding

عنوان: Debug کردن Binding در XAML
نویسنده: بزمی
تاریخ: ۱۳۹۲/۰۵/۰۹ ۱۵:۰
آدرس: www.dotnettips.info
برچسب‌ها: Debugging, Silverlight, WPF, XAML, Binding

اغلب در حین Bind کردن Propertyها در XAML به مشکل Bind نشدن بر می‌خوریم. من معمولا از روش زیر استفاده می‌کنم:

```
public class DatabindingDebugConverter : IValueConverter
    {
        #region IValueConverter Members

        public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
        {
            Debugger.Break();
            return value;
        }

        public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
        {
            Debugger.Break();
            return value;
        }

        #endregion IValueConverter Members
    }
```

و در XAML :

```
<DataTemplate.Resources>
            <debug:DatabindingDebugConverter x:Key="databindingDebugConverter"/>
</DataTemplate.Resources>
<DataGrid ItemsSource="{Binding myViewModel,Converter={StaticResource databindingDebugConverter}}" />
```

و حال دو حالت می‌تواند اتفاق بیفتد :

1 - Break Point Hit نمی‌شود:

در این حالت مقدار myViewModel خالی (null) است و یا اصلا myViewModel در DataContext مربوط به DataGrid وجود ندارد در این صورت همچنین در پنجره Out Put Visual Studio:

```
System.Windows.Data Error: 35 : BindingExpression path error: 'X' property not found ...
```

و با search متن "System.Windows.Data Error: 35 : BindingExpression path error:" در Out Put میتوان متوجه آن شد.

2 - Break Point Hit می‌شود:

در این حالت باید value را Watch کنیم (Shift+F9) تا ببینیم علت Bind نشدن چیست؟ شاید (در این مورد خاص) نوع myViewModel از IEnumerable نباشد ...

در حین بررسی و Debug ، شاید گاهی مسئله لاینحل به نظر برسد ، ولی به نظر من معمولا با کم و زیاد کردن آدرس (Binding Path) به یکی از دو حالت بالا خواهیم رسید ،

مثلا زمانی که Path به صورت myViewModel.MyProperty.MyInnerPtoperty است ، باید Path را با حالات زیر توسط Converter مذکور تست کنیم:

```
Binding"{Path=myViewModel.MyProperty.MyInnerPtoperty ,Converter="{StaticResource debugger}}"
Binding"{Path=myViewModel.MyProperty,Converter="{StaticResource debugger}}"
Binding"{Path=myViewModel,Converter="{StaticResource debugger}}"
Binding"{Path=.,Converter="{StaticResource debugger}}"
```

امیدوارم از Binding تان لذت ببرید.

عنوان: Bind کردن Enum به ItemsSource در XAML
نویسنده: بزمی
تاریخ: ۱۳۹۲/۰۵/۱۱ ۸:۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, Silverlight, WPF, XAML, Binding

شاید تا به حال در یک برنامه سازمانی نیاز به Bind کردن یک Enum به کنترل‌های XAML به چشمتان خورده باشد ، روشی که من برای این کار استفاده می‌کنم توسط یک Markup Extension به صورت زیر است :

```
public class ByteEnumerationExtention : MarkupExtension
    {
        public ByteEnumerationExtention(Type enumType)
        {
            this.enumType = enumType;
        }

        private Type enumType;

        public Type EnumType
        {
            get { return enumType; }
            private set
            {
                enumType = value;
            }
        }

        public override object ProvideValue(IServiceProvider serviceProvider)
        {
            return (from x in Enum.GetValues(EnumType).OfType<Enum>()
                    select new EnumerationMember
                    {
                        Value = GetValue(x),
                        Description = GetDescription(x)
                    }).ToArray();
        }

        private byte? GetValue(object enumValue)
        {
            return Convert.ToByte(enumValue.GetType().GetField("value__").GetValue(enumValue));
        }

        public object GetObjectValue(object enumValue)
        {
            return enumValue.GetType().GetField("value__").GetValue(enumValue);
        }

        public string GetDescription(object enumValue)
        {
            var descAttrib = EnumType.GetField(enumValue.ToString())
                .GetCustomAttributes(typeof(DescriptionAttribute), false)
                .FirstOrDefault() as DescriptionAttribute;
            return descAttrib != null ? descAttrib.Description : enumValue.ToString();
        }
    }

    public class EnumerationMember
    {
        public string Description { get; set; }

        public byte? Value { get; set; }
    }
```

: XAML

```
<ComboBox ItemsSource="{Binding Source={ Extensions:ByteEnumerationExtention {x:Type type:MyEnum} }}"
DisplayMemberPath="Description"
 SelectedValuePath="Value" SelectedValue="{Binding SelectedItemInViewModel}"/>
```

: Enum

```
public enum MyEnum : short
    {
        [Description("گزینه 1")]
        Item1 = 1,

        [Description("گزینه 2")]
        Item1 = 2,

        [Description("گزینه 3")]
        Item1 = 3,

        [Description("گزینه 4")]
        Item1 = 4,

        [Description("گزینه 5")]
        Item1 = 5,
.
.
.
    }
```

SelectedItem در ViewModel :

```
short? selectedItemInViewModel;
        public short? SelectedItemInViewModel
        {
            get
            {
                return selectedItemInViewModel;
            }
            set
            {
                selectedItemInViewModel = value;
                 RaisePropertyChanged("SelectedItemInViewModel");
               //do calculations if needed
            }
        }
```

عنوان: **دسترسی به فیلد های Static در XAML**
نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۰۵/۱۵ ۲۰:۵
آدرس: www.dotnettips.info
برچسب‌ها: WPF, XAML

MarkupExtension ها قبلا در اینجا توضیح داده شده اند. یکی از MarkupExtensionهای از پیش تعریف شده x:Static است که برای مقداردهی یک خصوصیت در XAML با یک مقدار استاتیک استفاده می‌شود.اگر بخواهید از یک ثابت(constant)،یک خصوصیت استاتیک(static property)، یا یک مقدار از یک enumeration، برای مقداردهی یک خصوصیت در XAML استفاده کنید باید از این MarkupExtension استفاده کنید.

برای مثال برای یک استفاده از یک خصوصیت استایک به صورت زیر عمل می‌کنیم:

```
namespace Test
 {
   public class Constants
   {
      public static readonly string ConstantString = "Test";
   }
 }
```

توجه داشته باشید که برای استفاده از این ثابت باید ابتدا فضای نام مربوط به آن را تعریف کنید.

```
xmlns:test="clr-namespace:ItemTest "
<Label Content="{x:Static test:Constants.ConstantString}" />
```

و یا برای مقدار دهی از طریق یک Enumeration

```
namespace Test
{
  public enum VisiblityEnum
     {
         Collapse,
         Hidden,
         Visible
     };
}
```

و در فایل XAML:

```
xmlns:test="clr-namespace:Test"
<Label Content="{x:Static test:VisiblityEnum.Collapse}" />
```

برای استفاده از یک ثابت نیز به همین صورت عمل می‌کنیم.

عنوان: **نگاهی به درون سیستم Binding در WPF و یافتن مواردی که هنوز در حافظه‌اند**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۰ ۱۴:۵
آدرس: www.dotnettips.info
گروه‌ها: WPF, Performance, Memory Manegement, GC

در WPF، زیر ساخت‌های ComponentModel توسط کلاسی به نام PropertyDescriptor ، منابع Binding موجود در قسمت‌های مختلف برنامه را در جدولی عمومی ذخیره و نگهداری می‌کند. هدف از آن، مطلع بودن از مواردی است که نیاز دارند توسط مکانیزم‌هایی مانند INotifyPropertyChanged و DependencyProperty ها، اطلاعات اشیاء متصل را به روز کنند.
در این سیستم، کلیه اتصالاتی که Mode آن‌ها به OneTime تنظیم نشده است، به صورت اجباری دارای یک valueChangedHandlers متصل توسط سیستم PropertyDescriptor خواهند بود و در حافظه زنده نگه داشته می‌شوند؛ تا بتوان در صورت نیاز، توسط سیستم binding اطلاعات آن‌ها را به روز کرد.
همین مساله سبب می‌شود تا اگر قرار نیست خاصیتی برای نمونه توسط مکانیزم INotifyPropertyChanged اطلاعات UI را به روز کند (یک خاصیت معمولی دات نتی است) و همچنین حالت اتصال آن به OneTime نیز تنظیم نشده، سبب مصرف حافظه بیش از حد برنامه شود.
اطلاعات بیشتر
A memory leak may occur when you use data binding in Windows Presentation Foundation

راه حل آن هم ساده است. برای اینکه valueChangedHandler ایی به خاصیت ساده‌ای که قرار نیست بعدها UI را به روز کند، متصل نشود، حالت اتصال آن‌را باید به OneTime تنظیم کرد.

سؤال: در یک برنامه بزرگ که هم اکنون مشغول به کار است، چطور می‌توان این مسایل را ردیابی کرد؟

برای دستیابی به اطلاعات کش Binding در WPF، باید به Reflection متوسل شد. به این ترتیب در برنامه جاری، در کلاس PropertyDescriptor به دنبال یک کلاس خصوصی تو در توی دیگری به نام ReflectTypeDescriptionProvider خواهیم گشت (این اطلاعات از طریق مراجعه به سورس دات نت و یا حتی برنامه‌های ILSpy و Reflector قابل استخراج است) و سپس در این کلاس خصوصی داخلی، فیلد خصوصی propertyCache آن‌را که از نوع HashTable است استخراج می‌کنیم:

```
var reflectTypeDescriptionProvider =
typeof(PropertyDescriptor).Module.GetType("System.ComponentModel.ReflectTypeDescriptionProvider");
var propertyCacheField = reflectTypeDescriptionProvider.GetField("_propertyCache",
BindingFlags.Static | BindingFlags.NonPublic);
```

```
ReflectTypeDescriptionProvider

[HostProtection(SecurityAction.LinkDemand, SharedState=true)]
internal sealed class ReflectTypeDescriptionProvider : TypeDescriptionProvider
{
    // Fields
    private static Hashtable _attributeCache;
    private static object _dictionaryKey;
    private static Hashtable _editorTables;
    private static Hashtable _eventCache;
    private static Hashtable _extendedPropertyCache;
    private static readonly Guid _extenderPropertiesKey;
    private static readonly Guid _extenderProviderKey;
    private static readonly Guid _extenderProviderPropertiesKey;
    private static object _internalSyncObject;
    private static object _intrinsicNullableKey;
    private static object _intrinsicReferenceKey;
    private static Hashtable _intrinsicTypeConverters;
    private static Hashtable _propertyCache;
    private static readonly Type[] _skipInterfaceAttributeList;
    private static Type[] _typeConstructor;
    private Hashtable _typeData;
```

اکنون به لیست داخلی Binding نگهداری شونده توسط WPF دسترسی پیدا کرده‌ایم. در این لیست به دنبال مواردی خواهیم گشت که فیلد valueChangedHandlers به آن‌ها متصل شده است و در حال گوش فرا دادن به سیستم binding هستند (سورس کامل و طولانی این مبحث را در پروژه پیوست شده می‌توانید ملاحظه کنید).

یک مثال: تعریف یک کلاس ساده، اتصال آن و سپس بررسی اطلاعات درونی سیستم Binding

فرض کنید یک کلاس مدل ساده به نحو ذیل تعریف شده است:

```
namespace WpfOneTime.Models
{
    public class User
    {
        public string Name { set; get; }
    }
}
```

سپس این کلاس به صورت یک List، توسط ViewModel برنامه در اختیار View متناظر با آن قرار می‌گیرد:

```
using WpfOneTime.Models;
using System.Collections.Generic;

namespace WpfOneTime.ViewModels
{
    public class MainWindowViewModel
    {
        public IList<User> Users { set; get; }

        public MainWindowViewModel()
        {
            Users = new List<User>();
```

```
            for (int i = 0; i < 1000; i++)
            {
                Users.Add(new User { Name = "name " + i });
            }
        }
    }
}
```

تعاریف View برنامه نیز به نحو زیر است:

```
<Window x:Class="WpfOneTime.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:ViewModels="clr-namespace:WpfOneTime.ViewModels"
        Title="MainWindow" Height="350" Width="525">
    <Window.Resources>
        <ViewModels:MainWindowViewModel x:Key="vmMainWindowViewModel" />
    </Window.Resources>
    <Grid DataContext="{Binding Source={StaticResource vmMainWindowViewModel}}">
        <ListBox ItemsSource="{Binding Users}">
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding Name}" />
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
    </Grid>
</Window>
```

همه چیز در آن معمولی به نظر می‌رسد. ابتدا به ViewModel برنامه دسترسی یافته و DataContext را با آن مقدار دهی می‌کنیم. سپس اطلاعات این لیست را توسط یک ListBox نمایش خواهیم داد.

خوب؛ اکنون اگر اطلاعات HashTable داخلی سیستم Binding را در مورد View فوق بررسی کنیم به شکل زیر خواهیم رسید:

بله. تعداد زیادی خاصیت Name زنده و موجود در حافظه باقی هستند که تحت ردیابی سیستم Binding می‌باشند.
در ادامه، نکته‌ی ابتدای بحث را جهت تعیین حالت Binding به OneTime ، به View فوق اعمال می‌کنیم (یک سطر ذیل باید تغییر کند):

```
<TextBlock Text="{Binding Name, Mode=OneTime}" />
```

در این حالت اگر نگاهی به سیستم ردیابی WPF داشته باشیم، دیگر خبری از اشیاء زنده دارای خاصیت Name در حال ردیابی نیست:

به این ترتیب می‌توان در لیست‌های طولانی، به مصرف حافظه کمتری در برنامه WPF خود رسید.
بدیهی است این نکته را تنها در مواردی می‌توان اعمال کرد که نیاز به به‌روز رسانی‌های ثانویه اطلاعات UI در کدهای برنامه وجود ندارند.

چطور از این نکته برای پروفایل یک برنامه موجود استفاده کنیم؟

کدهای برنامه را از انتهای بحث دریافت کنید. سپس دو فایل ReflectPropertyDescriptorWindow.xaml و ReflectPropertyDescriptorWindow.xaml.cs آن‌را به پروژه خود اضافه نمائید و در سازنده پنجره اصلی برنامه، کد ذیل را فراخوانی نمائید:

```
new ReflectPropertyDescriptorWindow().Show();
```

کمی با برنامه کار کرده و منتظر شوید تا لیست نهایی اطلاعات داخلی Binding ظاهر شود. سپس مواردی را که دارای HandlerCount بالا هستند، مدنظر قرار داده و بررسی نمائید که آیا واقعا این اشیاء نیاز به valueChangedHandler متصل دارند یا خیر؟ آیا قرار است بعدها UI را از طریق تغییر مقدار خاصیت آن‌ها به روز نمائیم یا خیر. اگر خیر، تنها کافی است نکته Mode=OneTime را به این Bindingها اعمال نمائیم.

دریافت کدهای کامل پروژه این مطلب

WpfOneTime.zip

نظرات خوانندگان

نویسنده: سیما
تاریخ: ۱۳۹۳/۰۳/۲۳ ۱۹:۲

سلام،
می‌خواستم بدونم به چه شکل میتوانم متوجه شوم کدام قسمت از برنامه من موجب افزایش مصرف رم شده است؟ برای مثال برنامه من بعد گذشت 1 دقیقه از اجرای آن مصرف رم معادل 5MB دارم ولی پس از گذشت 10 دقیقه به 1GB میرسد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۲۳ ۱۹:۱۷

از برنامه‌های Profiler باید استفاده کنید؛ مانند:

- ابزارهای توکار VS.NET
- New Memory Usage Tool for WPF and Win32 Applications
- Windows Performance Toolkit
- dotMemory
- ANTS Memory Profiler

عنوان: **بهبود کارآیی کنترل‌های لیستی WPF در حین بارگذاری تعداد زیادی از رکوردها**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۱ ۸:۰
آدرس: www.dotnettips.info
گروه‌ها: WPF, Performance

کنترل‌های WPF در حالت پیش فرض و بدون اعمال قالب خاصی به آن‌ها عموما خوب عمل می‌کنند. مشکل از جایی شروع می‌شود که قصد داشته باشیم حالت پیش فرض را اندکی تغییر دهیم و یا Visual tree این کنترل‌ها اندکی پیچیده شوند. برای نمونه مدل زیر را در نظر بگیرید:

```
using System;

namespace WpfLargeLists.Models
{
    public class User
    {
        public int Id { set; get; }
        public string FirstName { set; get; }
        public string LastName { set; get; }
        public string Address { set; get; }
        public DateTime DateOfBirth { set; get; }
    }
}
```

قصد داریم فقط 1000 رکورد ساده از این مدل را به یک ListView اعمال کنیم.

```
<ListView ItemsSource="{Binding UsersTab1}" Grid.Row="1" Margin="3">
                        <ListView.View>
                            <GridView>
                                <GridViewColumn Header="Id" Width="50" DisplayMemberBinding="{Binding
Id, Mode=OneTime}" />
                                <GridViewColumn Header="FirstName" Width="100"
DisplayMemberBinding="{Binding FirstName, Mode=OneTime}" />
                                <GridViewColumn Header="LastName" Width="100"
DisplayMemberBinding="{Binding LastName, Mode=OneTime}" />
                                <GridViewColumn Header="Address" Width="100"
DisplayMemberBinding="{Binding Address, Mode=OneTime}" />
                                <GridViewColumn Header="DateOfBirth" Width="150"
DisplayMemberBinding="{Binding DateOfBirth, Mode=OneTime}" />
                            </GridView>
                        </ListView.View>
                    </ListView>
```

در اینجا UsersTab1، لیستی حاوی فقط 1000 رکورد از شیء User است. در حالت معمولی این لیست بدون مشکل بارگذاری می‌شود. اما با اعمال مثلا قالب MahApp.Metro ، بارگذاری همین لیست، حدود 5 ثانیه با CPU usage صد در صد طول می‌کشد. علت اینجا است که در این حالت WPF سعی می‌کند تا ابتدا در VisualTree، کل 1000 ردیف را کاملا ایجاد کرده و سپس نمایش دهد.

راه حل توصیه شده برای بارگذاری تعداد بالایی رکورد در WPF : استفاده از UI Virtualization

UI Virtualization روشی است که در آن تنها المان‌هایی که توسط کاربر در حال مشاهده هستند، تولید و مدیریت خواهند شد. در این حالت اگر 1000 رکورد را به یک ListBox یا ListView ارسال کنید و کاربر بر اساس اندازه صفحه جاری خود تنها 10 رکورد را مشاهده می‌کند، WPF فقط 10 عنصر را در VisualTree مدیریت خواهد کرد. با اسکرول به سمت پایین، مواردی که دیگر نمایان نیستند dispose شده و مجموعه نمایان دیگری خلق خواهند شد. به این ترتیب می‌توان حجم بالایی از اطلاعات را در WPF با میزان مصرف پایین حافظه و همچنین مصرف CPU بسیار کم مدیریت کرد. این مجازی سازی در WPF به وسیله VirtualizingStackPanel در دسترس است.

برای اینکه WPF virtualization به درستی کار کند، نیاز است یک سری شرایط مقدماتی فراهم شوند:
- از کنترلی استفاده شود که از virtualization پشتیبانی می‌کند ؛ مانند ListBox و ListView.
- ارتفاع کنترل لیستی باید دقیقا مشخص باشد؛ یا درون یک ردیف از Grid ایی باشد که ارتفاع آن مشخص است. برای نمونه اگر ارتفاع ردیف Grid ایی که ListView را دربرگرفته است به * تنظیم شده، مشکلی نیست؛ اما اگر ارتفاع این ردیف به Auto تنظیم

شده، کنترل لیستی برای محاسبه vertical scroll bar خود دچار مشکل خواهد شد.
- کنترل مورد استفاده نباید در یک کنترل ScrollViewer محصور شود؛ در غیر اینصورت virtualization رخ نخواهد داد. علاوه بر آن در خود کنترل باید خاصیت ScrollViewer.HorizontalScrollBarVisibility نیز به Disabled تنظیم گردد.
- در کنترل در حال استفاده، ScrollViewer.CanContentScroll باید به true تنظیم شود.

مورد مشخص بودن ارتفاع بسیار مهم است. برای نمونه در برنامه‌ای پس از فعال سازی مجازی سازی، کنترل لیستی کلا از کار افتاد و حرکت scroll bar آن سبب بروز CPU Usage بالایی می‌شد. این مشکل با تنظیم ارتفاع آن به شکل زیر برطرف شد:

```
Height="{Binding Path=RowDefinitions[1].ActualHeight, RelativeSource={RelativeSource
AncestorType=Grid}}"
```

در این تنظیم، ارتفاع کنترل، به ارتفاع سطر دوم گرید دربرگیرنده ListView متصل شده است.

- پس از اعمال موارد یاد شده، باید VirtualizingStackPanel کنترل را فعال کرد. ابتدا دو خاصیت زیر باید مقدار دهی شوند:

```
VirtualizingStackPanel.IsVirtualizing="True"
  VirtualizingStackPanel.VirtualizationMode="Recycling"
```

سپس ItemsPanelTemplate کنترل باید به صورت یک VirtualizingStackPanel مقدار دهی شود. برای مثال اگر از ListBox استفاده می‌کنید، تنظیمات آن به نحو زیر است:

```
<ListBox.ItemsPanel>
    <ItemsPanelTemplate>
          <VirtualizingStackPanel IsVirtualizing="True" VirtualizationMode="Recycling" />
    </ItemsPanelTemplate>
</ListBox.ItemsPanel>
```

و اگر از ListView استفاده می‌شود، تنظیمات آن مشابه ListBox است:

```
<ListView.ItemsPanel>
    <ItemsPanelTemplate>
          <VirtualizingStackPanel
               IsVirtualizing="True"
               VirtualizationMode="Recycling" />
    </ItemsPanelTemplate>
</ListView.ItemsPanel>
```

با این توضیحات ListView ابتدای بحث به شکل زیر تغییر خواهد یافت تا مجازی سازی آن فعال گردد:

```
<ListView ItemsSource="{Binding UsersTab2}" Grid.Row="1" Margin="3"
                             ScrollViewer.HorizontalScrollBarVisibility="Disabled"
                             ScrollViewer.CanContentScroll="True"
                             VirtualizingStackPanel.IsVirtualizing="True"
                             VirtualizingStackPanel.VirtualizationMode="Recycling">
                         <ListView.ItemsPanel>
                             <ItemsPanelTemplate>
                                 <VirtualizingStackPanel
                                         IsVirtualizing="True"
                                         VirtualizationMode="Recycling" />
                             </ItemsPanelTemplate>
                         </ListView.ItemsPanel>
                         <ListView.View>
                             <GridView>
                                 <GridViewColumn Header="Id" Width="50" DisplayMemberBinding="{Binding
Id, Mode=OneTime}" />
                                 <GridViewColumn Header="FirstName" Width="100"
DisplayMemberBinding="{Binding FirstName, Mode=OneTime}" />
                                 <GridViewColumn Header="LastName" Width="100"
DisplayMemberBinding="{Binding LastName, Mode=OneTime}" />
                                 <GridViewColumn Header="Address" Width="100"
DisplayMemberBinding="{Binding Address, Mode=OneTime}" />
                                 <GridViewColumn Header="DateOfBirth" Width="150"
```

```
DisplayMemberBinding="{Binding DateOfBirth, Mode=OneTime}" />
                                </GridView>
                            </ListView.View>
                        </ListView>
```

کدهای کامل مثال فوق را از اینجا می‌توانید دریافت کنید: WpfLargeLists.zip

در این مثال دو برگه را ملاحظه می‌کنید. برگه اول حالت normal ابتدای بحث است و برگه دوم پیاده سازی UI Virtualization را انجام داده است.

نظرات خوانندگان

نویسنده: سیما
تاریخ: ۱۳۹۳/۰۳/۲۳ ۱۸:۵۳

با سلام،
میخواستم بدونم امکانش هست از WrapPanel هم بعنوان یک ItemsPanelTemplate با رعایت Virtualization استفاده کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۲۳ ۱۹:۲۲

Virtualizing WrapPanel : ^ و ^

عنوان: **چگونه تشخیص دهیم UI Virtualization در WPF خاموش شده است؟**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۲ ۱۲:۰
آدرس: www.dotnettips.info
برچسب‌ها: WPF, Performance

در مطلب « بهبود کارآیی کنترل‌های لیستی WPF در حین بارگذاری تعداد زیادی از رکوردها » عنوان شد که در حالت فعال بودن UI Virtualization، فقط به تعداد ردیف‌های نمایان، اشیاء متناظری به یک کنترل لیستی اضافه می‌شوند و حالت برعکس آن زمانی است که ابتدا کلیه اشیاء بصری یک لیست تولید شده و سپس لیست نهایی نمایش داده می‌شود.

سؤال: چگونه می‌توان تعداد اشیاء اضافه شده به Visual tree یک کنترل لیستی را شمارش کرد؟

شبیه به افزونه FireBug فایرفاکس، برنامه‌ای به نام Snoop نیز جهت WPF تهیه شده است که با تزریق خود به درون پروسه برنامه، امکان بررسی ساختار Visual tree کل یک صفحه را فراهم می‌کند. برای دریافت آن به آدرس ذیل مراجعه نمائید:

http://snoopwpf.codeplex.com

پس از دریافت، ابتدا مثال انتهای بحث « بهبود کارآیی کنترل‌های لیستی WPF در حین بارگذاری تعداد زیادی از رکوردها » را اجرا کرده و سپس برنامه Snoop را نیز جداگانه اجرا نمائید. اگر نام برنامه WPF مورد نظر، در لیست برنامه‌های تشخیص داده شده توسط Snoop ظاهر نشد، یکبار بر روی دکمه Refresh آن کلیک نمائید. پس از آن برنامه نمایش لیست‌ها را در Snoop انتخاب کرده و دکمه کنار آیکن minimize کردن Snoop را کشیده و بر روی پنجره برنامه رها کنید. شکل زیر ظاهر خواهد شد:

بله. همانطور که ملاحظه می‌کنید، در برگه Slow version به علت فعال نبودن مجازی سازی UI، تعداد اشیاء موجود در Visual tree کنترل لیستی، بالای 10 هزار مورد است. به همین جهت بارگذاری آن بسیار کند انجام می‌شود.

اکنون همین عملیات کشیدن و رها کردن دکمه بررسی Snoop را بر روی برگه دوم برنامه انجام دهید:

در اینجا چون مجازی سازی UI فعال شده است، فقط به تعداد ردیف‌های نمایان به کاربر، اشیاء لازم جهت نمایش لیست، تولید و اضافه شده‌اند که در اینجا فقط 188 مورد است و در مقایسه با 10 هزار مورد برگه اول، بسیار کمتر می‌باشد و بدیهی است در این حالت مصرف حافظه برنامه بسیار کمتر بوده و همچنین سرعت نمایش لیست نیز بسیار بالا خواهد بود.

عنوان: **اضافه کردن امکان ویرایش WPF DataGrid در صورت نامعتبر بودن سلول ها**
نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۰۷/۱۷ ۹:۱۵
آدرس: www.dotnettips.info
برچسب‌ها: WPF

اگر با دیتا گریدهای WPF کار کرده باشید، به این مساله برخورد کرده اید که وقتی روی یک سلول از دیتا گرید Validaion اعمال شده باشد و آن سلول مقدار نامعتبر داشته باشد،امکان ویرایش سایر ردیف‌ها و سلول‌ها وجود ندارد.در بعضی مواقع نیاز است که این رفتار دیتا گرید غیر فعال شود.یکی از راه هایی که می‌توان این کار را انجام داد Override کردن متد OnCanExecuteBeginEdit مربوط به دیتا گرید و تغییر مقدار CanExecute در صورت Invalid بودن سلول‌ها است.

```
protected override void OnCanExecuteBeginEdit(System.Windows.Input.CanExecuteRoutedEventArgs e)
        {
            var hasCellValidationError = false;
            var hasRowValidationError = false;
            const BindingFlags bindingFlags =
                BindingFlags.FlattenHierarchy | BindingFlags.NonPublic | BindingFlags.Instance;
            //Current cell
            var cellErrorInfo = this.GetType().BaseType.GetProperty("HasCellValidationError",
bindingFlags);
            //Grid row
            var rowErrorInfo = this.GetType().BaseType.GetProperty("HasRowValidationError",
bindingFlags);
            if (cellErrorInfo != null) hasCellValidationError = (bool) cellErrorInfo.GetValue(this,
null);
            if (rowErrorInfo != null) hasRowValidationError = (bool) rowErrorInfo.GetValue(this, null);
            base.OnCanExecuteBeginEdit(e);
            if ((!e.CanExecute && hasCellValidationError) || (!e.CanExecute && hasRowValidationError))
            {
                e.CanExecute = true;
                e.Handled = true;
            }
     }
```

درحالت پیش فرض هنگامی که CellValidationError یا RowValidationError وجود داشته باشد CanExecute برابر False است و امکان ویرایش دیتا گرید وجود ندارد که در تکه کد بالا این رفتار تغییر داده شده است.

عنوان: **انقیاد RadioButtonها در WPF به یک Enum**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۲۷ ۱۲:۰
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF

فرض کنید قصد دارید برای انتخاب بین چند گزینه‌ی محدود، از RadioButtonها بجای سایر کنترل‌های موجود استفاده کنید. این گزینه‌ها نیز توسط یک Enum تعریف شده‌اند. اکنون نیاز است گزینه‌های مختلف این Enum را به RadioButtonهای تعریف شده Bind کنیم.

تعریف Enum برنامه به صورت زیر است:

```
namespace WpfBindRadioButtonToEnum.Models
{
    public enum Gender
    {
        Female,
        Male
    }
}
```

در ادامه با توجه به اینکه RadioButtonها با خاصیت IsChecked از نوع bool کار می‌کنند، نیاز است بتوانیم گزینه‌های Enum را به bool و یا برعکس تبدیل کنیم . برای این منظور از تبدیلگر EnumBooleanConverter ذیل می‌توان استفاده کرد:

```
using System;
using System.Globalization;
using System.Windows;
using System.Windows.Data;

namespace WpfBindRadioButtonToEnum.Converters
{
    public class EnumBooleanConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            if (Enum.IsDefined(value.GetType(), value) == false)
                return DependencyProperty.UnsetValue;

            return Enum.Parse(value.GetType(), parameter.ToString()).Equals(value);
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            return Enum.Parse(targetType, parameter.ToString());
        }
    }
}
```

پیش‌فرض تبدیلگر تهیه شده بر این است که مقدار ثابت Enum را از طریق سومین پارامتر، یعنی ConverterParameter تنظیم شده در حین عملیات Binding، دریافت می‌کند. پارامتر value مقداری است که از طریق Binding خاصیت IsChecked دریافت خواهد شد.

اکنون اگر ViewModel برنامه به شکل زیر باشد که GenderValue را در اختیار View قرار می‌دهد:

```
using System.ComponentModel;
using WpfBindRadioButtonToEnum.Models;

namespace WpfBindRadioButtonToEnum.ViewModels
{
    public class MainWindowViewModel : INotifyPropertyChanged
    {
        Gender _genderValue;
        public Gender GenderValue
        {
            get { return _genderValue; }
            set
            {
                _genderValue = value;
```

```
                notifyPropertyChanged("GenderValue");
            }
        }



        #region INotifyPropertyChanged Members
        public event PropertyChangedEventHandler PropertyChanged;
        private void notifyPropertyChanged(string propertyName)
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
            }
        }
        #endregion
    }
}
```

View متناظری که از آن و همچنین Enum و تبدیلگر تهیه شده استفاده می‌کند، به شرح ذیل خواهد بود:

```
<Window x:Class="WpfBindRadioButtonToEnum.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:VM="clr-namespace:WpfBindRadioButtonToEnum.ViewModels"
        xmlns:C="clr-namespace:WpfBindRadioButtonToEnum.Converters"
        xmlns:Models="clr-namespace:WpfBindRadioButtonToEnum.Models"
        Title="MainWindow" Height="350" Width="525">
    <Window.Resources>
        <VM:MainWindowViewModel x:Key="VMainWindowViewModel" />
        <C:EnumBooleanConverter x:Key="CEnumBooleanConverter" />
    </Window.Resources>
    <StackPanel DataContext="{Binding Source={StaticResource VMainWindowViewModel}}">
        <TextBlock Text="Gender" Margin="3" />
        <RadioButton Content="{x:Static Models:Gender.Male}"
                     IsChecked="{Binding GenderValue, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged,
                                 Converter={StaticResource CEnumBooleanConverter},
                                 ConverterParameter={x:Static Models:Gender.Male}}"
                     Margin="3" GroupName="G1" />
        <RadioButton Content="{x:Static Models:Gender.Female}"
                     IsChecked="{Binding GenderValue, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged,
                                 Converter={StaticResource CEnumBooleanConverter},
                                 ConverterParameter={x:Static Models:Gender.Female}}"
                     Margin="3" GroupName="G1" />
    </StackPanel>
</Window>
```

در این View از یک markup extension به نام x:Static برای دسترسی به فیلدهای ثابت برنامه کمک گرفته شده‌است. از x:Static در ConverterParameter و همچنین Content می‌توان استفاده کرد. برای دسترسی به Enum تعریف شده در برنامه، فضای نام آن توسط xmlns:Models در ابتدای کار تعریف گردیده‌است.

در اینجا EnumBooleanConverter تهیه شده، کار تبدیل مقدار true و false دریافتی از IsChecked را به معادل Enum آن و برعکس، انجام می‌دهد.

به صورت خلاصه: ابتدا تبدیلگر EnumBooleanConverter باید اضافه شود. سپس به ازای هر گزینه‌ی Enum، یک RadioButton در صفحه قرار می‌گیرد که ConverterParameter خاصیت IsChecked آن مساوی است با یکی از گزینه‌های Enum متناظر.

عنوان: **یکی کردن اسمبلی‌های یک پروژه‌ی WPF**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۳۰ ۱۸:۳۵
آدرس: www.dotnettips.info
گروه‌ها: WPF

فرض کنید پروژه‌ی WPF شما از چندین پروژه‌ی Class library و اسمبلی‌های جانبی دیگر، تشکیل شده‌است. اکنون نیاز است جهت سهولت توزیع آن، تمام این فایل‌ها را با هم یکی کرده و تبدیل به یک فایل EXE نهایی کنیم. مایکروسافت ابزاری را به نام ILMerge ، برای یک چنین کارهایی تدارک دیده‌است؛ اما این برنامه با WPF سازگار نیست. در ادامه قصد داریم اسمبلی‌های جانبی را تبدیل به منابع مدفون شده در فایل EXE برنامه کرده و سپس آن‌ها را در اولین بار اجرای برنامه، به صورت خودکار بارگذاری و در برنامه مورد استفاده قرار دهیم.

یک مثال جهت بازتولید کدهای این مطلب

الف) یک پروژه‌ی WPF جدید را به نام MergeAssembliesIntoWPF ایجاد کنید.

ب) یک پروژه‌ی Class library جدید را به نام MergeAssembliesIntoWPF.ViewModels به این Solution اضافه کنید. از آن برای تعریف ViewModelهای برنامه استفاده خواهیم کرد.

برای نمونه کلاس ذیل را به آن اضافه کنید:

```
namespace MergeAssembliesIntoWPF.ViewModels
{
    public class ViewModel1
    {
        public string Data { set; get; }

        public ViewModel1()
        {
            Data = "Test";
        }
    }
}
```

ج) یک پروژه‌ی WPF User control library را نیز به نام MergeAssembliesIntoWPF.Shell به این Solution اضافه کنید. از آن برای تعریف Viewهای برنامه کمک خواهیم گرفت.

به این پروژه ارجاعی را به اسمبلی قسمت (ب) اضافه نموده و برای نمونه User control ذیل را به نام View1.xaml به آن اضافه نمائید:

```
<UserControl x:Class="MergeAssembliesIntoWPF.Shell.View1"
             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"
             xmlns:VM="clr-
namespace:MergeAssembliesIntoWPF.ViewModels;assembly=MergeAssembliesIntoWPF.ViewModels"
             d:DesignHeight="300" d:DesignWidth="300">
    <UserControl.Resources>
        <VM:ViewModel1 x:Key="ViewModel1" />
    </UserControl.Resources>
    <Grid DataContext="{Binding Source={StaticResource ViewModel1}}">
        <TextBlock Text="{Binding Data}" />
    </Grid>
</UserControl>
```

در پروژه اصلی Solution (قسمت الف)، ارجاعاتی را به دو اسمبلی قسمت‌های ب و ج اضافه کنید. سپس MainWindow.xaml آن‌را به نحو ذیل تغییر داده و برنامه را اجرا کنید:

```
<Window x:Class="MergeAssembliesIntoWPF.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:V="clr-namespace:MergeAssembliesIntoWPF.Shell;assembly=MergeAssembliesIntoWPF.Shell"
        Title="MainWindow" Height="350" Width="525">
    <Window.Resources>
```

```
        <V:View1 x:Key="View1" />
    </Window.Resources>
    <Grid>
        <V:View1 />
    </Grid>
</Window>
```

تا اینجا باید متن Test در پنجره اصلی برنامه ظاهر شود.

ب) مدفون کردن خودکار اسمبلی‌های جانبی برنامه در فایل EXE آن

فایل csproj پروژه اصلی را خارج از VS.NET باز کنید. در انتهای آن سطر ذیل قابل مشاهده است:

```
<Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />
```

پس از این سطر، چند سطر ذیل را اضافه کنید:

```
<Target Name="AfterResolveReferences">
    <ItemGroup>
      <EmbeddedResource Include="@(ReferenceCopyLocalPaths)"
Condition="'%(ReferenceCopyLocalPaths.Extension)' == '.dll'">

<LogicalName>%(ReferenceCopyLocalPaths.DestinationSubDirectory)%(ReferenceCopyLocalPaths.Filename)%(Ref
erenceCopyLocalPaths.Extension)</LogicalName>
      </EmbeddedResource>
    </ItemGroup>
  </Target>
```

این task جدید MSBuild سبب خواهد شد تا با هر بار Build برنامه، اسمبلی‌هایی که در ارجاعات برنامه دارای خاصیت Copy local مساوی true هستند، به صورت خودکار به صورت یک resource جدید در فایل exe برنامه مدفون شوند . عموما ارجاعاتی که دستی اضافه می‌شوند، مانند دو اسمبلی یاد شده در ابتدای بحث، دارای خاصیت Copy local=true نیز هستند.

پس از این تغییر نیاز است یکبار پروژه را بسته و مجددا باز کنید. اکنون پروژه را build کنید و جهت اطمینان بیشتر آن‌را برای مثال توسط ILSpy مورد بررسی قرار دهید:

همانطور که مشاهده می‌کنید، دو اسمبلی مورد استفاده در برنامه به صورت خودکار در قسمت منابع فایل EXE مدفون شده‌اند. اگر به مسیر LogicalName تنظیمات فوق دقت کنید، DestinationSubDirectory نیز ذکر شده‌است. علت این است که بسیاری از اسمبلی‌های بومی سازی شده WPF با نام‌هایی یکسان اما در پوشه‌هایی مانند fa، fr و امثال آن ذخیره می‌شوند. به همین جهت نیاز است بین این‌ها تمایز قائل شد.

ج) بارگذاری خودکار اسمبلی‌ها در AppDomain برنامه

تا اینجا اسمبلی‌های جانبی را در فایل EXE مدفون کرده‌ایم. اکنون نوبت به بارگذاری آن‌ها در AppDomain برنامه است. برای اینکار نیاز است تا روال رخدادگردان AppDomain.CurrentDomain.AssemblyResolve را تحت نظر قرار داده و اسمبلی‌هایی را که برنامه درخواست می‌کند، در همینجا از منابع خوانده و به AppDomain اضافه کرد.
انجام اینکار در برنامه‌های WinForms ساده‌است. فقط کافی است به متد Program.Main برنامه مراجعه کرده و تعریف یاد شده را به ابتدای متد Main اضافه کرد. اما در WPF هرچند فایل App.xaml.cs به نظر نقطه‌ی آغازین برنامه است، اما در واقع اینطور نیست. برای نمونه، پوشه‌ی obj\Debug برنامه را گشوده و فایل App.g.i.cs آن‌را بررسی کنید. در اینجا می‌توانید همان رویه شبیه به برنامه‌های WinForm را در متد Program.Main آن، مشاهده کنید. بنابراین نیاز است کنترل این مساله را راسا در دست بگیریم:

```
using System;
using System.Globalization;
using System.Reflection;

namespace MergeAssembliesIntoWPF
{
    public class Program
    {
        [STAThreadAttribute]
        public static void Main()
        {
            AppDomain.CurrentDomain.AssemblyResolve += OnResolveAssembly;
            App.Main();
        }

        private static Assembly OnResolveAssembly(object sender, ResolveEventArgs args)
        {
            var executingAssembly = Assembly.GetExecutingAssembly();
            var assemblyName = new AssemblyName(args.Name);

            var path = assemblyName.Name + ".dll";
            if (assemblyName.CultureInfo.Equals(CultureInfo.InvariantCulture) == false)
            {
                path = String.Format(@"{0}\{1}", assemblyName.CultureInfo, path);
            }

            using (var stream = executingAssembly.GetManifestResourceStream(path))
            {
                if (stream == null)
                    return null;

                var assemblyRawBytes = new byte[stream.Length];
                stream.Read(assemblyRawBytes, 0, assemblyRawBytes.Length);
                return Assembly.Load(assemblyRawBytes);
            }
        }
    }
}
```

کلاس Program را با تعاریف فوق به پروژه خود اضافه نمائید. در اینجا Program.Main مورد نیاز خود را تدارک دیده‌ایم. کار آن مدیریت روال رخدادگردان AppDomain.CurrentDomain.AssemblyResolve برنامه پیش از شروع به هر کاری است. در روال رخداد گردان OnResolveAssembly، برنامه اعلام می‌کند که به چه اسمبلی خاصی نیاز دارد. ما آن‌را از قسمت منابع خوانده و سپس توسط متد Assembly.Load آن‌را در AppDomain برنامه بارگذاری می‌کنیم.
پس از اینکه کلاس فوق را اضافه کردید، نیاز است کلاس Program اضافه شده را به عنوان Startup object برنامه نیز معرفی کنید:

انجام اینکار ضروری است؛ در غیراینصورت با متد Main موجود در فایل App.g.i.cs تداخل می‌کند.

اکنون برای آزمایش برنامه، یکبار آن‌را Build کرده و بجز فایل Exe، مابقی فایل‌های موجود در پوشه‌ی bin را حذف کنید. سپس برنامه را خارج از VS.NET اجرا کنید. کار می‌کند!

MergeAssembliesIntoWPF.zip

نظرات خوانندگان

نویسنده: ژوپیتر
تاریخ: ۱۳۹۲/۱۱/۰۱ ۷:۵۷

با سلام؛ در یک پروژه ویندوزی روش **ب** را انجام دادم اما پس از Build ارجاع‌ها به بخش Resources اضافه نشد، علت چیه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۰۱ ۹:۳۵

Copy local=true در پروژه اصلی (تولید کننده فایل EXE) به این صورت تنظیم می‌شود:

نویسنده: ژوپیتر
تاریخ: ۱۳۹۲/۱۱/۰۱ ۱۰:۵۴

منظورم یک پروژه WinForms بود، نه WPF. تمامی اسمبلی‌های لازم به صورت CopyLocal هستند. بنده فقط بند ب را انجام دادم، یعنی فقط فایل CsProj را ویرایش کردم. (اگر باید الف-ج کامل انجام شوند، برای WinForms قسمت الف چگونه خواهد بود؟)
تا پیش از این از Smart Assembly برای merge استفاده می‌کردم. ولی متاسفانه این نرم‌افزار توانایی ادغام همه‌ی اسمبلی‌ها به خصوص اسمبلی‌های Third party company را ندارد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۰۱ ۱۱:۳۵

فرقی نمی‌کند. یک مثال:

MergeAssembliesIntoWinFormsApp.zip

- فایل MergeAssembliesIntoWinFormsApp.csproj آن (فایل csproj پروژه اصلی) ویرایش شده برای افزودن AfterResolveReferences قسمت ب
- فایل Program.cs استاندارد آن ویرایش شده برای افزودن تعاریف AppDomain.CurrentDomain.AssemblyResolve قسمت ج

نویسنده: ژوپیتر
تاریخ: ۱۳۹۲/۱۱/۰۱ ۱۵:۱۲

نمونه ارسالی شما به خوبی کار می‌کند.

اما برای یک پروژه عملیاتی که از کامپوننت‌های ثالث استفاده می‌کند این روش پاسخ گو نبود و با پیام Windows unhandled error متوقف می‌شود. (تمامی اسمبلی هایی که تا پیش از این کنار فایل EXE مستقر بودند و برنامه کنار آنها صحیح کار می‌کرد، با روش گفته شده در فایل EXE برنامه مدفون شدند.)

همچنین ILSpy صحت وجود را تایید کرد:

بررسی لاگ Application ویندوز خبر از پیدا نشدن فایلی (System.IO.Exception) در متد Main می‌دهد و نکته دیگری در آن ذکر نشده.

آیا راه حلی وجود دارد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۰۱ ۱۶:۳۶

- بعضی از اسمبلی‌های دات نتی Mixed mode هستند؛ مانند System.Data.SQLite.DLL. کد هسته اصلی آن، SQLite نوشته شده با زبان سی است. برای استفاده از آن در دات نت با استفاده از C++ CLI ، یک روکش دات نتی تهیه کرده‌اند تا در دات نت به راحتی قابل استفاده شود (روش مرسوم و سریعی است برای استفاده از کتابخانه‌های C و C++ در دات نت). این نوع DLLها با استفاده از روش Assembly.Load ذکر شده در متن قابل بارگذاری نیستند. باید در یک پوشه temp نوشته شده و سپس توسط Assembly.LoadFile بارگذاری شوند. یک مثال کامل در این مورد (قسمت Loading Unmanaged DLL آن مد نظر است): Load DLL From Embedded Resource

- یک try/catch در قسمت بارگذاری اسمبلی قرار دهید تا بهتر منبع مشکل را شناسایی کنید. یک مثال

- شخص دیگری در اینجا گزارش داده اگر Generate serialization assembly در قسمت تنظیمات پروژه، ذیل Build > Output فعال است، باید خاموش شود تا پروژه کرش نکند.

- اگر نوع اسمبلی، PCL است (Portable Class Library)، باز هم روش Assembly.Load به نحوی که در مطلب ذکر شده کار نمی‌کند و باید به صورت ذیل اصلاح شود:

```
private static Assembly loadEmbeddedAssembly(string name)
{
    if (name.EndsWith("Retargetable=Yes")) {
```

```
        return Assembly.Load(new AssemblyName(name));
    }
    // Rest of your code
    //...
}
```

- همچنین در کامنت‌های این مطلب شخصی عنوان کرده کرش را با افزودن ویژگی ذیل به متد Main، حل کرده:

```
[MethodImpl(MethodImplOptions.NoOptimization)]
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۰۲ ۲۲:۱۱

یک نکته‌ی تکمیلی

اگر به دنبال یک راه حل پخته‌تر هستید که با انواع و اقسام اسمبلی‌ها بتواند کار کند (از mixed mode گرفته تا pcl و غیره)، افزونه‌ی Fody / Costura توصیه می‌شود. کار با آن نیز بسیار ساده‌است. فقط کافی است دستور زیر را در کنسول پاور شل نیوگت VS.NET اجرا کنید:

```
PM> Install-Package Costura.Fody
```

بعد از نصب، تنها یکبار برنامه را مجددا build کنید.
اکنون اگر اسمبلی آن‌را بررسی کنید موارد ذیل را مشاهده خواهید کرد:

الف) اسمبلی‌های مدفون شده را zip کرده‌است.
ب) فایل pdb هم لحاظ شده.
ج) راه انداز خودکار و کدهای AssemblyResolver را تحت فضای نام Costura به فایل EXE نهایی افزوده‌است.

Fody یکی از ابزارهای AOP سورس باز دات نت است.

نویسنده: ژوپیتر
تاریخ: ۱۳۹۲/۱۱/۰۳ ۹:۱۳

راه حل بسیار جامع و ساده ای ارائه کردید که مشکلات روش‌های قبل را ندارد، برنامه به خوبی اجرا می‌شود ولی هنگام گرفتن گزارش با استفاده از stimulsoft خطای زیر ظاهر می‌شود:

(ساختار try-catch نادیده گرفته می‌شود و یک Unhandled Exception رخ می‌دهد.)

```
The type or namespace name 'Stimulsoft' could not be found (are you missing a using directive or an
assembly reference?)
```

با قرار دادن اسمبلی‌های StimulReport در کنار فایل EXE مشکل برطرف می‌شود در صورتی که این اسمبلی‌ها درون ExE مدفون هستند:

چرا برای اسمبلی‌های تلریک چنین مشکلی به وجود نمی‌آید و اینکه علاوه بر اسمبلی‌های زیپ شده خود اسمبلی‌ها نیز در فایل قرار داده شد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۰۳ ۹:۲۲

- به نظر پس از افزودن روش Fody / Costura یاد شده، هنوز تنظیمات قبلی Target Name=AfterResolveReferences که در مطلب جاری توضیح داده شد، در فایل csproj شما موجود است که باید حذف شود. دیگر نیازی به آن نیست (علت درج فایل‌های اضافی؛ چون فایل‌های Fody / Costura فقط با یک پیشوند Costura در فایل نهایی قرار می‌گیرند). همچنین تعریف قبلی AppDomain.CurrentDomain.AssemblyResolve را هم حذف کنید.

+ به احتمال زیاد اسمبلی‌های Stimulsoft فقط همین چند مورد نیستند.

همچنین این مورد را می‌توانید در bug tracker آن‌ها نیز ارسال کنید.

نویسنده: ژوپیتر
تاریخ: ۱۳۹۲/۱۱/۰۶ ۱۱:۱۴

راه حل مشکل یاد شده در اینجا

```
<Weavers>
  <Costura CreateTemporaryAssemblies='true' />
</Weavers>
```

برای مواردی که اسمبلی جاری یک اسمبلی پویا را تولید کرده و سپس ارجاعی را به خود به صورت پویا به آن اضافه می‌کند.

عنوان: **بستن یک پنجره از طریق ViewModel با استفاده از خصوصیت های پیوست شده هنگام استفاده از الگوی MVVM**
نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۱۱/۰۲ ۱۴:۲۰
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF

در نظر بگیرید که یک پروژه WPF را با الگوی MVVM پیاده سازی کرده اید و نیاز پیدا می‌کنید تا یک پنجره را از طریق کد ببندید. از آنجایی که به کنترل Window درون ViewModel دسترسی ندارید، نمی‌توانید از متد Close آن برای اینکار استفاده کنید. راه‌های مختلفی برای اینکار وجود دارند، مثلا اگر از MVVM Light Toolkit استفاده می‌کنید با ارسال یک Message و نوشتن یک تکه کد در CodeBehind پنجره می‌توانید اینکار را انجام بدهید.

اما برای اینکار یک راه حل ساده‌تری بدون نیاز به نوشتن کد در CodeBehind و استفاده از Toolkit خاصی وجود دارد و آن استفاده ازخاصیت‌های پیوست شده یا *Attached Properties است*. برای اینکار یک خاصیت از نوع Boolean مانند زیر تعریف می‌کنیم و آن را به پنجره ای که می‌خواهیم Colse شود پیوست می‌کنیم.

```
namespace TestProject.XamlServices
{
    public class CloseBehavior
    {
        public static readonly DependencyProperty CloseProperty =
DependencyProperty.RegisterAttached("Close", typeof(bool), typeof(CloseBehavior), new
UIPropertyMetadata(false, OnClose));

        private static void OnClose(DependencyObject sender, DependencyPropertyChangedEventArgs e)
        {
            if (!(e.NewValue is bool) || !((bool) e.NewValue)) return;
            var win = GetWindow(sender);
            if (win != null)
                win.Close();
        }

        private static Window GetWindow(DependencyObject sender)
        {
            Window w = null;
            if (sender is Window)
                w = (Window)sender;
            return w ?? (w = Window.GetWindow(sender));
        }

        public static bool GetClose(Window target)
        {
            return (bool)target.GetValue(CloseProperty);
        }

        public static void SetClose(DependencyObject target, bool value)
        {
            target.SetValue(CloseProperty, value);
        }
    }
}
```

در تکه کد بالا یک خصوصیت از نوع Boolean ایجاد کردیم که می‌تواند به هر پنجره ای که قرار است از طریق کد بسته شود، پیوست شود. خصوصیت‌های پیوست شده یک Callback مربوط به تغییر مقدار دارند که یک متداستاتیک است و مقدار جدید، از طریق EventArg و شیءایی که این خاصیت به آن پیوست شده نیز بعنوان Source به آن ارسال می‌شود. هر وقت مقدار خصوصیت، تغییر کند این متد فراخوانی می‌گردد. در کد بالا متد OnClose ایجاد شده است و زمانی که مقدار این خصوصیت برابر true می‌شود پنجره close خواهد شد. برای استفاده از این خصوصیت و اتصال آن باید یک خصوصیت از نوع Boolean نیز در ViewModel مربوط به Window ایجاد کنید:

```
private bool _isClose;
 public bool IsClose
   {
      get { return _isClose; }
      set
      {
       _isClose = value;
      OnClosed();
```

```
      RaisePropertyChanged("IsClose");
      }
}
```

و آن را به صورت زیر Bind کنید:

```
<Window x:Class="TestProject.TestView"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:xamlServices="clr-namespace:TestProject.XamlServices;assembly=TestProject.XamlServices"
       xamlServices:CloseBehavior.Close="{Binding IsClose}">
       ...
</Window>
```

پس از انجام اتصالات فوق، کافیست هر جایی از ViewModel که نیاز است پنجره بسته شود،مقدار این خصوصیت برابر False بشود.

نظرات خوانندگان

نویسنده: نفیسه الف
تاریخ: ۱۳۹۳/۰۳/۳۱ ۲:۷

سلام
وقتی مقدار تغییر میکنه propertychangedcallback اجرا نمیشه
onclosed() که تو set پراپرتی isclose هست از کجا اومده؟
ممنون از مطلب مفیدتون

عنوان: **حرکت روی سلول های دیتا گرید با فشردن کلید Enter در برنامه های WPF**
نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۱۱/۲۳ ۱۳:۳۵
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF, XAML

کاربران بیشتر برنامه‌های فارسی تمایل دارند که توسط کلیدEnter درون فرم‌ها حرکت کنند.در برنامه‌های WPF و مخصوصا زمانی که شما از الگوی MVVM استفاده می‌کنید ،انجام این کار اگر از روش‌های مناسب استفاده نکنید تا حدودی سخت می‌شود.
برای حرکت روی TextBoxها و کنترل‌های مشابه می‌توانید این کار را به راحتی با Register کردن رویداد مربوط به آن نوع کنترل‌ها توسط EventManager یک بار در ابتدای برنامه انجام دهید.

```
public partial class App : Application
{
EventManager.RegisterClassHandler(typeof(TextBox), TextBox.KeyDownEvent, new
KeyEventHandler(TextBox_KeyDown));
     ...
}
 private void TextBox_KeyDown(object sender, KeyEventArgs e)
   {
         if (e.Key != Key.Enter)
             return;
         var focusedElement = Keyboard.FocusedElement as TextBox;
          focusedElement.MoveFocus(new TraversalRequest(FocusNavigationDirection .Next));
   }
```

اما همانطور که در عنوان مطلب آورده شده است در این مطلب تصمیم دارم حرکت روی سلول‌های دیتا گرید توسط کلید Enter را شرح بدهم.
برای این کار نیز یک راه حل ساده وجود دارد و آن شبیه سازی فراخوانی کلید Tab هنگام فشرده شدن کلید Enter است.چون همانطور که می‌دانید کلید Tab به صورت پیش فرض حرکت روی سلول‌ها را انجام می‌دهد.برای انجام آن کافی ست دیتاگرید خود را سفارشی کرده و در متد OnPreviewKeyDown عملیات زیر را انجام دهید:

```
public class CommonDataGrid : DataGrid
{
        protected override void OnPreviewKeyDown(KeyEventArgs e)
        {
            base.OnPreviewKeyDown(e);
            if (e.Key != Key.Enter || Keyboard.PrimaryDevice.ActiveSource == null) return;
            this.CommitEdit();
            var args = new KeyEventArgs
                (Keyboard.PrimaryDevice,
                Keyboard.PrimaryDevice.ActiveSource, 0, Key.Tab) { RoutedEvent = Keyboard.KeyDownEvent
};
            InputManager.Current.ProcessInput(args);
        }
}
```

عنوان: دسترسی به Collectionها در یک ترد دیگر در WPF
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۰۷ ۱:۴۵
آدرس: www.dotnettips.info
گروه‌ها: WPF, Threading

اگر در WPF سعی کنیم آیتمی را به مجموعه اعضای یک Collection مانند یک List یا ObservableCollection از طریق تردی دیگر اضافه کنیم، با خطای ذیل متوقف خواهیم شد:

```
This type of CollectionView does not support changes to its SourceCollection
from a thread different from the Dispatcher thread
```

راه حلی که برای آن تا دات نت 4 در اکثر سایت‌ها توصیه می‌شد به نحو ذیل است:

Adding to an ObservableCollection from a background thread

مشکل!

اگر همین برنامه را که برای دات نت 4 کامپایل شده‌است، بر روی سیستمی که دات نت 4.5 بر روی آن نصب است اجرا کنیم، برنامه با خطای ذیل متوقف می‌شود:

```
System.InvalidOperationException: This exception was thrown because the generator for control
'System.Windows.Controls.ListView Items.Count:62' with name '(unnamed)' has received sequence of
CollectionChanged events that do not agree with the current state of the Items collection.
The following differences were detected:  Accumulated count 61 is different from actual count 62.
```

مشکل از کجاست؟

در دات نت 4 و نیم، دیگر نیازی به استفاده از کلاس MTObservableCollection یاد شده نیست و به صورت توکار امکان کار با Collectionها از طریق تردی دیگر میسر است. فقط برای فعال سازی آن باید نوشت:

```
private static object _lock = new object();
//...
BindingOperations.EnableCollectionSynchronization(persons, _lock);
```

پس از اینکه برای نمونه، مجموعه‌ی فرضی persons وهله سازی شد، تنها کافی است متد جدید EnableCollectionSynchronization بر روی آن فراخوانی شود.

برای برنامه‌ی دات نت 4 ایی که قرار است در سیستم‌های مختلف اجرا شود چطور؟

در اینجا باید از Reflection کمک گرفت. اگر متد EnableCollectionSynchronization بر روی کلاس BindingOperations یافت شد، یعنی برنامه‌ی دات نت 4، در محیط جدید در حال اجرا است:

```
public static void EnableCollectionSynchronization(IEnumerable collection, object lockObject)
{
      MethodInfo method = typeof(BindingOperations).GetMethod("EnableCollectionSynchronization",
             new Type[] { typeof(IEnumerable), typeof(object) });
      if (method != null)
      {
         method.Invoke(null, new object[] { collection, lockObject });
      }
}
```

در این حالت فقط کافی است این متد جدید یافت شده را بر روی Collection مدنظر فراخوانی کنیم.
همچنین اگر بخواهیم کلاس MTObservableCollection معرفی شده را جهت سازگاری با دات نت 4 و نیم به روز کنیم، به کلاس ذیل خواهیم رسید. این کلاس با دات نت 4 و 4.5 سازگار است و جهت کار با ObservableCollectionها از طریق تردهای مختلف

تهیه شده‌است:

```
using System;
using System.Collections;
using System.Collections.ObjectModel;
using System.Collections.Specialized;
using System.Linq;
using System.Windows.Data;
using System.Windows.Threading;

namespace WpfAsyncCollection
{
    public class AsyncObservableCollection<T> : ObservableCollection<T>
    {
        public override event NotifyCollectionChangedEventHandler CollectionChanged;
        private static object _syncLock = new object();

        public AsyncObservableCollection()
        {
            enableCollectionSynchronization(this, _syncLock);
        }

        protected override void OnCollectionChanged(NotifyCollectionChangedEventArgs e)
        {
            using (BlockReentrancy())
            {
                var eh = CollectionChanged;
                if (eh == null) return;

                var dispatcher = (from NotifyCollectionChangedEventHandler nh in eh.GetInvocationList()
                                  let dpo = nh.Target as DispatcherObject
                                  where dpo != null
                                  select dpo.Dispatcher).FirstOrDefault();

                if (dispatcher != null && dispatcher.CheckAccess() == false)
                {
                    dispatcher.Invoke(DispatcherPriority.DataBind, (Action)(() =>
OnCollectionChanged(e)));
                }
                else
                {
                    foreach (NotifyCollectionChangedEventHandler nh in eh.GetInvocationList())
                        nh.Invoke(this, e);
                }
            }
        }

        private static void enableCollectionSynchronization(IEnumerable collection, object lockObject)
        {
            var method = typeof(BindingOperations).GetMethod("EnableCollectionSynchronization",
                                    new Type[] { typeof(IEnumerable), typeof(object) });
            if (method != null)
            {
                // It's .NET 4.5
                method.Invoke(null, new object[] { collection, lockObject });
            }
        }
    }
}
```

در این کلاس، در سازنده‌ی آن متد عمومی enableCollectionSynchronization فراخوانی می‌شود. اگر برنامه در محیط دات نت 4 فراخوانی شود، تاثیری نخواهد داشت چون method در حال بررسی نال است. در غیراینصورت، برنامه در حالت سازگار با دات نت 4.5 اجرا خواهد شد.

نظرات خوانندگان

نویسنده: جلال
تاریخ: ۱۳۹۴/۰۱/۲۵ ۱۲:۴

چرا _syncLock به صورت static تعریف شده؟
به نظرم static بودن نه تنها غیرضروریه، بلکه حتی در مواردی (هر چند نادر) که چند نمونه‌ی جداگانه از Persons در بخش‌های مختلف برنامه همزمان در حال کار باشند، باید همه شون منتظر آزاد شدن شیء syncLock برای انجام عملیات خودشون بشن که در حقیقت نیازی نیست. Why does the lock object have to be static?

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۱/۲۵ ۱۳:۰

lock تعریف شده در اینجا، صرفا جهت کار با UI Dispatcher وجود دارد و هر برنامه یک UI Dispatcher بیشتر برای به روز رسانی UI ندارد (Application.Current.Dispatcher) و EnableCollectionSynchronization دات نت 4.5، شبیه به کار متد OnCollectionChanged نوشته شده جهت دات نت 4 را انجام می‌دهد. در متد OnCollectionChanged هم یک lock وجود دارد تا هربار از هر تردی، فقط یک مورد جهت ارسال به dispatcher برنامه، قابلیت ورود پیدا کند. هدف اصلی این است که اطلاعات دریافتی از تردهای مختلف، در ترد UI نمایش داده شوند یا به روز شوند و ترد UI هربار فقط یک آیتم را قبول می‌کند، آن هم نه از طریق تردهای دیگر. به همین جهت، این تردها باید صبر کنند تا عملیات قبلی UI خاتمه یابد.

عنوان: **آموزش WAF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۳/۰۱/۱۳ ۹:۲۵
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF, WAF

در طراحی پروژه‌های مقیاس بزرگ و البته به صورت ماژولار همیشه ساختار پروژه اهمیت به سزایی دارد. متاسفانه این مورد خیلی در طراحی پروژه‌ها در نظر گرفته نمی‌شود و اغلب اوقات شاهد آن هستیم که یک پروژه بسیار بزرگ دقیقا به همان صورت پروژهای کوچک و کم اهمیت‌تر مدیریت و پیاده سازی می‌شود که این مورد هم مربوط به پروژه‌های تحت وب و هم پروژه‌های تحت ویندوز و WPF است. برای مدیریت پروژه‌های WPF و Silverlight در این پست به اختصار درباره PRISM بحث شد. مزایا و معایب آن بررسی و در طی این پست ها(^ و ^) مثال هایی را پیاده سازی کردیم. اما در این پست مفتخرم شما را با یکی دیگر از کتابخانه‌های مربوط به پیاده سازی مدل MVVM آشنا کنم. کتابخانه ای متن باز، بسیار سبک با کارایی بالا.
اما نکته ای که ذکر آن خالی از لطف نیست این است که قبلا از این کتابخانه در یک پروژه بزرگ و ماژولار WPF استفاده کردم و نتیجه مطلوب نیز حاصل شد.
معرفی:
WPF Application Framework یا به اختصار WAF کتابخانه کم حجم سبک و البته با کارایی عالی برای طراحی پروژه‌های ماژولار WPF در مقیاس بزرگ طراحی شده است که مدل پیاده سازی ان بر مبنای مدل MVVM و MVC است. شاید برایتان جالب باشد که این کتابخانه دقیقا مدل MVC را با مدل MVVM ترکیب کرده در نتیجه مفاهیم آن بسیار شبیه به پروژه‌های تحت وب MVC است. همانطور که از نام آن پیداست این کتابخانه صرفا برای پروژه‌های WPF طراحی شده، در نتیجه در پروژه‌های Silverlight نمی‌توان از آن استفاده کرد.
ساختار کلی آن به شکل زیر می‌باشد:

همانطور که مشاهده می‌کنید پروژه‌های مبتنی بر این کتابخانه همانند سایر کتابخانه‌های MVVM از سه بخش تشکیل شده اند. بخش اول با عنوان Shell یا Presentation معرف فایل‌های Xaml پروژه است، بخش دوم یا Application معرف ViewModel و Controller و البته IView می‌باشد. بخش Domain نیز در برگیرنده مدل‌های برنامه است.

معرفی برخی مفاهیم:

»Shell : این کلاس معادل یک فایل Xaml است که حتما باید یک اینترفیس IView را پیاده سازی نماید.

»IView : معرف یک اینترفیس جهت برقراری ارتباط بین ViewModel و Shell

»ViewModel : در این جا ViewModel با مفهوم ViewModel در سایر کتابخانه‌های MVVM کمی متفاوت است. در این کتابخانه ViewModel فقط شامل تعاریف است و هیچ گونه پیاده سازی در اینجا صورت نمی‌گیرد. دقیقا معادل مفهوم ViewModel در پروژه‌های MVC تحت وب.

»Controller : پیاده سازی ViewModel و تعریف رفتارها در این قسمت انجام می‌گیرد.

اما در بسیاری از پروژها نیاز به پیاده سازی الگوی DataModel-View-ViewModel است که این کتابخانه با دراختیار داشتن برخی

کلاس‌های پایه این مهم را برایمان میسر کرده است.

همانطور که می‌بینید در این حالت بر خلاف حالت قبلی ViewModel و کنترلرهای پروژه به جای ارتباط با مدل با مفهوم DataModel تغذیه می‌شوند که یک پیاده سازی سفارشی از مدل‌های پروژه است. هم چنین این کتابخانه یک سری Converterهای سفارشی

جهت تبدیل Model به DataModel و برعکس را ارائه می‌دهد.
سرویس‌های پیش فرض: که شامل DialogBox جهت نمایش پیغام‌ها و Save|Open File Dialog سفارشی نیز می‌باشد.
»برای پیاده سازی Modularity از کتابخانه MEF استفاده شده است.
Commandهای سفارشی: پیاده سازی خاص از اینترفیس ICommand
»مفاهیم مربوط به Weak Event Pattern به صورت توکار در این کتابخانه تعبیه شده است.
»به صورت پیش فرض مباحث مربوط به اعتبارسنجی با استفاده از DataAnnotation و IDataErrorInfo در این کتابخانه تعبیه شده است.
»ارائه Extensionهای مربوط به UnitTest نظیر Exceptions و CanExecuteChangedEvent و PopertyChanged جهت سهولت در تهیه unit test

دانلود و نصب
با استفاده از nuget و دستور زیر می‌توانید این کتابخانه را نصب نمایید:

```
Install-Package waf
```

هم چنین می‌توانید سورس آن به همراه فایل‌های باینری را از اینجا دریافت کنید. در پست بعدی یک نمونه از پیاده سازی مثال با این کتابخانه را بررسی خواهیم کرد.

عنوان: **آموزش WAF (بررسی ساختار همراه با پیاده سازی یک مثال)**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۳/۰۱/۱۳ ۱۶:۳۵
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF, WAF

در این پست با مفاهیم اولیه این کتابخانه آشنا شدید. برای بررسی و پیاده سازی مثال، ابتدا یک Blank Solution را ایجاد نمایید. فرض کنید قصد پیاده سازی یک پروژه بزرگ ماژولار را داریم. برای این کار لازم است مراحل زیر را برای طراحی ساختار مناسب پروژه دنبال نمایید.

نکته: آشنایی اولیه با مفاهیم MEF از ملزومات این بخش است.

»ابتدا یک Class Library به نام Views ایجاد نمایید و اینترفیس زیر را به صورت زیر در آن تعریف نمایید. این اینترفیس رابط بین کنترلر و View از طریق ViewModel خواهد بود.

```
public interface IBookView : IView
    {
        void Show();
        void Close();
    }
```

اینترفیس IView در مسیر System.Waf.Applications قرار دارد. در نتیجه از طریق nuget اقدام به نصب Package زیر نمایید:

```
Install-Package WAF
```

»حال در Solution ساخته شده یک پروژه از نوع WPF Application به نام Shell ایجاد کنید. با استفاده از نیوگت، Waf Package را نصب نمایید؛ سپس ارجاعی از اسمبلی Views را به آن ایجاد کنید. output type اسمبلی Shell را به نوع ClassLibrary تغییر داده، همچنین فایل‌های موجود در آن را حذف نمایید. یک فایل Xaml جدید را به نام BookShell ایجاد نمایید و کدهای زیر را در آن کپی نمایید:

```
<Window x:Class="Shell.BookShell"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Book View" Height="350" Width="525">
    <Grid>
        <DataGrid ItemsSource="{Binding Books}" HorizontalAlignment="Left" Margin="10,10,0,0"
VerticalAlignment="Top" Width="400" Height="200">
            <DataGrid.Columns>
                <DataGridTextColumn Header="Code" Binding="{Binding Code}"
Width="100"></DataGridTextColumn>
                <DataGridTextColumn Header="Title" Binding="{Binding Title}"
Width="300"></DataGridTextColumn>
            </DataGrid.Columns>
        </DataGrid>

    </Grid>
</Window>
```

این فرم فقط شامل یک دیتاگرید برای نمایش اطلاعات کتاب‌هاست. دیتای آن از طریق ViewModel تامین خواهد شد، در نتیجه ItemsSource آن به خاصیتی به نام Books بایند شده است. حال ارجاعی به اسمبلی System.ComponentModel.Composition دهید. سپس در Code behind این فرم کدهای زیر را کپی کنید:

```
[Export(typeof(IBookView))]
    [PartCreationPolicy(CreationPolicy.NonShared)]
    public partial class BookShell : Window, IBookView
    {
        public BookShell()
        {
            InitializeComponent();
        }
    }
```

کاملا واضح است که این فرم اینترفیس IBookView را پیاده سازی کرده است. از آنجاکه کلاس Window به صورت پیش فرض دارای متدهای Show و Close است در نتیجه نیازی به پیاده سازی مجدد متدهای IBookView نیست. دستور Export باعث می‌شود که این کلاس به عنوان وابستگی به Composition Container اضافه شود تا در جای مناسب بتوان از آن وهله سازی کرد. نکته‌ی مهم این است که به دلیل آنکه این کلاس، اینترفیس IBookView را پیاده سازی کرده است در نتیجه نوع Export این کلاس حتما باید به صورت صریح از نوع IBookView باشد.

»یک Class Library به نام Models بسازید و بعد از ایجاد آن، کلاس زیر را به عنوان مدل Book در آن کپی کنید:

```
public class Book
    {
        public int Code { get; set; }

        public string Title { get; set; }
    }
```

»یک Class Library دیگر به نام ViewModels ایجاد کنید و همانند مراحل قبلی، Package مربوط به WAF را نصب کنید. سپس کلاسی به نام BookViewModel ایجاد نمایید و کدهای زیر را در آن کپی کنید (ارجاع به اسمبلی‌های Views و Models را فراموش نکنید):

```
[Export]
    [Export(typeof(ViewModel<IBookView>))]
    public class BookViewModel : ViewModel<IBookView>
    {
        [ImportingConstructor]
        public BookViewModel(IBookView view)
            : base(view)
        {
        }

        public ObservableCollection<Book> Books { get; set; }

    }
```

ViewModel مورد نظر از کلاس ViewModel of T ارث برده است. نوع این کلاس معادل نوع View مورد نظر ماست که در اینجا مقصود IBookView است. این کلاس شامل خاصیتی به نام ViewCore است که امکان فراخوانی متدها و خاصیت‌های View را فراهم می‌نماید. وظیفه اصلی کلاس پایه ViewModel، وهله سازی از View سپس ست کردن خاصیت DataContext در View مورد نظر به نمونه وهله سازی شده از ViewModel است. در نتیجه عملیات مقید سازی در Shell به درستی انجام خواهدشد.
به دلیل اینکه سازنده پیش فرض در این کلاس وجود ندارد حتما باید از ImportingConstructor استفاده نماییم تا CompositionContainer در هنگام عملیات وهله سازی Exception صادر نکند.

»بخش بعدی ساخت یک Class Library دیگر به نام Controllers است. در این Library نیز بعد از ارجاع به اسمبلی‌های زیر کتابخانه WAF را نصب نمایید.

Views

Models

ViewModels

System.ComponentModel.Composition

کلاسی به نام BookController بسازید و کدهای زیر را در آن کپی نمایید:

```
[Export]
    public class BookController
    {
        [ImportingConstructor]
        public BookController(BookViewModel viewModel)
        {
            ViewModelCore = viewModel;
        }

        public BookViewModel ViewModelCore
        {
```

```
            get;
            private set;
        }

        public void Run()
        {
            var result = new List<Book>();
            result.Add(new Book { Code = 1, Title = "Book1" });
            result.Add(new Book { Code = 2, Title = "Book2" });
            result.Add(new Book { Code = 3, Title = "Book3" });

            ViewModelCore.Books = new ObservableCollection<Models.Book>(result);

            (ViewModelCore.View as IBookView).Show();
        }
    }
```

نکته مهم این کلاس این است که BookViewModel به عنوان وابستگی این کنترلر تعریف شده است. در نتیجه در هنگام وهله سازی از این کنترلر Container مورد نظر یک وهله از BookViewModel را در اختیار آن قرار خواهد داد. در متد Run نیز ابتدا مقدار Book که به ItemsSource دیتا گرید در BookShell مقید شده است مقدار خواهد گرفت. سپس با فراخوانی متد Show از اینترفیس IBookView، متد Show در BookShell فراخوانی خواهد شد که نتیجه آن نمایش فرم مورد نظر است.

طراحی Bootstrapper

در پروژه‌های ماژولار Bootstrapper از ملزومات جدانشدنی این گونه پروژه هاست. برای این کار ابتدا یک WPF Application دیگر به نام Bootstrapper ایجاد نماید. سپس ارجاعی به اسمبلی‌های زیر را در آن قرار دهید:

«Controllers

«Views

«ViewModels

«Shell

«System.ComponentModel.Composition

«نصب بسته WAF با استفاده از nuget

حال یک کلاس به نام AppBootstrapper ایجاد نمایید و کدهای زیر را در آن کپی نمایید:

```
public class AppBootstrapper
    {
        public CompositionContainer Container
        {
            get;
            private set;
        }

        public AggregateCatalog Catalog
        {
            get;
            private set;
        }

        public void Run()
        {
            Catalog = new AggregateCatalog();
            Catalog.Catalogs.Add(new AssemblyCatalog(Assembly.GetExecutingAssembly()));

            Catalog.Catalogs.Add(new AssemblyCatalog(String.Format("{0}\\{1}",
Environment.CurrentDirectory, "Shell.dll")));
            Catalog.Catalogs.Add(new AssemblyCatalog(String.Format("{0}\\{1}",
Environment.CurrentDirectory, "ViewModels.dll")));
            Catalog.Catalogs.Add(new AssemblyCatalog(String.Format("{0}\\{1}",
Environment.CurrentDirectory, "Controllers.dll")));

            Container = new CompositionContainer(Catalog);

            var batch = new CompositionBatch();
            batch.AddExportedValue(Container);
            Container.Compose(batch);

            var bookController = Container.GetExportedValue<BookController>();
```

```
            bookController.Run();

        }
    }
```

اگر با MEF آشنا باشید کدهای بالا نیز برای شما مفهوم مشخصی دارند. در متد Run این کلاس ابتدا Catalog ساخته می‌شود. سپس با اسکن اسمبلی‌های مورد نظر تمام Exportها و Importهای لازم واکشی شده و به Conrtainer مورد نظر رجیستر می‌شوند. در انتها نیز با وهله سازی از BookController و فراخوانی متد Run آن خروجی زیر نمایان خواهد شد.

نکته بخش Startup را از فایل App.Xaml خذف نمایید و در متد Startup این فایل کد زیر را کپی کنید:

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

در پایان، ساختار پروژه به صورت زیر خواهد شد:

Solution 'FirstWafSample' (6 projects)
Bootstrapper
Controllers
Properties
References
BookController.cs
packages.config
Models
Properties
References
Book.cs
Shell
Properties
References
App.config
BookShell.xaml
packages.config
ViewModels
Properties
References
BookViewModel.cs
FodyWeavers.xml
packages.config
Views
Properties
References
IBookView.cs
packages.config

نکته: می‌توان بخش اسکن اسمبلی‌ها را توسط یک DirecotryCatalog به صورت زیر خلاصه کرد:

```
Catalog.Catalogs.Add(new DirectoryCatalog(Environment.CurrentDirectory));
```

در این صورت تمام اسمبلی‌های موجود در این مسیر اسکن خواهند شد.

نکته: می‌توان به جای جداسازی فیزیکی لایه‌ها آن‌ها را از طریق Directoryها به صورت منطقی در قالب یک اسمبلی نیز مدیریت کرد.

نکته: بهتر است به جای رفرنس مستقیم اسمبلی‌ها به Bootstrapper با استفاده از Pre post build در قسمت Build Event، اسمبلی‌های مورد نظر را در یک مسیر Build کپی نمایید که روش آن به تفصیل در این پست و این پست شرح داده شده است.

دانلود سورس پروژه

عنوان: آموزش WAF (بررسی Commandها)
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۳/۰۱/۱۶ ۲۲:۵۰
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF, WAF

در این پست قصد داریم مثال قسمت قبل را توسعه داده و پیاده سازی Commandها را در آن در طی یک مثال بررسی کنیم. از این جهت دکمه‌ای، جهت حذف آیتم انتخاب شده در دیتا گرید، به فرم BookShell اضافه می‌نماییم. به صورت زیر:

```
<Button Content="RemoveItem" Command="{Binding RemoveItemCommand}" HorizontalAlignment="Left"
VerticalAlignment="Top" Width="75"/>
```

Command تعریف شده در Button مورد نظر به خاصیتی به نام RemoveItemCommand در BookViewModel که نوع آن ICommand است اشاره می‌کند. پس باید تغییرات زیر را در ViewModel اعمال کنیم:

```
public ICommand RemoveItemCommand { get; set; }
```

از طرفی نیاز به خاصیتی داریم که به آیتم جاری در دیتاگرید اشاره کند.

```
public Book CurrentItem
        {
            get
            {
                return currentItem;
            }
            set
            {
                if(currentItem != value)
                {
                    currentItem = value;
                    RaisePropertyChanged("CurrentItem");
                }
            }
        }
        private Book currentItem;
```

همان طور که در پست قبلی توضیح داده شد پیاده سازی‌ها تعاریف ViewModel در Controller انجام می‌گیرد برای همین منظور باید تعریف DelegateCommand که یک پیاده سازی خاص از ICommand است در کنترلر انجام شود. :

```
[Export]
    public class BookController
    {
        [ImportingConstructor]
        public BookController(BookViewModel viewModel)
        {
            ViewModelCore = viewModel;
        }

        public BookViewModel ViewModelCore
        {
            get;
            private set;
        }

        public DelegateCommand RemoveItemCommand
        {
            get;
            private set;
        }

        private void ExecuteRemoveItemCommand()
        {
            ViewModelCore.Books.Remove(ViewModelCore.CurrentItem);
        }

        private void Initialize()
        {
            RemoveItemCommand = new DelegateCommand(ExecuteRemoveItemCommand);
            ViewModelCore.RemoveItemCommand = RemoveItemCommand;
```

```
        }

        public void Run()
        {

            var result = new List<Book>();
            result.Add(new Book { Code = 1, Title = "Book1" });
            result.Add(new Book { Code = 2, Title = "Book2" });
            result.Add(new Book { Code = 3, Title = "Book3" });

            Initialize();

            ViewModelCore.Books = new ObservableCollection<Models.Book>(result);

            (ViewModelCore.View as IBookView).Show();
        }
    }
```

تغییرات:

»خاصیتی به نام RemoveItemCommand که از نوع DelegateCommand است تعریف شده است؛

»متدی به نام Initialize اضافه شد که متدهای Execute و CanExecute برای Commandها را در این قسمت رجیستر می‌کنیم.

»در نهایت Command تعریف شده در کنترلر به Command مربوطه در ViewModel انتساب داده شد.

حال کافیست خاصیت SelectedItem دیتاگرید BookShell به خاصیت CurrentItem موجود در ViewModel مقید شود:

```
<DataGrid ItemsSource="{Binding Books}" SelectedItem="{Binding CurrentItem
,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}" HorizontalAlignment="Left" Margin="10,10,0,0"
VerticalAlignment="Top" Width="400" Height="200">
            <DataGrid.Columns>
                <DataGridTextColumn Header="Code" Binding="{Binding Code}"
Width="100"></DataGridTextColumn>
                <DataGridTextColumn Header="Title" Binding="{Binding Title}"
Width="300"></DataGridTextColumn>
            </DataGrid.Columns>
        </DataGrid>
```

اگر پروژه را اجرا نمایید، بعد از انتخاب سطر مورد نظر و کلیک بر روی دکمه RemoveItem مورد زیر قابل مشاهده است:

```
1 reference
private void ExecuteRemoveItemCommand()
{
    ViewModelCore.Books.Remove(ViewModelCore.CurrentItem);
}
```

عنوان: آموزش WAF (مشاهده تغییرات خواص ViewModel در Controller)
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۳/۰۱/۱۷ ۹:۳۵
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF, WAF

قصد داریم در مثال پست قبلی برای Command مورد نظر، عملیات اعتبارسنجی را فعال کنیم. اگر با الگوی MVVM آشنایی داشته باشید می‌دانید که می‌توان برای Commandها اکشنی به عنوان CanExecute تعریف کرد و در آن عملیات اعتبارسنجی را انجام داد. اما از آن جا که پیاده سازی این روش زمانی مسیر است که تغییرات خواص ViewModel در دسترس باشد در نتیجه در WAF مکانیزمی جهت ردیابی تغییرات خواص ViewModel در کنترلر فراهم شده است. در نسخه‌های قبلی WAF (قبل از نسخه 3) هر کنترلر از کلاس پایه ای به نام Controller ارث می‌برد که متد هایی جهت ردیابی تغییرات در آن در نظر گرفته شده بود به صورت زیر:

```
public class MyController : Controller
    {
        [ImportingConstructor]
        public MyController(MyViewModel viewModel)
        {
            ViewModelCore = viewModel;
        }

        public MyViewModel ViewModelCore
        {
            get;
            private set;
        }

        public void Run()
        {
            AddWeakEventListener(ViewModelCore , ViewModelCoreChanged)
        }

        private void ViewModelCoreChanged(object sender , PropertyChangedEventArgs e)
        {
            if(e.PropertyName=="CurrentItem")
            {

            }
        }
    }
```

همان طور که مشاهده می‌کنید با استفاده از متد AddWeakEventListener توانستیم تمامی تغییرات خواص ViewModel مورد نظر را از طریق متد ViewModelCoreChanged ردیابی کنیم. این متد بر مبنای الگوی WeakEvent پیاده سازی شده است. البته این تغییرات فقط زمانی قابل ردیابی هستند که در ViewModel متد RaisePropertyChanged برای متد set خاصیت فراخوانی شده باشد.

از آنجا که در دات نت 4.5 یک پیاده سازی خاص از الگوی WeakEvent در کلاس PropertyChangedEventManager موجود در اسمبلی WindowsBase و فضای نام System.ComponentModel انجام شده است در نتیجه توسعه دهندگان این کتابخانه نیز تصمیم به استفاده از این روش گرفتند که نتیجه آن Obsolete شدن کلاس پایه کنترلر در نسخه‌های 3 به بعد آن است. در روش جدید کافیست به صورت زیر عمل نمایید:

```
[Export]
    public class BookController
    {
        [ImportingConstructor]
        public BookController(BookViewModel viewModel)
        {
            ViewModelCore = viewModel;
        }

        public BookViewModel ViewModelCore
        {
            get;
            private set;
        }

        public DelegateCommand RemoveItemCommand
```

```
        {
            get;
            private set;
        }

        private void ExecuteRemoveItemCommand()
        {
            ViewModelCore.Books.Remove(ViewModelCore.CurrentItem);
        }

        private bool CanExecuteRemoveItemCommand()
        {
            return ViewModelCore.CurrentItem != null;
        }
        private void Initialize()
        {
            RemoveItemCommand = new DelegateCommand(ExecuteRemoveItemCommand ,
CanExecuteRemoveItemCommand);
            ViewModelCore.RemoveItemCommand = RemoveItemCommand;
        }

        public void Run()
        {
            var result = new List<Book>();
            result.Add(new Book { Code = 1, Title = "Book1" });
            result.Add(new Book { Code = 2, Title = "Book2" });
            result.Add(new Book { Code = 3, Title = "Book3" });

            Initialize();
            ViewModelCore.Books = new ObservableCollection<Models.Book>(result);

            PropertyChangedEventManager.AddHandler(ViewModelCore, ViewModelChanged, "CurrentItem");

            (ViewModelCore.View as IBookView).Show();
        }

        private void ViewModelChanged(object sender,PropertyChangedEventArgs e)
        {
            if(e.PropertyName == "CurrentItem")
            {
                RemoveItemCommand.RaiseCanExecuteChanged();
            }
        }
    }
```

تغییرات:

»ابتدا متدی به نام CanExecuteRemoveItemCommand ایجاد کردیم و کدهای اعتبارسنجی را در آن قرار دادیم؛

»هنگام تعریف Command مربوطه متد بالا را به DelegateCommand رجیستر کردیم:

```
RemoveItemCommand = new DelegateCommand(ExecuteRemoveItemCommand , CanExecuteRemoveItemCommand);
```

در این حالت بعد از اجرای برنامه همواره دکمه RemoveItem غیر فعال خواهد بود. دلیل آن این است که بعد از انتخاب آیتم مورد نظر از لیست باید کنترلر را متوجه تغییر در مقدار خاصیت CurrentItem نماییم. بدین منظور کد زیر را به متد Run اضافه کردم:

```
PropertyChangedEventManager.AddHandler(ViewModelCore, ViewModelChanged, "CurrentItem");
```

دستور بالا دقیقا معادل دستور AddWeakEventListener موجود در نسخه‌های قدیمی WAF است. سپس در صورتی که نام خاصیت مورد نظر CurrentItem بود با استفاده از دستور RaiseCanExecuteChanged در کلاس DelegateCommand کنترلر را ملزم به اجرای دوباره متد CanExecuteRemoveItemCommand می‌کنیم.

اجرای برنامه:

ابتدا دکمه RemoveItem غیر فعال است:

بعد از انتخاب یکی از گزینه و فراخوانی مجدد متد CanExecuteRemoveItemCommand دکمه مورد نظر فعال می‌شود:

```
private bool CanExecuteRemoveItemCommand()
{
    return ViewModelCore.CurrentItem != null;
}
```

و در نهایت دکمه RemoveItem فعال خواهد شد:

دانلود سورس پروژه

عنوان: **تصادفی کردن آیتمهای لیست با استفاده از Extension Method**
نویسنده: بهزاد دات نت
تاریخ: ۱۳۹۳/۰۲/۰۷ ۱۴:۱۵
آدرس: www.dotnettips.info
گروه‌ها: LINQ, WPF, Extension Method, C#.NET

شاید برای شما هم پیش آمده باشد که بخواهید در هر بار واکشی لیستی از اطلاعات، مثلا از دیتابیس، آیتمهای آن را بصورت تصادفی مرتب کنید.
من در پروژه اخیرم برای نمایش یک سری سوال مجبور بودم که در هر بار نمایش سوالات، لیست را به صورت رندوم مرتب کنم و به کاربر نمایش بدم. برای حصول این مهم، یک extension method به شکل زیر نوشتم:

```
public static class RandomExtentions
    {
        public static void Shuffle<T>(this IList<T> list)
        {
            Random rng = new Random();
            Thread.Sleep(100);
            int n = list.Count;
            while (n > 1)
            {
                n--;
                int k = rng.Next(n + 1);
                T value = list[k];
                list[k] = list[n];
                list[n] = value;
            }
        }
    }
```

در این تابع که اسمش را Shuffle گذاشتم، با دریافت یک لیست از نوع T، آیتم‌های درون لیست را به صورت تصادفی مرتب می‌کند.

مثال :

```
var x =new  List<int>();
x.Add(1);
x.Add(2);
x.Add(3);
x.Add(4);
x.Add(5);
x.Shuffle();
```

در این مثال لیست x که از نوع int میباشد پس از فراخوانی Shuffle به یک لیست نامرتب تبدیل میشود که نحوه چیدمان در هر بار فراخوانی، تصادفی خواهد بود.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۰۷ ۱۴:۱۶

اگر از EF استفاده می‌کنید، برای اینکار یک ستون Guid پویا را اضافه می‌کند. سپس بر اساس این ستون، مرتب سازی را انجام می‌دهد. اطلاعات بیشتر

نویسنده: بهزاد دات نت
تاریخ: ۱۳۹۳/۰۲/۰۷ ۱۴:۳۰

با سپاس از شما. در صورت استفاده ار EF روشی که شما فرمودین بهتر و کارامدتر هستش.

عنوان: استفاده از AvalonEdit در WPF
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۵/۲۷ ۱۲:۴۰
آدرس: www.dotnettips.info
گروه‌ها: WPF

AvalonEdit یکی از زیرساخت‌های برنامه‌ی SharpDevelop است که ویرایشگر متنی به همراه syntax highlighting زبان‌های مختلف را در آن پشتیبانی می‌کند. کیفیت بالایی داشته و بسیاری از برنامه‌های دیگر نیز از آن جهت ارائه ویرایشگر و یا syntax highlighting متون ارائه شده، استفاده می‌کنند. در ادامه نحوه‌ی استفاده از این ویرایشگر را در برنامه‌های WPF خصوصا با دید MVVM بررسی خواهیم کرد.

دریافت و نصب AvalonEdit

برای نصب AvalonEdit می‌توان دستور ذیل را در کنسول پاورشل نیوگت صادر کرد:

```
PM> install-package AvalonEdit
```

استفاده‌ی مقدماتی از AvalonEdit

برای استفاده از این ویرایشگر ابتدا نیاز است فضای نام xmlns:avalonEdit تعریف شود. سپس کنترل avalonEdit:TextEditor در دسترس خواهد بود:

```
<Window x:Class="SyntaxHighlighter.MainWindow"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:avalonEdit="http://icsharpcode.net/sharpdevelop/avalonedit"
   Title="MainWindow" Height="401" Width="617">
  <Grid>
<avalonEdit:TextEditor
    Name="txtCode"
    SyntaxHighlighting="C#"
  FontFamily="Consolas"
  FontSize="10pt"/>
  </Grid>
</Window>
```

توسط خاصیت SyntaxHighlighting آن می‌توان زبان مشخصی را تعریف کرد. لیست زبان‌های توکار پشتیبانی شده

استفاده از AvalonEdit در برنامه‌های MVVM

خاصیت Text این ویرایشگر به صورت معمولی تعریف شده (DependencyProperty نیست) و امکان binding دو طرفه به آن وجود ندارد. به همین جهت نیاز است یک چنین DependencyProperty را به آن اضافه کرد:

```
using System;
using System.Collections.Concurrent;
using System.Reflection;
using System.Windows;
using System.Xml;
using ICSharpCode.AvalonEdit;
using ICSharpCode.AvalonEdit.Highlighting;
using ICSharpCode.AvalonEdit.Highlighting.Xshd;

namespace AvalonEditWpfTest.Controls
{
    public class BindableAvalonTextEditor : TextEditor
    {
        public static readonly DependencyProperty BoundTextProperty =
            DependencyProperty.Register("BoundText",
                typeof(string),
                typeof(BindableAvalonTextEditor),
                new FrameworkPropertyMetadata(default(string), propertyChangedCallback));

        public static string GetBoundText(DependencyObject obj)
        {
            return (string)obj.GetValue(BoundTextProperty);
        }

        public static void SetBoundText(DependencyObject obj, string value)
        {
            obj.SetValue(BoundTextProperty, value);
        }

        protected override void OnTextChanged(EventArgs e)
        {
            SetCurrentValue(BoundTextProperty, Text);
            base.OnTextChanged(e);
        }

        private static void propertyChangedCallback(DependencyObject obj,
                                    DependencyPropertyChangedEventArgs args)
        {
            var target = (BindableAvalonTextEditor)obj;
            var value = args.NewValue;
            if (value == null)
                return;

            if (string.IsNullOrWhiteSpace(target.Text) ||
                !target.Text.Equals(args.NewValue.ToString()))
            {
                target.Text = args.NewValue.ToString();
            }
        }
    }
}
```

کار با ارث بری از TextEditor (ویرایشگر AvalonEdit) شروع می‌شود. سپس یک DependencyProperty به نام BoundText در اینجا اضافه شده‌است. هر زمان که متن داخل آن تغییر کرد، آن‌را به خاصیت متنی Text این ویرایشگر نسبت می‌دهد. به این ترتیب binding یک طرفه (از کدها به کنترل) کار می‌کند. فعال سازی binding دو طرفه با پشتیبانی از انتقال تغییرات از ویرایشگر به خواص ViewModel در متد بازنویسی شده‌ی OnTextChanged انجام می‌شود.

اکنون برای استفاده از این کنترل جدید که BindableAvalonTextEditor نام دارد، می‌توان به نحو ذیل عمل کرد:

```
<Window x:Class="AvalonEditWpfTest.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:viewModels="clr-namespace:AvalonEditTests.ViewModels"
        xmlns:controls="clr-namespace:AvalonEditWpfTest.Controls"
        Title="MainWindow" Height="350" Width="525">
    <Window.Resources>
        <viewModels:MainWindowViewModel x:Key="MainWindowViewModel"/>
    </Window.Resources>
```

```
    <Grid DataContext="{Binding Source={StaticResource MainWindowViewModel}}">
        <controls:BindableAvalonTextEditor
                BoundText="{Binding SourceCode, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
                        WordWrap="True"
                        ShowLineNumbers="True"
                        LineNumbersForeground="MediumSlateBlue"
                        FontFamily="Consolas"
                        VerticalScrollBarVisibility="Auto"
                        Margin="3"
                        HorizontalScrollBarVisibility="Auto"
                        FontSize="10pt"/>
    </Grid>
</Window>
```

ابتدا فضای نام جدید کنترل BindableAvalonTextEditor مشخص می‌شود و سپس به controls:BindableAvalonTextEditor دسترسی خواهیم داشت. در اینجا نحوه‌ی استفاده از خاصیت جدید BoundText را نیز مشاهده می‌کنید.

افزودن syntax highlighting زبان‌هایی که به صورت رسمی پشتیبانی نمی‌شوند

به خاصیت SyntaxHighlighting این کنترل صرفا مقادیری را می‌توان نسبت داد که به صورت توکار پشتیبانی می‌شوند. برای مثال#C، XML و امثال آن.
فرض کنید نیاز است SyntaxHighlighting زبان SQL را فعال کنیم. برای اینکار نیاز به فایل‌های ویژه‌ای است، با پسوند xshd.
برای نمونه فایل sql-ce.xshd را در اینجا می‌توانید مطالعه کنید. در آن یک سری واژه‌های کلیدی و حروفی که باید با رنگی متفاوت نمایش داده شوند، مشخص می‌گردند.
برای استفاده از فایل sql-ce.xshd باید به نحو ذیل عمل کرد:
الف) فایل sql-ce.xshd را به پروژه اضافه کرده و سپس در برگه‌ی خواص آن در VS.NET، مقدار build action آن‌را به embedded resource تغییر دهید.

ب) با استفاده از متد ذیل، این فایل مدفون شده در اسمبلی را گشوده و به متد HighlightingLoader.Load ارسال می‌کنیم:

```
private static IHighlightingDefinition getHighlightingDefinition(string resourceName)
        {
            if (string.IsNullOrWhiteSpace(resourceName))
                throw new NullReferenceException("Please specify SyntaxHighlightingResourceName.");

            using (var stream =
Assembly.GetExecutingAssembly().GetManifestResourceStream(resourceName))
            {
                if (stream == null)
                    throw new NullReferenceException(string.Format("{0} resource is null.",
resourceName));

                using (var reader = new XmlTextReader(stream))
                {
                    return HighlightingLoader.Load(reader, HighlightingManager.Instance);
                }
            }
        }
```

نحوه استفاده از آن نیز به صورت ذیل است:

```
txtCode.SyntaxHighlighting = getHighlightingDefinition(resourceName);
```

به این ترتیب می‌توان یک فایل xhsd را به صورت پویا بارگذاری و به خاصیت SyntaxHighlighting کنترل انتساب داد.

برای سهولت استفاده از این قابلیت شاید بهتر باشد یک DependencyProperty دیگر به نام SyntaxHighlightingResourceName را به کنترل جدید BindableAvalonTextEditor اضافه کنیم:

```
using System;
using System.Collections.Concurrent;
using System.Reflection;
using System.Windows;
using System.Xml;
using ICSharpCode.AvalonEdit;
using ICSharpCode.AvalonEdit.Highlighting;
using ICSharpCode.AvalonEdit.Highlighting.Xshd;

namespace AvalonEditWpfTest.Controls
{
    public class BindableAvalonTextEditor : TextEditor
    {
         public static readonly DependencyProperty SyntaxHighlightingResourceNameProperty =
           DependencyProperty.Register("SyntaxHighlightingResourceName",
               typeof(string),
               typeof(BindableAvalonTextEditor),
               new FrameworkPropertyMetadata(default(string), resourceNamePropertyChangedCallback));

        public static string GetSyntaxHighlightingResourceName(DependencyObject obj)
        {
            return (string)obj.GetValue(SyntaxHighlightingResourceNameProperty);
        }

        public static void SetSyntaxHighlightingResourceName(DependencyObject obj, string value)
        {
            obj.SetValue(SyntaxHighlightingResourceNameProperty, value);
        }

        private static void loadHighlighter(TextEditor @this, string resourceName)
        {
            if (@this.SyntaxHighlighting != null)
                return;

            @this.SyntaxHighlighting = getHighlightingDefinition(resourceName);
        }

        private static void resourceNamePropertyChangedCallback(DependencyObject obj,
                                        DependencyPropertyChangedEventArgs args)
        {
            var target = (BindableAvalonTextEditor)obj;
            var value = args.NewValue;
            if (value == null)
                return;
```

```
            loadHighlighter(target, value.ToString());
        }
    }
}
```

کاری که در اینجا انجام شده، افزودن یک خاصیت جدید به نام SyntaxHighlightingResourceName به کنترل BindableAvalonTextEditor است. هر زمانیکه مقدار آن تغییر کند، متد getHighlightingDefinition بحث شده، فراخوانی گردیده و به صورت پویا مقدار خاصیت SyntaxHighlighting این کنترل، مقدار دهی می‌شود.
استفاده از آن نیز به شکل زیر است:

```
<controls:BindableAvalonTextEditor
SyntaxHighlightingResourceName = "AvalonEditWpfTest.Controls.sql-ce.xshd"
/>
```

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید:

AvalonEditWpfTest.zip

عنوان: **فرمت شرطی اطلاعات به کمک تریگرها در WPF**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۶/۳۱ ۱۰:۱۵
آدرس: www.dotnettips.info
گروه‌ها: WPF

فرض کنید در یک لیست، تعداد زیادی صفر وجود دارند و تنها معدودی از آن‌ها دارای مقداری متفاوت هستند. شاید بد نباشد برای کاهش نویز صفحه، صفرها نمایش داده نشوند و در کل لیست، فقط مقادیر بیشتر از صفر مشخص باشند. برای اینکار راه حل‌های زیادی وجود دارند؛ منجمله، استفاده از تبدیلگرها. اما با استفاده از تریگرهای WPF اینکار را با چند سطر کد ساده، در همان فایل XAML یا یک شیوه‌نامه جدید می‌توان انجام داد.

تعریف تریگر مخفی سازی یک برچسب

```
<Style TargetType="TextBlock" x:Key="TextBlockStyle1">
            <Style.Triggers>
                <Trigger Property="Text" Value="0">
                    <Setter Property="Visibility" Value="Collapsed" />
                </Trigger>
            </Style.Triggers>
        </Style>
```

برای تعریف تریگر، ابتدا در یک Style جدید، مشخص می‌کنیم که اطلاعات تعریف شده باید به چه نوع المانی اعمال شوند. سپس در قسمت Style.Triggers تعیین می‌کنیم که اگر خاصیت Text این المان مساوی صفر بود، مقدار خاصیت Visibility آن به Collapsed تغییر یابد تا مخفی شود.

این تعاریف در مورد یک TextBlock بود. برای کنترل Label به علت نداشتن خاصیت Text و داشتن خاصیت Content می‌توان به نحو ذیل عمل کرد:

```
<Style TargetType="Label" x:Key="LabelStyle1">
            <Style.Triggers>
                <Trigger Property="Content">
                    <Trigger.Value>
                        <system:Int32>0</system:Int32>
                    </Trigger.Value>
                    <Setter Property="Visibility" Value="Collapsed" />
                </Trigger>
            </Style.Triggers>
        </Style>
```

چون خاصیت Content می‌تواند یک object نیز باشد، توسط Trigger.Value مقدار آن به یک Int32 تبدیل شده و سپس بر این مبنا تصمیم گیری می‌شود.

برای اعمال آن‌ها نیز می‌توان به نحو ذیل عمل کرد:

```
<TextBlock Text="{Binding Rating, Mode=OneTime}"
Style="{StaticResource TextBlockStyle1}" />
```

با اعمال این تریگر، مقادیر صفر در ستون rating نمایش داده نخواهند شد.

یک مثال کامل را در این زمینه از اینجا می‌توانید دریافت کنید

WpfVisibilityTriggers.zip

برای مطالعه بیشتر

Trigger, DataTrigger & EventTrigger

WPF MultiTrigger and MultiDataTrigger

عنوان: first chance exception چیست؟
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۶/۳۱ ۱۲:۱۰
آدرس: www.dotnettips.info
گروه‌ها: WPF, exception handling

چند سال قبل یک datapicker تقویم شمسی را برای سیلورلایت تهیه کردم. بعد از آن نسخه‌ی WPF آن هم به پروژه اضافه شد . تا اینکه مدتی قبل مشکل عدم کار کردن آن در یک صفحه‌ی دیالوگ جدید در ویندوز 8 گزارش شد. در حین برطرف کردن این مشکل، مدام سطر ذیل در پنجره‌ی output ویژوال استودیو نمایش داده می‌شد:

```
A first chance exception of type 'System.ArgumentOutOfRangeException' occurred in mscorlib.dll
```

البته برنامه بدون مشکل کار می‌کرد و صفحه‌ی نمایش Exception در VS.NET ظاهر نمی‌شد.

سؤال: first chance exception چیست؟

وقتی استثنایی در یک برنامه رخ می‌دهد، به آن یک first chance exception می‌گویند. این اولین شانسی است که سیستم به شما می‌دهد تا استثنای رخ داده را مدیریت کنید. اگر کدهای برنامه یا ابزاری (یک try/catch یا دیباگر) این اولین شانس را ندید بگیرند، یک second chance exception رخ می‌دهد. این‌جا است که برنامه به احتمال زیاد خاتمه خواهد یافت.
مشاهده‌ی پیام‌های A first chance exception در پنجره‌ی output ویژوال استودیو به این معنا است که استثنایی رخ داده، اما توسط یک استثناءگردان مدیریت شده‌است. بنابراین در اکثر موارد، موضوع خاصی نیست و می‌توان از آن صرفنظر کرد.

سؤال: چگونه می‌توان منشاء اصلی پیام رخ‌دادن یک first chance exception را یافت؟

ویژوال استودیو در پنجره‌ی output، مدام پیام رخ‌دادن first chance exception را نمایش می‌دهد؛ اما واقعا کدام قطعه از کدهای برنامه سبب بروز آن شده‌اند؟ به صورت پیش فرض صفحه‌ی نمایش استثناءها در VS.NET زمانی نمایان می‌شود که استثنای رخ داده، مدیریت نشده باشد. برای فعال سازی نمایش استثناهای مدیریت شده باید تنظیمات ذیل را اعمال کرد:
- به منوی Debug | Exceptions مراجعه کنید.
- گره Common Language Runtime Exceptions را باز کنید.
- سپس گروه System آن‌را نیز باز کنید.
- در اینجا بر اساس نوع استثنایی که در پنجره‌ی output نمایش داده می‌شود، آن استثناء را یافته و Thrown آن‌را انتخاب کنید.

اینبار اگر برنامه را اجرا کنید، دقیقا محلی که سبب بروز استثنای ArgumentOutOfRangeException شده در VS.NET گزارش داده خواهد شد.

عنوان: **معرفی کتابخانه‌ی OxyPlot**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۲۳ ۱۶:۰
آدرس: www.dotnettips.info
گروه‌ها: WPF, chart, OxyPlot

برای ترسیم نمودار در برنامه‌های WPF، چندین کتابخانه‌ی سورس باز مانند Dynamic Data Display ، Sparrow Toolkit ، GraphIT و ... OxyPlot وجود دارند. در بین این‌ها، کتابخانه‌ی OxyPlot دارای این مزایا است:

- دارای مجوز MIT است. (مجاز هستید از آن در هر نوع برنامه‌ای استفاده کنید)
- cross-platform است. به این معنا که دات نت، WinRT و Xamarin را به خوبی پشتیبانی می‌کند.
- WPF و همچنین WinForms تا Xamarin.Android را پوشش می‌دهد.
- بسته‌های اصلی NuGet آن تا به امروز نزدیک به 40 هزار بار دریافت شده‌اند.
- انجمن فعالی دارد .
- بسیار بسیار غنی است. تا حدی که مرور سطحی مجموعه مثال‌های آن شاید چند ساعت وقت را به خود اختصاص دهد.
- طراحی آن به نحوی است که با الگوی MVVM کاملا سازگاری دارد.
- به صورت متناوبی به روز شده و نگهداری می‌شود.

این برنامه (تصویر فوق) که حاوی مرورگر مثال‌های آن است، در پوشه‌ی Source\Examples\WPF\ExampleBrowser سورس‌های آن قرار دارد.

در ادامه نگاهی خواهیم داشت به نحوه‌ی استفاده از OxyPlot در برنامه‌های WPF جهت رسم نموداری بلادرنگ که اطلاعات آن در زمان اجرای برنامه تهیه شده و در همین حین نیز تغییر می‌کنند.

دریافت بسته‌های نیوگت OxyPlot

برای دریافت دو بسته‌ی OxyPlot.Core و OxyPlot.Wpf تنها کافی است دستور ذیل را در کنسول پاورشل نیوگت اجرا کنیم:

```
PM> install-package OxyPlot.Wpf
```

افزودن تعاریف چارت به View

```
<Window x:Class="OxyPlotWpfTests.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:oxy="http://oxyplot.org/wpf"
        xmlns:oxyPlotWpfTests="clr-namespace:OxyPlotWpfTests"
        Title="MainWindow" Height="350" Width="525">
    <Window.Resources>
        <oxyPlotWpfTests:MainWindowViewModel x:Key="MainWindowViewModel" />
    </Window.Resources>
    <Grid DataContext="{Binding Source={StaticResource MainWindowViewModel}}">
        <oxy:PlotView Model="{Binding PlotModel}"/>
    </Grid>
</Window>
```

ابتدا باید فضای نام oxy اضافه شود. پس از آن oxy:PlotView به صفحه اضافه شده و سپس Model آن از ViewModel برنامه تغذیه می‌گردد.

ساختار کلی ViewModel برنامه

کار ViewModel متصل شده به View فوق، مقدار دهی PlotModel است.

```
public class MainWindowViewModel
{
   public PlotModel PlotModel { get; set; }
```

یک نکته‌ی کاربردی

اگر هیچ ایده‌ای نداشتید که این PlotModel را چگونه باید مقدار دهی کرد، به همان برنامه‌ی ExampleBrowser ابتدای مطلب مراجعه کنید.

مثال‌های اجرای شده‌ی آن یک برگه‌ی نمایشی و یک برگه‌ی Code دارند. خروجی این متدها را اگر به خاصیت PlotModel فوق انتساب دهید ... یک چارت کامل خواهید داشت.

مراحل ساخت یک PlotModel

ابتدا نیاز است یک وهله‌ی جدید از PlotModel را ایجاد کنیم:

```
private void createPlotModel()
        {
            PlotModel = new PlotModel
            {
                Title = "سری خطوط",
                Subtitle = "Pan (right click and drag)/Zoom (Middle click and drag)/Reset (double-
click)"
            };
            PlotModel.MouseDown += (sender, args) =>
            {
                if (args.ChangedButton == OxyMouseButton.Left && args.ClickCount == 2)
                {
                    foreach (var axis in PlotModel.Axes)
                        axis.Reset();

                    PlotModel.InvalidatePlot(false);
                }
            };
        }
```

PlotModel در برگیرنده‌ی محورها، نقاط و تمام ناحیه‌ی چارت است. در اینجا عنوان و زیرعنوان نمودار، مقدار دهی شده‌اند.
همچنین در همین ViewModel بدون نیاز به مراجعه به View، می‌توان به رخدادهای مختلف OxyPlot دسترسی داشت. برای مثال می‌خواهیم اگر کاربر دو بار بر روی چارت کلیک کرد، کلیه اعمال zoom و pan آن به حالت اول برگردانده شوند.
برای pan، کافی است دکمه‌ی سمت راست ماوس را نگه داشته و بکشید. به این ترتیب می‌توانید نمودار را بر روی محورهای X و Y حرکت دهید.
برای zoom نیاز است دکمه‌ی وسط ماوس را نگه داشته و بکشید. ناحیه‌ای که در این حالت نمایان می‌گردد، محل بزرگنمایی نهایی

خواهد بود.

این دو قابلیت به صورت توکار در OxyPlot قرار دارند و نیازی به کدنویسی برای فعال سازی آن‌ها نیست.

افزودن محورهای X و Y

محور X در مثال ما، از نوع LinearAxis است. بهتر است متغیر آن‌را در سطح کلاس تعریف کرد تا بتوان از آن در سایر قسمت‌های چارت نیز بهره گرفت:

```
readonly LinearAxis _xAxis = new LinearAxis();
        private void addXAxis()
        {
            _xAxis.Minimum = 0;
            _xAxis.MaximumPadding = 1;
            _xAxis.MinimumPadding = 1;
            _xAxis.Position = AxisPosition.Bottom;
            _xAxis.Title = "X axis";
            _xAxis.MajorGridlineStyle = LineStyle.Solid;
            _xAxis.MinorGridlineStyle = LineStyle.Dot;
            PlotModel.Axes.Add(_xAxis);
        }
```

در اینجا مقدار خاصیت Position، مشخص می‌کند که این محور در کجا باید قرار گیرد. اگر مقدار دهی نشود، محور Y را تشکیل خواهد داد.

مقدار دهی GridlineStyleها سبب ایجاد یک Grid خاکستری در نمودار می‌شوند.

در آخر نیاز است این محور به محورهای PlotModel اضافه شود.

تعریف محور Y نیز به همین نحو است. اگر مقدار خاصیت Position ذکر نشود، این محور در سمت چپ صفحه قرار می‌گیرد:

```
readonly LinearAxis _yAxis = new LinearAxis();
        private void addYAxis()
        {
            _yAxis.Minimum = 0;
            _yAxis.Title = "Y axis";
            _yAxis.MaximumPadding = 1;
            _yAxis.MinimumPadding = 1;
            _yAxis.MajorGridlineStyle = LineStyle.Solid;
            _yAxis.MinorGridlineStyle = LineStyle.Dot;
            PlotModel.Axes.Add(_yAxis);
        }
```

افزودن تعاریف سری‌های خطوط

در تصویر فوق، دو سری خط را ملاحظه می‌کنید. تعاریف پایه سری اول آن به این صورت است:

```
readonly LineSeries _lineSeries1 = new LineSeries();
        private void addLineSeries1()
        {
            _lineSeries1.MarkerType = MarkerType.Circle;
            _lineSeries1.StrokeThickness = 2;
            _lineSeries1.MarkerSize = 3;
            _lineSeries1.Title = "Start";
            _lineSeries1.MouseDown += (s, e) =>
            {
                if (e.ChangedButton == OxyMouseButton.Left)
                {
                    PlotModel.Subtitle = "Index of nearest point in LineSeries: " +
Math.Round(e.HitTestResult.Index);
                    PlotModel.InvalidatePlot(false);
                }
            };
            PlotModel.Series.Add(_lineSeries1);
        }
```

مقدار خاصیت MarkerType، نحوه‌ی نمایش نقاط اضافه شده را مشخص می‌کند. خاصیت Title، عنوان آن‌را که در کنار صفحه نمایش داده شده، تعیین کرده و در آخر، این سری نیز باید به سری‌های PlotModel اضافه گردد.
هر سری دارای خاصیت MouseDown نیز هست. برای مثال اگر علاقمندید که کلیک کاربر بر روی نقاط مختلف را دریافت کرده و سپس بر این اساس، اطلاعات خاصی را نمایش دهید، می‌توانید از مقدار e.HitTestResult.Index استفاده کنید. در اینجا ایندکس نزدیک‌ترین نقطه به محل کلیک کاربر یافت می‌شود.

تعاریف اولیه سری دوم نیز به همین ترتیب هستند:

```
readonly LineSeries _lineSeries2 = new LineSeries();
        private void addLineSeries2()
        {
            _lineSeries2.MarkerType = MarkerType.Circle;
            _lineSeries2.Title = "End";
            _lineSeries2.StrokeThickness = 2;
            _lineSeries2.MarkerSize = 3;
            _lineSeries2.MouseDown += (s, e) =>
            {
                if (e.ChangedButton == OxyMouseButton.Left)
                {
                    PlotModel.Subtitle = "Index of nearest point in LineSeries: " +
Math.Round(e.HitTestResult.Index);
                    PlotModel.InvalidatePlot(false);
                }
            };
            PlotModel.Series.Add(_lineSeries2);
        }
```

به روز رسانی دستی OxyPlot

پس از نمایش اولیه OxyPlot، هر تغییری که در اطلاعات آن صورت گیرد، نمایش داده نخواهد شد. برای به روز رسانی آن فقط کافی است متد PlotModel.InvalidatePlot را فراخوانی نمائید. برای نمونه در متدهای فوق، کلیک ماوس، پس از رسم نمودار انجام می‌شود. بنابراین اگر نیاز است زیرعنوان نمودار تغییر کند، باید متد PlotModel.InvalidatePlot نیز فراخوانی گردد.

ایجاد یک تایمر برای افزودن نقاط به صورت پویا

در ادامه می‌خواهیم نقاطی را به صورت پویا به نمودار اضافه کنیم. نمایش یکباره نمودار، نکته‌ی خاصی ندارد. تنها کافی است توسط lineSeries1.Points.Add یک سری DataPoint را اضافه کنید. این نقاط در زمان نمایش View، به یکباره نمایش داده خواهند شد. اما در اینجا ابتدا یک چارت خالی نمایش داده می‌شود و سپس قرار است نقاطی به آن اضافه شوند.

```
private int _xMax;
        private int _yMax;
        private bool _haveNewPoints;
        private void addPoints()
        {
            var timer = new DispatcherTimer {Interval = TimeSpan.FromSeconds(1)};
            var rnd = new Random();
            var x = 1;
            updateXMax(x);
            timer.Tick += (sender, args) =>
            {
                var y1 = rnd.Next(100);
                updateYMax(y1);
                _lineSeries1.Points.Add(new DataPoint(x, y1));

                var y2 = rnd.Next(100);
                updateYMax(y2);
                _lineSeries2.Points.Add(new DataPoint(x, rnd.Next(y2)));

                x++;

                updateXMax(x);
                _haveNewPoints = true;
            };
            timer.Start();
        }

        private void updateXMax(int value)
        {
            if (value > _xMax)
            {
                _xMax = value;
            }
        }

        private void updateYMax(int value)
        {
            if (value > _yMax)
            {
                _yMax = value;
            }
        }
```

چند نکته در اینجا حائز اهمیت هستند:

- افزودن نقاط جدید توسط متدهای lineSeries1.Points.Add انجام می‌شوند.

- مقادیر max محورهای x و y را نیز ذخیره می‌کنیم. اگر نقاط برنامه پویا نباشند، OxyPlot به صورت خودکار نمودار را با مقیاس درستی ترسیم می‌کند. اما اگر نقاط پویا باشند، نیاز است حداکثر محورهای x و y را به صورت دستی در آن تنظیم کنیم. به همین جهت متدهای updateXMax و updateYMax در اینجا فراخوانی شده‌اند.

- به روز رسانی ظاهر چارت، توسط متد زیر انجام می‌شود:

```
private readonly Stopwatch _stopwatch = new Stopwatch();
        private void updatePlot()
```

```
        {
            CompositionTarget.Rendering += (sender, args) =>
            {
                if (_stopwatch.ElapsedMilliseconds > _lastUpdateMilliseconds + 2000 && _haveNewPoints)
                {
                    if (_yMax > 0 && _xMax > 0)
                    {
                        _yAxis.Maximum = _yMax + 3;
                        _xAxis.Maximum = _xMax + 1;
                    }

                    PlotModel.InvalidatePlot(false);

                    _haveNewPoints = false;
                    _lastUpdateMilliseconds = _stopwatch.ElapsedMilliseconds;
                }
            };
        }
```

کل کاری که در اینجا انجام شده، فراخوانی کنترل شده‌ی PlotModel.InvalidatePlot هر دو ثانیه یکبار است.
CompositionTarget.Rendering بر اساس رندر View، عمل کرده و از آن می‌توان برای به روز رسانی نمایشی چارت استفاده کرد.
اگر متد PlotModel.InvalidatePlot را دقیقا در زمان افزودن نقاط فراخوانی کنیم به CPU Usage بالایی خواهیم رسید. به همین جهت نیاز است فراخوانی آن کنترل شده و در فواصل زمانی مشخصی باشد. همچنین اگر نقطه‌ای اضافه نشده (بر اساس مقدار haveNewPoints)، به روز رسانی انجام نخواهد شد.
نکته‌ی دیگری که در متد updatePlot فوق درنظر گرفته شده‌است، تغییر مقدار Maximum محورهای x و y بر اساس حداکثرهای نقاط اضافه شده‌است. به این ترتیب نمودار به صورت خودکار جهت نمایش کل اطلاعات، تغییر اندازه خواهد داد.
البته همانطور که عنوان شد، تمام این تهمیدات برای نمایش نمودارهای بلادرنگ است. اگر کار مقدار دهی Points.Add را فقط یکبار در سازنده‌ی ViewModel انجام می‌دهید، نیازی به این نکات نخواهید داشت.

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید :

OxyPlotWpfTests.zip

نظرات خوانندگان

نویسنده: رامین علیرضایی
تاریخ: ۱۳۹۳/۰۷/۲۳ ۱۹:۱۱

سلام و تشکر فراوان بابت معرفی این کتابخانه.
با توجه به استفاده این کتابخانه از PdfSharp آیا خروجی PDF آن با زبان فارسی مشکلی نخواهد داشت؟
با سپاس

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۲۳ ۱۹:۴۰

- هسته‌ی آن هیچ وابستگی به کتابخانه‌ی خاصی ندارد.
- PdfSharp از زبان‌های راست به چپ پشتیبانی نمی‌کند. اما ... یک مثال کامل export در اینجا دارد. خروجی تصویر، Svg، Xaml و XPS و امثال آن، مشکلی با زبان فارسی ندارند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۲۸ ۱۴:۲۰

یک نکته‌ی تکمیلی

نمایش tracker آن با حرکت ماوس، بجای کلیک بر روی نقاط (حالت پیش فرض)

```
private IPlotController _controller;
        public IPlotController Controller
        {
            get
            {
                if (_controller == null)
                {
                    // show tracker with mouse move
                    _controller = new PlotController();
                    _controller.BindMouseEnter(PlotCommands.HoverPointsOnlyTrack);
                }
                return _controller;
            }
        }
```

و بعد

```
Controller="{Binding Controller}"
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۰۴ ۱۴:۳۱

مثال این نکته را به همراه نمایش اطلاعات اضافی در tracker آن، از اینجا می‌توانید دریافت کنید: OxyPlotWpfTests2.zip

عنوان: **معرفی DNTProfiler**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۲/۲۹ ۱۶:۵۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, MVVM, NHibernate, WPF, ASP.NET Web API, DNTProfiler

پیشاپیش فرا رسیدن سال نو را به تمام همراهان گرامی سایت net tips. تبریک عرض می‌کنم. به امید سالی پر از سلامتی و رونق، به همراه اشتیاق روز افزون جستجوگری و کشف زوایای پنهان دنیای برنامه نویسی!
هدیه‌ی نوروزی سایت net tips. پروژه‌ی پروفایلر سورس بازی است که با EF 6.x و همچنین NHibernate 4.x سازگار است. این پروژه از دو قسمت کلاینت و سرور تشکیل می‌شود.

نصب کلاینت EF برنامه‌ی DNTProfiler

تفاوتی نمی‌کند که برنامه‌ی شما وبی است یا ویندوزی؛ برای هر دو حالت ابتدا دستور ذیل را در کنسول پاورشل نیوگت اجرا کنید:

```
PM> Install-Package DNTProfiler.EntityFramework.Core
```

با اینکار interceptorهای پروفایلر نصب می‌شوند. برای فعال سازی آن نیاز است چند سطر ذیل را به فایل app.config/web.config پروژه‌ی خود اضافه کنید:

```
<configuration>
  <entityFramework>
    <interceptors>
      <interceptor type="DNTProfiler.EntityFramework.Core.DatabaseLogger,
DNTProfiler.EntityFramework.Core">
        <parameters>
          <parameter value="http://localhost:8080" />
          <parameter value="|DataDirectory|\ErrorsLog.Log" />
        </parameters>
      </interceptor>
    </interceptors>
  </entityFramework>
</configuration>
```

دریافت و راه اندازی برنامه‌ی DNTProfiler

آخرین نگارش برنامه‌ی DNTProfiler را از برگه‌ی releases مخزن کد آن می‌توانید دریافت کنید:
https://github.com/VahidN/DNTProfiler/releases

این برنامه برای دات نت 4 نوشته شده‌است. بنابراین اگر هنوز از ویندوز XP استفاده می‌کنید، امکان کار کردن با آن‌را خواهید داشت.
البته بسته‌ی نیوگت DNTProfiler.EntityFramework.Core آن برای دات نت 4 و 4.5 تهیه شده‌است و به صورت خودکار بر اساس ساختار پروژه‌ی شما، یکی از آن‌ها نصب خواهد شد.

تا اینجا کار راه اندازی این برنامه به پایان می‌رسد. برای استفاده‌ی از آن باید ابتدا برنامه‌ی DNTProfiler را اجرا کنید. این برنامه به پیام‌های رسیده‌ی از برنامه‌ی اصلی شما (ارسال شده توسط DNTProfiler.EntityFramework.Core) گوش فرا می‌دهد و سپس شروع به آنالیز آن‌ها خواهد کرد. ساختار این تبادل اطلاعات هم بر اساس تهیه‌ی یک ASP.NET Self host Web API است.

این برنامه به صورت ماژولار تهیه شده‌است و تمام آنالیز کننده‌های آن در حقیقت یک پلاگین هستند (در حال حاضر دارای 32 پلاگین است). این پلاگین‌ها در گروه‌های Alerts (برای مثال یافتن جوین‌های تکراری و یک سری موارد بهینه سازی سرعت)، Loggers (طبقه بندی خام اطلاعات رسیده)، Visualizers (نمایش بصری اطلاعات رسیده) قرار می‌گیرند.

نظرات، پیشنهادات و همکاری

لطفا برای طرح سؤالات و ارائه‌ی پیشنهادات خود در زمینه‌ی این پروژه، به قسمت اختصاصی آن در سایت مراجعه نمائید:
http://www.dotnettips.info/projects/details/21

نظرات خوانندگان

نویسنده: سیروان عفیفی
تاریخ: ۱۳۹۳/۱۲/۲۹ ۱۷:۲۸

بنده هم پیشاپیش سال نو رو خدمت شما و اعضای محترم سایت تبریک عرض میکنم، امیدوارم سال خوب و پر از موفقیت‌های روزافزون داشته باشید. بابت پروژه هم خیلی خیلی ممنون واقعاً عالیه!

نویسنده: امین کاشانی
تاریخ: ۱۳۹۴/۰۱/۰۱ ۲:۵۱

باسلام و تبریک عید نوروز
به نظر میرسه در ویندوز 8.1 فایل اجرایی اجرا نمیشه و فرم را نشان نمی‌دهد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۱/۰۱ ۱۰:۱

- تصویر انتهای بحث در ویندوز 8.1 تهیه شده‌است.
- این برنامه برای اجرا نیاز به دات نت فریم ورک 4 دارد. بنابراین بر روی ویندوزهای XP SP3 به بعد قابل اجرا است.
- ضمنا خطاهای برنامه در فایل متنی به نام **ErrorsLog.Log** در کنار فایل اجرایی برنامه ثبت می‌شوند.

نویسنده: میثم ثوامری
تاریخ: ۱۳۹۴/۰۱/۰۲ ۳:۲۲

با سلام و تبریک سال نو
نرمافزار تو ویندوز 8.1 کارنمیده
فایل **ErrorsLog** ظاهرا ایجاد نشد
windows 8.1 64bit,vs2013sp4

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۱/۰۲ ۱۳:۷

- لطفا نگارش جدید را دریافت کنید (^). فایل System.Net.Http.Formatting.dll فراموش شده بود.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۱/۰۲ ۱۳:۷

- لطفا نگارش جدید را دریافت کنید (^). فایل System.Net.Http.Formatting.dll فراموش شده بود.

نویسنده: امین کاشانی
تاریخ: ۱۳۹۴/۰۱/۰۲ ۲۲:۱۵

البته در ورژن قبلی بااضافه کردن خط زیر فایل کانفیگ و اجرای برنامه به صورت Run As Administrator مشکل حل شد.

```
<runtime>
<loadFromRemoteSources enabled="true"/>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="System.Net.Http" publicKeyToken="b03f5f7f11d50a3a" culture="neutral" />
        <bindingRedirect oldVersion="0.0.0.0-2.2.29.0" newVersion="2.2.29.0" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="Newtonsoft.Json" publicKeyToken="30ad4fe6b2a6aeed" culture="neutral" />
        <bindingRedirect oldVersion="0.0.0.0-6.0.0.0" newVersion="6.0.0.0" />
      </dependentAssembly>
```

```
    </assemblyBinding>
  </runtime>
```

نویسنده: امین کاشانی
تاریخ: ۱۳۹۴/۰۱/۰۲ ۲۲:۲۶

در نسخه جدید نیز در حالت please wait loading فرم قرار میگیرد.
که با اضافه کردن خط زیر در فایل کانفیگ مشکل حل میشود

```
<loadFromRemoteSources enabled="true"/>
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۱/۰۲ ۲۳:۵۱

البته loadFromRemoteSources مربوط است به بارگذاری اسمبلی‌ها برای مثال از یک درایو به اشتراک گذاشته شده‌ی در شبکه. فایل‌های کپی شده‌ی از اینترنت هم چنین نیازی را دارند. به این معنا که attribute فایل‌ها احتمالا توسط مرورگر (اگر از download manager استفاده نکرده‌اید)، تغییر کرده‌اند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۱/۰۳ ۰:۲۷

- برنامه نیازی به run as admin ندارد. فقط در حالت فعال سازی allow remote connections آن (قرار دادن برنامه در یک سیستم دیگر و اتصال از راه دور به آن) نیاز به run as admin هست.
- اگر نیاز به run as admin بوده یعنی attribute فایل‌ها احتمالا توسط مرورگر (اگر از download manager استفاده نکرده‌اید)، تغییر کرده‌اند.

نویسنده: ژوپیتر
تاریخ: ۱۳۹۴/۰۱/۰۷ ۱۱:۴۲

با سلام و عرض تبریک به مناسبت سال جدید و همچنین تشکر ویژه برای ارائه این Profiler عالی که برای استفاده از نوع غیربومی همیشه دچار مشکل بودیم. واقعا عیدی خوبی بود. با آرزوی موفقیت برای همه‌ی دوستان در سال جدید.

نویسنده: محمد عیدی مراد
تاریخ: ۱۳۹۴/۰۱/۰۸ ۱۳:۱۰

سلام و عرض تبریک سال نو
بعد از راه اندازی کلاینت زمانی که اپلیکیشن رو اجرا میکنم با ارور زیر مواجه میشم:

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۱/۰۸ ۱۳:۱۷

- لطفا برای طرح سؤالات و ارائه‌ی پیشنهادات خود در زمینه‌ی این پروژه، به قسمت اختصاصی آن در سایت مراجعه نمائید: http://www.dotnettips.info/projects/details/21

- استثناءها را با تصویر ارائه ندهید. اصل متنی استثناء، بهتر قابلیت پیگیری دارد.

خطاهای برنامه در فایل متنی به نام **ErrorsLog.Log** در کنار فایل اجرایی برنامه ثبت می‌شوند.

- سورس را اجرا کردید یا اصل برنامه‌ی توزیع شده را؟ به نظر سورس را اجرا کرده‌اید. آیا بسته‌های نیوگت آن‌را به درستی بازیابی کرده‌اید ؟ آیا از فایل Microsoft.SqlServer.TransactSql.ScriptDom با شماره نگارش صحیحی استفاده می‌کنید که پیام داده‌است متدهای آن یافت نشدند؟

```
<package id="Microsoft.SqlServer.TransactSql.ScriptDom" version="12.0.1" targetFramework="net40" />
```

این فایل‌ها به همراه بسته‌ی توزیع شده وجود دارند.

نویسنده: محمدعلی ک
تاریخ: ۱۳۹۴/۰۱/۲۹ ۸:۵۹

سلام؛ در لایه وب این پروژه ، با استفاده از کنسول نیوگت،بسته زیر رو نصب کردم : DNTProfiler.EntityFramework.Core
سپس تنظیمات مورد نظر در فایل web.config رو اعمال کردم که البته مشکلی در اجرای پروژه به وجود نمیاد ولی با نگه داشتن ماوس روی تگ <interceptors> ، با پیغام زیر مواجه میشم :
the element entityframework has invalid chid element iterceptors
وقتی برنامه DNT Profiler رو اجرا میکنم متاسفانه هیچ اطلاعاتی در مورد پروژه ام بهم نشون نمیده. حتی با تعویض پورت هم مسئله حل نشد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۱/۲۹ ۹:۵۳

- برنامه‌ی DNT Profiler باید پیش از اجرای برنامه‌ی وب شما اجرا شود و نه پس از آن.

+ خطاهای DNTProfiler.EntityFramework.Core داخل فایل **App_Data \ErrorsLog.Log** ثبت می‌شوند. این فایل را بررسی کنید.

نویسنده: م ابوعلی
تاریخ: ۱۳۹۴/۰۲/۰۸ ۰:۵

با سلام و تشکر به خاطر این نرم افزار مفید.
نرم افزار و پکت‌های اون برای EntityFramework در دو حالت وب و ویندوز درست عمل می‌کند ولی برای NHibernate فقط در حالت ویندوزی درست عمل میکند و در حالت وبی، موقع فراخوانی BuildSessionFactory به اکسپشن زیر برخورد میکنم:

```
Could not create the driver from
DNTProfiler.NHibernate.Core.Drivers.ProfiledSql2008ClientDriver,DNTProfiler.NHibernate.Core
```

و بعد از لود صفحه‌ی وب اکسپشن زیر نمایش داده می‌شود:

```
exePath must be specified when not running inside a stand alone exe
```

با سرچی که انجام دادم به نظر می‌رسد به پکت نیوگت NHibenate این نرم افزار و نحوه‌ی کار با Configuration manager در کانتکست وبی و ویندوزی مربوط باشد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۲/۰۸ ۱:۳۳

ممنون. این مشکل برطرف شد و با نگارش بعدی ارائه می‌شود.

نویسنده: م ابوعلی
تاریخ: ۱۳۹۴/۰۲/۰۸ ۲۲:۶

کد برنامه را دریافت کردم و به روز رسانی‌های لازم را انجام دادم. این خطا برطرف شده است. خیلی ممنون.

آیا این پروفایلر امکان دریافت درخواست هایی که از طریق سرویس‌های wcf به پایگاه داده ارسال میشوند را هم دارد؟ یا می‌تواند داشته باشد؟ (البته این سوال صرفاً جنبه‌ی تئوریک دارد.)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۲/۰۸ ۲۳:۹

در اینجا فناوری مورد استفاده مهم نیست. همینقدر که از EF یا NH استفاده شده باشد، کار می‌کند.

عنوان: **چگونه برنامه‌های دات نت را خارج از ویژوال استودیو دیباگ کنیم؟**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۱/۰۲ ۱۵:۰
آدرس: www.dotnettips.info
گروه‌ها: Debugging, WPF, exception handling, windbg

مشکل: نگارش 1.0.808.0 برنامه‌ی DNTProfiler بر روی سایر سیستم‌ها، هنوز به مرحله‌ی نمایش نرسیده، کرش می‌کند. علت چیست؟

این نگارش بر روی سیستم من مشکلی نداشت ولی پس از چند گزارش عدم امکان اجرای آن بر روی سایر سیستم‌ها، یک ماشین مجازی ویندوز 8.1 را تهیه و برنامه را بر روی آن اجرا کردم. بله ... برنامه هنوز به مرحله‌ی نمایش نرسیده، محو می‌شد. در این مرحله‌ی ابتدایی امکان تهیه‌ی لاگ استثنای حاصل توسط برنامه وجود نداشت و تنها این خطا در event viewer ویندوز (Computer management -> event viewer -> windows logs -> application) قابل مشاهده بود:

```
Application: DNTProfiler.exe
Framework Version: v4.0.30319
Description: The process was terminated due to an unhandled exception.
Exception Info: System.Windows.Markup.XamlParseException
Stack:
at System.Windows.Markup.WpfXamlLoader.Load
```

متاسفانه از این استثنای لاگ شده‌ی در لاگ‌های ویندوز، اطلاعات مفیدی را نمی‌توان دریافت کرد. عنوان کرده XamlParse برنامه را نمی‌تواند آغاز کند.
روش حل این نوع مشکلات و بررسی آن‌ها در خارج از VS.NET، با استفاده از برنامه‌ی معروف WinDBG مایکروسافت میسر است.
دو نگارش 32 بیتی و 64 بیتی آن‌را از اینجا می‌توانید دریافت کنید:
http://codemachine.com/downloads.html

پس از دریافت، بهتر است نسخه‌ی 32 بیتی آن‌را اجرا کنید، زیرا برای دیباگ برنامه‌های دات نت این نسخه است که امکان بارگذاری فایل C:\Windows\Microsoft.NET\Framework\v4.0.30319\sos.dll را دارد.

تا اینجا فرض بر این است که نسخه‌ی 32 بیتی windbg را دریافت و اجرا کرده‌اید. در ادامه از منوی File آن گزینه‌ی Open executable را انتخاب کرده و فایل exe برنامه را به آن معرفی کنید.

```
CommandLine: C:\Tests\DNTProfiler.1.0.808.0\DNTProfiler.exe
Symbol search path is: srv*
Executable search path is:
ModLoad: 00150000 00176000 DNTProfiler.exe
ModLoad: 76f30000 77098000 ntdll.dll
ModLoad: 72ee0000 72f36000 C:\Windows\SysWOW64\MSCOREE.DLL
ModLoad: 74be0000 74d20000 C:\Windows\SysWOW64\KERNEL32.dll
ModLoad: 756e0000 757af000 C:\Windows\SysWOW64\KERNELBASE.dll
(684.764): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=00000000 ecx=37af0000 edx=00000000 esi=7eb8f000 edi=00000000
eip=76fe2d15 esp=002ff604 ebp=002ff630 iopl=0 nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b efl=00000246
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for ntdll.dll -
ntdll!LdrResolveDelayLoadsFromDll+0xa89:
76fe2d15 cc    int    3
```

برنامه داخل این دیباگر اجرا شده و در همینجا نیز به پایان می‌رسد. در ادامه‌ی کار در textbox ذیل این گزارش، دستور g را صادر کرده و enter کنید.

```
0:000> g
ModLoad: 757b0000 75827000 C:\Windows\SysWOW64\ADVAPI32.dll
ModLoad: 05080000 05102000 Newtonsoft.Json.dll
(684.764): C++ EH exception - code e06d7363 (first chance)
(684.764): C++ EH exception - code e06d7363 (first chance)
(684.764): C++ EH exception - code e06d7363 (first chance)
(684.764): CLR exception - code e0434352 (first chance)
First chance exceptions are reported before any exception handling.
```

```
This exception may be expected and handled.
eax=002fd0a0 ebx=00000005 ecx=00000005 edx=00000000 esi=002fd164 edi=00000001
eip=756f3d67 esp=002fd0a0 ebp=002fd0f8 iopl=0 nv up ei pl nz ac pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b efl=00200216
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for
C:\Windows\SysWOW64\KERNELBASE.dll -
KERNELBASE!RaiseException+0x48:
756f3d67 8b4c2454   mov   ecx,dword ptr [esp+54h] ss:002b:002fd0f4=3fce1188
```

فرمان g سبب می‌شود تا کار دیباگ برنامه ادامه پیدا کند و به اولین CLR exception رسیده و متوقف شود. در ادامه نیاز است تا با صدور فرمان sxe clr، دیباگ clr را فعال کرده و سپس مجددا دستور g را برای ادامه‌ی دیباگ صادر کنیم.

```
0:000> sxe clr
0:000> g
(684.764): C++ EH exception - code e06d7363 (first chance)
(684.764): CLR exception - code e0434352 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=002fe2a0 ebx=00000005 ecx=00000005 edx=00000000 esi=002fe368 edi=00000001
eip=756f3d67 esp=002fe2a0 ebp=002fe2fc iopl=0 nv up ei pl nz ac pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b efl=00200216
KERNELBASE!RaiseException+0x48:
756f3d67 8b4c2454   mov   ecx,dword ptr [esp+54h] ss:002b:002fe2f4=3fce2388
```

بنابراین ابتدا دستور sxe clr را صادر و enter و سپس g و enter. مجددا برنامه پس از رسیدن به یک CLR exception متوقف می‌شود.

اکنون می‌خواهیم پیام استثنای واقعی دات نت را مشاهده کنیم. به همین جهت ابتدا دستور loadby sos clr! و سپس دستور CLRStack! را صادر می‌کنیم:

```
0:000> !loadby sos clr
0:000> !CLRStack
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for
C:\Windows\Microsoft.NET\Framework\v4.0.30319\clr.dll -
PDB symbol for clr.dll not loaded
OS Thread Id: 0x764 (0)
Child SP IP Call Site
002fe3f4 756f3d67 [GCFrame: 002fe3f4]
002fe430 756f3d67 [GCFrame: 002fe430]
002fe528 756f3d67 [GCFrame: 002fe528]
002fe544 756f3d67 [HelperMethodFrame_2OBJ: 002fe544]
System.RuntimeTypeHandle.CreateInstance(System.RuntimeType, Boolean, Boolean, Boolean ByRef,
System.RuntimeMethodHandleInternal ByRef, Boolean ByRef)
DNTProfiler.App.Main()
002ff190 720e2652 [GCFrame: 002ff190]
```

در اینجا کار بررسی stack trace برنامه پایان می‌یابد. اکنون با صدور دستور PrintException! می‌توان علت اصلی استثناء را مشاهده کرد:

```
0:000> !PrintException
Exception object: 021e6f0c
Exception type: System.Reflection.TargetInvocationException
Message: Exception has been thrown by the target of an invocation.
InnerException: System.IO.FileNotFoundException, Use !PrintException 021e6950 to see more.
StackTrace (generated):
<none>
StackTraceString: <none>
HResult: 80131604
0:000> !PrintException 021e6950
Exception object: 021e6950
Exception type: System.IO.FileNotFoundException
Message: Could not load file or assembly 'System.Net.Http.Formatting, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35' or one of its dependencies. The system cannot find the file specified.
InnerException: <none>
StackTrace (generated):
SP IP Function
00000000 00000001 DNTProfiler!DNTProfiler.Services.SelfHostConfig.getHostConfiguration(System.String,
Boolean)+0x2
002FE298 006B4E54 DNTProfiler!DNTProfiler.Services.SelfHostConfig.OpenWait(System.String, Boolean)+0x14
002FE2B0 006B4A3F DNTProfiler!DNTProfiler.ViewModels.MainWindowViewModel.doStart(System.String)+0x1f
```

```
002FE2C0 006B368D
DNTProfiler!DNTProfiler.ViewModels.MainWindowViewModel..ctor(DNTProfiler.Common.Mvvm.ICommonDialogsServ
ice, DNTProfiler.Common.Controls.DialogManagement.Contracts.IDialogManager)+0xcd
002FE2D8 006B04C9 DNTProfiler!DNTProfiler.MainWindow..ctor()+0x91

StackTraceString: <none>
HResult: 80070002
```

همانطور که مشاهده می‌کنید، عنوان کرده‌است که فایل System.Net.Http.Formatting قابل بارگذاری نیست. بله ... این فایل جزو بسته‌ی نگارش اول برنامه نبود و به نظر علت عدم کرش آن، نصب این فایل در GAC سیستم توسط نصاب‌های ASP.NET MVC بوده‌است (نگارش‌های 3 و 4 آن). بنابراین اگر کسی این فایل را در GAC سیستم خود نداشته باشد، قادر به اجرای برنامه نخواهد بود.

روش حل این مشکل، مراجعه به خواص پروژه، یافتن اسمبلی System.Net.Http.Formatting در لیست ارجاعات برنامه و سپس true کردن خاصیت copy to local آن است. به این ترتیب این اسمبلی در کنار فایل اجرایی برنامه کپی خواهد شد.

بنابراین خلاصه‌ی عملیات یافتن علت اصلی کرش برنامه در خارج از ویژوال استودیو به صورت ذیل است:

الف) اجرای نسخه‌ی 32 بیتی برنامه‌ی windbg

ب) انتخاب منوی File آن و سپس باز کردن فایل اجرایی برنامه توسط گزینه‌ی Open executable

ج) اجرای دستورات ذیل به ترتیب:

```
0:000> g
0:000> sxe clr
0:000> g
0:000> !loadby sos clr
0:000> !CLRStack
0:000> !PrintException
```

چند نمونه‌ی مشابه برای مطالعه‌ی بیشتر

Finding CLR exceptions without visual studio

Power of WinDBG: The story of mysterious crash of WPF application

عنوان: **اسکرول روان لیست‌های مجازی سازی شده در WPF 4.5**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۱/۱۶ ۱۱:۴۵
آدرس: www.dotnettips.info
گروه‌ها: WPF, Performance

جهت « بهبود کارآیی کنترل‌های لیستی WPF در حین بارگذاری تعداد زیادی از رکوردها » توصیه شده‌است که مجازی سازی UI فعال گردد. به این ترتیب بجای تولید یکباره‌ی برای مثال 1000 ردیف، تنها 10 ردیفی که نمایان هستند تولید می‌شوند. بنابراین مصرف حافظه و سرعت برنامه به نحو قابل ملاحظه‌ای افزایش خواهد یافت. اما ... این مجازی سازی، اسکرول مطلوبی ندارد و بریده بریده به نظر می‌رسد.

خاصیت‌های جدید VirtualizingPanel در دات نت 4.5

تمام کنترل‌های مشتق شده‌ی از ListBox مانند ListView و امثال آن، در WPF 4.5 امکان تنظیم یک چنین خواصی را یافته‌اند:

```
<ListBox ItemsSource="{Binding Persons}"
        VirtualizingPanel.IsVirtualizing="True"
        VirtualizingPanel.ScrollUnit="Pixel"
        VirtualizingPanel.IsVirtualizingWhenGrouping="True"
        VirtualizingPanel.CacheLength="100"
        VirtualizingPanel.CacheLengthUnit="Pixel"/>
```

در WPF 4.5 پس از نمایش تعداد ردیف‌های قابل ملاحظه‌ی توسط کاربر، سیستم کش خودکاری با حق تقدم پایین فعال شده و آیتم‌هایی را که هنوز نمایان نیستند، ایجاد و کش می‌کند. به این ترتیب کاربر با اسکرول به سمت پایین یا بالا، کندی رندر یا بریده بریده به نظر رسیدن اسکرول را حس نخواهد کرد.
در اینجا می‌توان دو خاصیت CacheLength و CacheLengthUnit را مقدار دهی کرد. مقدار پیش فرض CacheLength مساوی 1.1 است. CacheLengthUnit می‌تواند یکی از مقادیر Pixel, Item یا Page را بپذیرد. هر Page در اینجا بر اساس اندازه‌ی viewport یا قسمت نمایان لیست، تعریف می‌شود. مقدار پیش فرض آن نیز Item است.
همچنین در اینجا می‌توان ScrollUnit را نیز تنظیم کرد که مقادیر Item یا Pixel را می‌پذیرد. حالت پیش فرض آن Item است؛ به این معنا که اگر ردیفی کاملا در viewport جای نگرفت، نمایش داده نمی‌شود. این مورد را با تنظیم مقدار ScrollUnit به Pixel می‌توان بهبود بخشید.
IsVirtualizingWhenGrouping سبب فعال سازی مجازی سازی حتی در حالت گروه بندی اطلاعات می‌گردد. به صورت پیش فرض اگر گروه بندی فعال شود، دیگر مجازی سازی رخ نخواهد داد.

فعال سازی قابلیت‌های دات نت 4.5 در برنامه‌های WPF 4

اگر برنامه‌ی WPF 4 خود را فعلا قصد ندارید به دات نت 4.5 ارتقاء دهید، با توجه به اینکه اگر کاربر دات نت 4.5 را نصب کرده باشد، برنامه‌ی شما به صورت خودکار همانند یک برنامه‌ی WPF 4.5 رفتار می‌کند (دات نت 4.5 جایگزین دات نت 4 می‌شود)، می‌توان با اندکی Reflection این قابلیت‌ها را در صورت وجود، فعال کرد:

```
using System;
using System.Reflection;
using System.Windows;
using System.Windows.Controls;

namespace DNTProfiler.Common.Behaviors
{
    /// <summary>
    /// Smooth scrolling VirtualizingStackPanels, without sacrificing virtualization.
    /// </summary>
    public static class PixelBasedScrollingBehavior
    {
        private static readonly MethodInfo _setScrollUnit = typeof(VirtualizingPanel)
            .GetMethod("SetScrollUnit", BindingFlags.Public | BindingFlags.Static);

        private static readonly MethodInfo _setCacheLengthUnit = typeof(VirtualizingPanel)
            .GetMethod("SetCacheLengthUnit", BindingFlags.Public | BindingFlags.Static);
```

```
        private static readonly MethodInfo _setCacheLength = typeof(VirtualizingPanel)
            .GetMethod("SetCacheLength", BindingFlags.Public | BindingFlags.Static);

        public static bool GetIsEnabled(DependencyObject obj)
        {
            return (bool)obj.GetValue(IsEnabledProperty);
        }

        public static void SetIsEnabled(DependencyObject obj, bool value)
        {
            obj.SetValue(IsEnabledProperty, value);
        }

        public static readonly DependencyProperty IsEnabledProperty =
            DependencyProperty.RegisterAttached("IsEnabled", typeof(bool),
typeof(PixelBasedScrollingBehavior), new UIPropertyMetadata(false, handleIsEnabledChanged));

        private static void handleIsEnabledChanged(DependencyObject obj,
DependencyPropertyChangedEventArgs e)
        {
            var listView = obj as ListView;
            if (listView == null)
            {
                throw new InvalidOperationException("This behavior can only be attached to a
ListView.");
            }

            if (_setScrollUnit != null)
            {
                // It's .NET 4.5
                _setScrollUnit.Invoke(listView, new object[] { listView, /*Pixel*/ 0 });
            }

            if (_setCacheLengthUnit != null)
            {
                // It's .NET 4.5
                _setCacheLengthUnit.Invoke(listView, new object[] { listView, /*Pixel*/ 0 });
            }


            if (_setCacheLength != null)
            {
                // It's .NET 4.5
                var type = Type.GetType("System.Windows.Controls.VirtualizationCacheLength,
PresentationFramework, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35");
                if (type == null) return;
                var instance = Activator.CreateInstance(type, 100.0);

                _setCacheLength.Invoke(listView, new[] { listView, instance });
            }
        }
    }
}
```

در اینجا با تعریف یک Attached property جدید، خواصی مانند ScrollUnit، CacheLengthUnit و CacheLength توسط Reflection یافت خواهند شد. اگر مقدار آن‌ها نال نباشند، یعنی برنامه‌ی دات نت 4 در سیستمی که بر روی آن دات نت 4.5 نصب است، در حال اجرا می‌باشد. بنابراین در این حالت می‌توان اسکروال مبتنی بر Pixel را فعال کرد تا برنامه اسکرول روان‌تری را پیدا کند.

برای استفاده از آن خواهیم داشت:

```
<ListView ItemsSource="{Binding}"
               behaviors:PixelBasedScrollingBehavior.IsEnabled="True">
```

عنوان: **پیاده سازی INotifyPropertyChanged با استفاده از Unity Container**
نویسنده: برات جوادی
تاریخ: ۱۳۹۴/۰۱/۲۴ ۱۴:۲۵
آدرس: www.dotnettips.info
گروه‌ها: WPF, UnityContainer, AOP

AOP چیست

AOP یکی از فناوری‌های مرتبط با توسعه نرم افزار محسوب می‌شود که توسط آن می‌توان اعمال مشترک و متداول موجود در برنامه را در یک یا چند ماژول مختلف قرار داد (که به آن‌ها Aspects نیز گفته می‌شود) و سپس آن‌ها را به مکان‌های مختلفی در برنامه متصل ساخت. عموما Aspects، قابلیت‌هایی را که قسمت عمده‌ای از برنامه را تحت پوشش قرار می‌دهند، کپسوله می‌کنند. اصطلاحا به این نوع قابلیت‌های مشترک، تکراری و پراکنده مورد نیاز در قسمت‌های مختلف برنامه، Cross cutting concerns نیز گفته می‌شود؛ مانند اعمال ثبت وقایع سیستم، امنیت، مدیریت تراکنش‌ها و امثال آن. با قرار دادن این نیازها در Aspects مجزا، می‌توان برنامه‌ای را تشکیل داد که از کدهای تکراری عاری است.

پیاده سازی INotifyPropertyChanged یکی از این مسائل می‌باشد که می‌توان آن را در یک Aspect محصور و در ماژول‌های مختلف استفاده کرد.

مسئله:

کلاس زیر مفروض است:

```
public class Foo
{
        public virtual int Id { get; set; }

        public virtual string Name { get; set; }
}
```

اکنون می‌خواهیم کلاس Foo را به INotifyPropertyChanged مزین، و یک Subscriber به قسمت set پراپرتی‌های کلاس تزریق کنیم.

راه حل:

ابتدا پکیج‌های Unity را از Nuget دریافت کنید:

```
PM> Install-Package Unity.Interception
```

این پکیج وابستگی‌های خود را که Unity و CommonServiceLocator هستند نیز دریافت می‌کند.

حال یک Interceptor که اینترفیس IInterceptionBehavior را پیاده سازی می‌کند، می‌نویسیم:

```
namespace NotifyPropertyChangedInterceptor.Interceptions
{
    using System;
    using System.Collections.Generic;
    using System.ComponentModel;
    using System.Reflection;
    using Microsoft.Practices.Unity.InterceptionExtension;

    class NotifyPropertyChangedBehavior : IInterceptionBehavior
    {
        private event PropertyChangedEventHandler PropertyChanged;

        private readonly MethodInfo _addEventMethodInfo =
            typeof(INotifyPropertyChanged).GetEvent("PropertyChanged").GetAddMethod();

        private readonly MethodInfo _removeEventMethodInfo =
            typeof(INotifyPropertyChanged).GetEvent("PropertyChanged").GetRemoveMethod();


        public IMethodReturn Invoke(IMethodInvocation input, GetNextInterceptionBehaviorDelegate
getNext)
        {
            if (input.MethodBase == _addEventMethodInfo)
            {
                return AddEventSubscription(input);
            }
```

```
            if (input.MethodBase == _removeEventMethodInfo)
            {
                return RemoveEventSubscription(input);
            }

            if (IsPropertySetter(input))
            {
                return InterceptPropertySet(input, getNext);
            }

            return getNext()(input, getNext);
        }

        public bool WillExecute
        {
            get { return true; }
        }

        public IEnumerable<Type> GetRequiredInterfaces()
        {
            yield return typeof(INotifyPropertyChanged);
        }

        private IMethodReturn AddEventSubscription(IMethodInvocation input)
        {
            var subscriber = (PropertyChangedEventHandler)input.Arguments[0];
            PropertyChanged += subscriber;

            return input.CreateMethodReturn(null);
        }

        private IMethodReturn RemoveEventSubscription(IMethodInvocation input)
        {
            var subscriber = (PropertyChangedEventHandler)input.Arguments[0];
            PropertyChanged -= subscriber;

            return input.CreateMethodReturn(null);
        }

        private bool IsPropertySetter(IMethodInvocation input)
        {
            return input.MethodBase.IsSpecialName && input.MethodBase.Name.StartsWith("set_");
        }

        private IMethodReturn InterceptPropertySet(IMethodInvocation input,
GetNextInterceptionBehaviorDelegate getNext)
        {
            var propertyName = input.MethodBase.Name.Substring(4);

            var subscribers = PropertyChanged;
            if (subscribers != null)
            {
                subscribers(input.Target, new PropertyChangedEventArgs(propertyName));
            }

            return getNext()(input, getNext);
        }
    }
}
```

متد Invoke : این متد Behavior مورد نظر را پردازش می‌کند (در اینجا، تزریق یک Subscriber در قسمت set پراپرتی ها). **متد GetRequiredInterfaces** : یک روش است برای یافتن کلاس هایی که با اینترفیس IInterceptionBehavior مزین شده‌اند.
پراپرتی WillExecute : ابن پراپرتی به Unity می‌گوید که این Behavior اعمال شود یا نه. اگر مقدار برگشتی آن false باشد، متد Invoke اجرا نخواهد شد.
همانطور که در متد Invoke مشاهده می‌کنید، شرط هایی برای افزودن و حذف یک Subscriber و چک کردن متد set نوشته شده و در غیر این صورت کنترل به متد بعدی داده می‌شود.

اتصال Interceptor به کلاس ها

در ادامه Unity را برای ساخت یک نمونه از کلاس پیکربندی می‌کنیم:

```
var container = new UnityContainer();

container.RegisterType<Foo, Foo>(
                new AdditionalInterface<INotifyPropertyChanged>(),
                new Interceptor<VirtualMethodInterceptor>(),
                new InterceptionBehavior<NotifyPropertyChangedBehavior>())
                .AddNewExtension<Interception>();
```

توسط متد RegisterType یک Type را با پیکربندی دلخواه به Unity معرفی می‌کنیم. در اینجا به ازای درخواست Foo (اولین پارامتر جنریک)، یک Foo (دومین پارامتر جنریک) برگشت داده می‌شود. این متد تعدادی InjetctionMember (بصورت params) دریافت می‌کند که در این مثال سه InjetctionMember به آن پاس داه شده است:

Interceptor : اطلاعاتی در مورد IInterceptor و نحوه‌ی Intercept یک شیء را نگه داری می‌کند. در اینجا از VirtualMethodInterceptor برای تزریق کد استفاده شده.

InterceptionBehavior : این کلاس Behavior مورد نظر را به کلاس تزریق می‌کند.

AddintionalInterface : کلاس target را مجبور به پیاده سازی اینترفیس دریافتی از پارامتر می‌کند. اگر کلاس behavior، متد GetRequiredInterfaces اینترفیس INotifyPropertyChanged را برمی گرداند، نیازی نیست از AddintionalInterface در پارامتر متد فوق استفاده کنید.

نکته : کلاس VirtualMethodInterceptor فقط اعضای virtual را تحت تاثیر قرار می‌دهد.

اکنون نحوه‌ی ساخت یک نمونه از کلاس Foo به شکل زیر است:

```
var foo = container.Resolve<Foo>();
(foo as INotifyPropertyChanged).PropertyChanged += FooPropertyChanged;
private void FooPropertyChanged (object sender, PropertyChangedEventArgs e)
 {
      // Do some things.......
 }
```

نکته‌ی تکمیلی

طبق مستندات MSDN ، کلاس VirtualMethodInterceptor یک کلاس جدید مشتق شده از کلاس target (در اینجا Foo) می‌سازد. بنابراین اگر کلاس‌های شما دارای Data annotation و یا در کلاس‌های Mapper یک ORM استفاده شده‌اند (مانند کلاس‌های لایه Domain)، بجای VirtualMethodInterceptor از TransparentProxyInterceptor استفاده کنید. سرعت اجرای VirtualMethodInterceptor سریعتر است ؛ اما به یاد داشته که برای استفاده از TransparentProxyInterceptor باید کلاس target از کلاس MarshalByRefObject ارث بری کند. دریافت مثال کامل این مقاله

نظرات خوانندگان

نویسنده: جلال
تاریخ: ۱۳۹۴/۰۱/۲۴ ۲۰:۲۳

این روش به همه‌ی Property Setterهای کلاس بدون در نظر گرفتن نیازهای کاربر/برنامه نویس، فراخوانی PropertyChanged رو اضافه می‌کنه. همینطور ممکنه کاربر بخواد با فراخوانی یه PropertyChanged برای یه Property، بعدش مجددا این رویداد رو برای یه Property دیگه فراخوانی کنه. به نظرم بهتره از روش‌های Attribute Base مثل این استفاده بشه.

نویسنده: برات جوادی
تاریخ: ۱۳۹۴/۰۱/۲۵ ۸:۵۴

- این Interceptor فقط کار تزریق یک Subscriber برای PropertyChanged را به عهده دارد و به سایر نیازها کاری ندارد. ضمن اینکه نیازهای کاربر/برنامه نویس اینجا کمی نامفهوم است!

- هنگام تشخیص متد set در Interceptor میتوان یک شرط دیگر گذاشت و اینکار را انجام داد.
- بسته به سناریو می‌توان از attribute هم استفاده کرد. در اینجا قصدم تزریق برای همه پراپرتی‌ها بوده، درصورتی که تزریق فقط برای برخی از انها باشد، میشه Attribute هم تعریف کرد .

عنوان: **آغاز کار با WPF**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۲/۰۶ ۱۸:۳۵
آدرس: www.dotnettips.info
گروه‌ها: WPF, XAML

من خودم به شخصه هنوز تا به حال با WPF کار نکرده‌ام؛ اما قصد دارم از امروز در هر فرصتی که پیش می‌آید به یادگیری این فناوری پر سر و صدا بپردازم. از آنجا که مجموعه‌ی مرتب و به ترتیبی مثل MVC و EF در این زمینه در سایت موجود نبود، تصمیم گرفتم که خودم استارت این کار را بزنم که باعث میشه هم خودم بهتر یاد بگیرم و هم این سری برای افراد تازه کار موجود باشه.

آشنایی اولیه

WPF مخفف عبارات Windows Presentation Foundation است که ویکی پدیا این گونه ترجمه می‌کند : بنیاد نمایش ویندوزی. در برنامه نویسی «ویندوز فرم» ما تمرکز دقیقی بر ساخت رابط کاربری برنامه به خصوص در رزولوشن‌های مختلف نداریم و در بسیاری از اوقات کد با رابط کاربری به شدت وابسته میشد که با ارائه WPF از نسخه‌ی سوم دات نت فریم ورک به بعد، این مشکل حل شد و همچنین عملیات refactoring را بسیار ساده‌تر کرد. در حالت ویندوز فرم به خاطر وابستگی شدید کد و UI، عملیات بهینه سازی کد اصلا موفق نبود.

WPF از ترکیب عناصر دو بعدی و سه بعدی، اسناد، موارد چند رسانه‌ای و رابط کاربری تشکیل شده‌است و موتور رندر آن بر اساس اطلاعات برداری از کارت گرافیک جهت نمایش ظاهر برنامه کمک می‌گیرد که باعث تهیه برنامه‌ای با رابط کاربری سریعتر، مقیاس پذیرتر و بدون وابستگی به رزولوشن می‌شود.

جداسازی رفتارها و ظاهر برنامه

همانطور که گفتیم بخش رابط کاربری دیگر مستقل از کد برنامه شده است و ظاهر برنامه توسط زبان نشانه گذاری XAML ایجاد می‌شود و بخش کد هم با یکی از زبان‌های موجود در مجموعه دات نت نوشته خواهد شد. نهایتا این دو بخش توسط رویدادها، فرامین و DataBinding با یکدیگر متصل می‌شوند. از مزایای جدا بودن این ویژگی:

عدم وابستگی این دو بخش
طراح و کدنویس می‌توانند هر کدام به طور جداگانه کار کنند.
ابزارهای طراحی میتوانند به طور جداگانه‌ای بر روی اسناد XML کار کنند بدون اینکه نیاز به درگیری با کدنویسی داشته باشند.

یکی از برنامه هایی که به طراحی رابط کاربری با پشتیبانی از XAML می‌پردازد برنامه Microsoft Experssion Blend از مجموعه Blend است

Rich Composition

یکی از ویژگی‌های XAML، ساخت اشیاء ترکیبی هست که به راحتی با ترکیب تگ‌ها با یکدیگر و قرار دادن هر شیء داخل یک شیء دیگر می‌توان به یک شیء جدید دست یافت؛ مثل قرار دادن مجموعه ویدیوها در یک لیست. شیء زیر از ترکیب سه شیء تصویر و متن و دکمه ایجاد شده است:

```
<Button Margin="148,123,126,130">
            <StackPanel Orientation="Horizontal">
                <Image Source="speaker.png" Stretch="Uniform"/>
                <TextBlock Text="Play Sound" VerticalAlignment="Center" Margin="10" />
            </StackPanel>
        </Button>
```

Highly Customizable

با استفاده از مفهوم Style همانند آنچه که در Html و CSS دارید می‌توانید اشیاء خود را خصوصی سازی کنید و ظاهر آن شیء را به طور کل تغییر دهید.

Resolution Independence

عدم وابستگی به رزولوشن یا وضوح تصویر دارد و به جای واحد پیکسل، از یک واحد منطقی که یک نود و ششم اینچ است، بهره می‌برد. از آنجا که این سیستم بر اساس وکتور ایجاد شده است، مقیاس پذیری آن در تغییر اندازه یا وضوح تصویر به شدت بالا رفته است.

به زودی در قسمت اول این سری کار را با XAML آغاز خواهیم کرد.

نظرات خوانندگان

نویسنده: حسین جهان
تاریخ: ۱۳۹۴/۰۲/۰۶ ۱۹:۵۴

با تشکر. امیدوارم ادامه داشته باشه چون من هم به تازگی کار با WPF رو شروع کردم.

نویسنده: حسن محسنی
تاریخ: ۱۳۹۴/۰۲/۰۷ ۱۴:۱۰

یه سری سیلورلایت چند سال قبل در این سایت منتشر شد:
https://github.com/DotNETTips/Exports/tree/master/Others/Silverlight

البته این سری برای سیلورلایت 4 هست و سیلورلایت هم به نظر توسط مایکروسافت کنار گذاشته شده اما اصول XAML اون منهای مباحث خاص سیلورلایت، می‌تونه پایه WPF و حتی مباحث برنامه‌های ویندوزهای 8 و 10 باشه.

عنوان: آشنایی با WPF قسمت اول : ساختار سلسله مراتبی
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۲/۰۷ ۲:۰
آدرس: www.dotnettips.info
گروه‌ها: WPF

عناصر رابط کاربری WPF با یکدیگر یک رابطه‌ی سلسله مراتبی دارند. به این رابطه، درخت منطقی یا Logical Tree می‌گویند که به توصیف ارتباط اجزای رابط کاربری می‌پردازد. نوع دیگری از درخت نیز وجود دارد که به آن درخت بصری یا Visual Tree می‌گویند. این درخت شامل عناصری است که باعث نمایش کنترل پدر می‌شوند و کنترل پدر بدون آن‌ها هیچ ظاهر نمایشی ندارد.
به عنوان مثال شما یک دکمه را در نظر بگیرید. این دکمه شامل عناصری چون Border,Block Text,Content Presenter می‌شود تا بتواند به عنوان یک دکمه نمایش یابد و بدون وجود این عناصر، کنترل دکمه هیچ ظاهری ندارد و در واقع با رندر شدن کنترل‌های فرزندان، دکمه معنا پیدا می‌کند. به تصویر بالا دقت کنید که به خوبی مرز بین درخت منطقی و درخت بصری را نمایش می‌دهد.
شکل سلسله مراتبی بالا از طریق کد زیر به دست آمده است:

```
<Window>
    <Grid>
        <Label Content="Label" />
        <Button Content="Button" />
    </Grid>
</Window>
```

درخت بصری می‌تواند به ما کمک کند تا بتوانیم بر روی عناصر تشکیل دهنده، یک کنترل قدرت عمل داشته باشیم و آن‌ها را مورد تغییر قرار دهیم.

Dependency Properties

خاصیت‌های وابستگی همان خاصیت‌ها یا property هایی هستند در ویندوزفرم با آن‌ها سر و کله می‌زدید ولی در اینجا تفاوت‌هایی با پراپرتی‌های قبلی وجود دارد که باعث ایجاد مزایای زیادی شده است.

اول اینکه بر خلاف پراپرتی‌های ویندوز فرم که در خود فیلدهای تعیین شده همان کنترل ذخیره می‌شدند، در این روش کلید (نام پراپرتی) و مقدار آن داخل یک شیء دیکشنری قرار می‌گیرند که از شیء DependencyObject ارث بری شده است و این شیء والد یک متد با نام GetValue برای دریافت مقادیر دارد. مزیت این روش این است که بیخود و بی‌جهت مانند روش قبلی، ما فیلدهایی را تعریف نمی‌کنیم که شاید به نصف بیشتر آن‌ها، حتی نیازی نداریم. در این حالت تنها فیلدهایی از حافظه را دریافت و ذخیره می‌کنیم که واقعا به آن‌ها نیاز داریم. فیلدها یا مقادیر پیش فرض موقع ایجاد شیء در آن ذخیره می‌شوند.

دومین مزیت این روش خاصیت ارث بری مقادیر از عناصر بالاتر درخت منطقی است. موقعی که از طرف شما برای فرزندان این عنصر مقداری تعیین نشده باشد، سیستم به سمت گره‌ها یا عناصر بالا یا والد حرکت می‌کند و اولین عنصری را که مقدارش تنظیم شده باشد، برای فرزندان در نظر می‌گیرد. به این استراتژی یافتن یک مقدار، استراتژی Resolution می‌گویند.

سومین مزیت آن وجود یک سیستم اعلان یا گزارش آنی است. در صورتی که شما یک تابع callback را برای یک پراپرتی ست نمایید، با تغییر این پراپرتی تابع معرفی شده صدا زده خواهد شد.

Value Resolution Strategy

همانطور که در بالا اشاره کردیم دریافت مقادیر یک کنترل از طریق یک استراتژی به اسم Resolution انجام می‌شود که طبق تصویر زیر از بالا به پایین بررسی می‌شود. در هر کدام از مراحل زیر اگر مقداری یافت شد، همان مقدار را انتخاب می‌کند. از متد SetValue هم برای درج مقدار استفاده می‌شود. برای مثال در مرحله‌ی سوم بررسی می‌شود که آیا کاربر برای کنترل مورد نظر مقداری را تنظیم کرده است یا خیر؛ اگر آری، پس از آن استفاده می‌کند و اگر پاسخ خیر بود، بررسی می‌کند آیا style برای آن موجود است که مقداری برایش تنظیم شده باشد یا خیر و الی آخر...

جادوی پشت صحنه

مقادیر پراپرتی‌ها در کلاسی استاتیک به اسم Dependency Property ذخیره می‌شوند که این ذخیره در حالت نام و مقدار است و مقدار آن شامل callback و مقدار پیش فرض است. شکل زیر نتیجه‌ی شکل دقیق‌تری را نسبت به قبلی در هنگام پیمایش درخت منطقی به سمت بالا، نشان می‌دهد.

نحوه‌ی تعریف یک خاصیت وابسته که باید به صورت ایستا تعریف شود به صورت زیر است و برای دریافت و درج مقدار جدید از یک پراپرتی معمولی کمک می‌گیریم:

```
// Dependency Property
public static readonly DependencyProperty CurrentTimeProperty =
     DependencyProperty.Register( "CurrentTime", typeof(DateTime),
     typeof(MyClockControl), new FrameworkPropertyMetadata(DateTime.Now));

// .NET Property wrapper
public DateTime CurrentTime
{
    get { return (DateTime)GetValue(CurrentTimeProperty); }
    set { SetValue(CurrentTimeProperty, value); }
}
```

یک قانون در WPF وجود دارد و آن اینست که نام خاصیت‌های وابسته را با کلمه Property به پایان ببرید مثل CurrentTimeProperty.

در مورد خاصیت‌های وابسته و کدنویسی آن ها در مطالب آینده بیشتر بحث خواهیم کرد.

عنوان: آشنایی با WPF قسمت دوم: Layouts بخش اول
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۲/۰۸ ۲:۱۵
آدرس: www.dotnettips.info
گروه‌ها: WPF

layoutها یکی از مهمترین قسمت‌های یک برنامه‌ی کاربردی هستند. چیدمان کنترل‌ها روی یک ناحیه با دادن مختصات پیکسلی ثابت، ممکن است در یک محیط محدود خود را خوب نشان بدهد ولی به زودی با تغییر محیط برنامه و یا تغییر وضوح تصویر صفحه نمایش، برنامه از کنترل خارج خواهد شد؛ در نتیجه استفاده از Layoutها یا پنل‌ها در WPF امری حیاتی و مهم هستند. layoutها که با نام container هم شناخته می‌شوند وظیفه دارند که بگویند چه کنترل‌هایی در کجا و چگونه باید در صفحه‌ی برنامه قرار بگیرند. پنل‌های توکار در WPF به دسته‌های زیر تقسیم می‌شوند:

Grid Panel

Stack Panel

Dock Panel

Wrap Panel

Canvas Panel

StackPanel

این پنل یکی از ساده‌ترین و سودمندترین پنل هاست که بر اساس جهت Orientation افقی یا عمودی که به آن تنظیم می‌شود، کنترل‌ها را کنار یکدیگر یا زیر یکدیگر قرار می‌دهد. این کنترل برای ایجاد و تهیه لیست‌های مختلف مناسب است. در ساختار داخلی کنترل‌های لیست و کامبو و منوها که در WPF موجود هستند این پنل استفاده شده است. کد زیر یک نمونه کاربرد Stack Panel را نشان می‌دهد که به صورت عمودی چینش شده است.

```
<StackPanel>
  <TextBlock Margin="10" FontSize="20">How do you like your coffee?</TextBlock>
  <Button Margin="10">Black</Button>
  <Button Margin="10">With milk</Button>
  <Button Margin="10">Latte machiato</Button>
  <Button Margin="10">Chappuchino</Button>
</StackPanel>
```

نکته‌ی مهم اینکه میتوانید در اینجا از یک nested layout هم استفاده کنید بدین صورت که یک layout را داخل یک layout دیگر قرار دهید. کد زیر ترکیب دو stack panel را به صورت افقی و عمودی به ما نشان می‌دهد:

```
<StackPanel Orientation="Vertical"> <!-- Vertical is the default -->
  <Label Background="Red">Red 1</Label>
  <Label Background="LightGreen">Green 1</Label>
  <StackPanel Orientation="Horizontal">
    <Label Background="Red">Red 2</Label>
    <Label Background="LightGreen">Green 2</Label>
    <Label Background="LightBlue">Blue 2</Label>
    <Label Background="Yellow">Yellow 2</Label>
    <Label Background="Orange">Orange 2</Label>
  </StackPanel>
  <Label Background="LightBlue">Blue 1</Label>
  <Label Background="Yellow">Yellow 1</Label>
  <Label Background="Orange">Orange 1</Label>
</StackPanel>
```

Dock Panel

احتمالا به خاطر نامش، نحوه کارش را حدس زده اید. این پنل، اشیاء موجود را در 4 جهت و مرکز می‌چسباند. مشخص نمودن جهت چسبیده شدن هر کنترل توسط خاصیت DockPanel.Dock صورت می‌گیرد و مقدار Left، مقدار پیش فرض است. در صورتی که بخواهید المانی را در مرکز بچسبانید باید آن را به عنوان آخرین المان معرفی کرده و در Dock Panel مقدار خاصیت LastChildFill را با True برابر کنید.

```
<DockPanel LastChildFill="True">
    <Button Content="Dock=Top" DockPanel.Dock="Top"/>
    <Button Content="Dock=Bottom" DockPanel.Dock="Bottom"/>
    <Button Content="Dock=Left"/>
    <Button Content="Dock=Right" DockPanel.Dock="Right"/>
    <Button Content="LastChildFill=True"/>
</DockPanel>
```

به نحوه‌ی تعریف خاصیت DockPanel.Dock دقت کنید به این نوع خاصیت‌ها، Attached Dependency Property (شاید در فارسی بتوانیم *خاصیت‌های وابستگی متصل* صدا بزنیم) می‌گویند. این خاصیت‌ها نوع خاصی از خاصیت‌های وابستگی هستند که به شما اجازه می‌دهند مقداری را به شیءایی نسبت دهید که آن شیء چیزی در مورد آن نمی‌داند. بهترین مثال در مورد این ویژگی، پنل‌ها هستند که یکی از موارد استفاده‌ی از آن را در بالا می‌بینید. هر پنل می‌تواند تا بی نهایت المان فرزند داشته باشد که هر المان باید خواصش توسط پنل مشخص گردد. ولی اگر پنل ما تعداد زیادی فرزند داشته باشد، نوشتن خواص هر کدام از فرزندها داخل تگ پنل، کاری غیر ممکن است. اینجاست که این نوع خاصیت‌ها خودشان را نشان می‌دهند. پس به این نحو مقادیر، داخل کنترل هر تگ تعریف می‌شود ولی توسط پنل مورد استفاده قرار می‌گیرد. نحوه‌ی نوشتن این نوع خاصیت: ابتدا یک پیشوند از نوع تگ پنل را در ابتدا آورده و سپس بعد از .(نقطه) نام خاصیت را ذکر می‌کنیم.

نحوه‌ی تعریف این نوع خاصیت‌ها در یک کلاس به صورت زیر است که برای شیء یا پنل canvas می‌باشد:

```
public static readonly DependencyProperty TopProperty =
    DependencyProperty.RegisterAttached("Top",
    typeof(double), typeof(Canvas),
    new FrameworkPropertyMetadata(0d,
```

```
        FrameworkPropertyMetadataOptions.Inherits));

public static void SetTop(UIElement element, double value)
{
    element.SetValue(TopProperty, value);
}

public static double GetTop(UIElement element)
{
    return (double)element.GetValue(TopProperty);
}
```

در مثال dockPanel بالا در هر طرف تنها یک المان قرار دادیم. برای قرار دادن المان‌های بیشتر در طرفین، تنها ذکر یک المان جدید با خاصیت Dockpanel.dock کافی است و الویت نمایش آن‌ها بر اساس ترتیب نوشتن تگها توسط شماست. مثال زیر این نکته را نشان می‌دهد:

```
<Button Content="Dock=Top" DockPanel.Dock="Top"/>
        <Button Content="Dock=Bottom" DockPanel.Dock="Bottom"/>
        <Button Content="Dock=Left"/>
        <Button Content="Dock=Left2"/>
        <Button Content="Right2" DockPanel.Dock="Right"/>
        <Button Content="Dock=Right" DockPanel.Dock="Right"/>
        <Button Content="LastChildFill=True"/>
```

Wrap Panel

این پنل بسیار شبیه StackPanel هست ولی مثل آن اشیاء را در یک سطر یا ستون ادامه نمی‌دهد؛ بلکه با رسیدن به انتهای پنجره، سطر یا ستون جدیدی را آغاز می‌کند. در stack panel با پایان پنجره، ادامه اشیا آن قابل مشاهده نبود ولی در این شیء با اتمام و رسیدن به لبه‌ی پنجره، اشیاء در سر جدید (افقی) یا ستون جدید (عمودی) نمایش داده می‌شوند. این پنل‌ها می‌توانند در ساخت تب‌ها و نوار ابزار استفاده شوند.

Canvas Panel

پایه‌ای‌ترین layout موجود در WPF است. موقعیت قرارگیری المان‌های فرزندش بر اساس نقاط تعیین شده است.این پنل بیشتر برای رسم اشکال و گرافیک دو بعدی مناسب است و اصلا برای قرارگیری کنترل‌های WPF روی آن توصیه نمی‌شود و مشکل winformها در آن رخ خواهد داد.

شروع ترسیم یک شکل دو بعدی روی آن بر اساس دوتا از چهار "خاصیت‌های وابستگی متصل" صورت می‌گیرد که به شرح زیر هستند:

Canvas.LEFT

Canvas.RIGHT

Canvas.TOP

Canvas.BOTTOM

نمونه از کد نوشته شده آن به صورت زیر است:

```
<Canvas>
    <Rectangle Canvas.Left="40" Canvas.Top="31" Width="63" Height="41" Fill="Blue"  />
    <Ellipse Canvas.Left="130" Canvas.Top="79" Width="58" Height="58" Fill="Blue"  />
    <Path Canvas.Left="61" Canvas.Top="28" Width="133" Height="98" Fill="Blue"
          Stretch="Fill" Data="M61,125 L193,28"/>
</Canvas>
```

ترتیب قرارگیری اشکال روی هم در canvas به ترتیبی انجام می‌گیرد که در XAML نوشته اید ولی می‌توان با استفاده از خاصیت Canvas.ZIndex این ترتیب را تغییر داد.

```
<Canvas>
    <Ellipse Fill="Green" Width="60" Height="60" Canvas.Left="30" Canvas.Top="20"
             Canvas.ZIndex="1"/>
    <Ellipse Fill="Blue"  Width="60" Height="60" Canvas.Left="60" Canvas.Top="40"/>
</Canvas>
```

شکل زیر در سمت راست، نتیجه‌ی کد بالاست ولی بدون ذکر ZIndex شکل سمت چپ نتیجه‌ی کار خواهد بود.

ViewBox

شاید عده‌ای به سختی آن را یک Layout بدانند و بیشتر آن را یک کنترل معمولی می‌شناسند ولی وظیفه‌ی آن بسیار شبیه Layout هاست. خصوصیتی که این شیء دارد این است که با تغییر اندازه محیط برنامه به هر نحوی، یک تغییر مقیاس روی اشیاء داخل آن رخ می‌دهد و کنترل‌ها به همراه متون و هر چیزی که در درخت منطقی و بصری است Scale آن تغییر می‌یابند.
نمونه‌ی کد زیر را تست کنید تا تفاوت بین دو Button را ببینید:

```
<StackPanel Orientation="Vertical">
        <Button Content="Test" />
        <Viewbox Stretch="Uniform">
            <Button Content="Test" />
        </Viewbox>
    </StackPanel>
```

نتیجه‌ی کار:

در بخش دوم Layoutها مبحث گرید و ساخت Layout اختصاصی و تعدادی از خاصیت‌ها را بررسی خواهیم کرد.

نظرات خوانندگان

نویسنده: افشین عباسپور
تاریخ: ۱۳۹۴/۰۲/۰۸ ۱۵:۲۶

خیلی ممنون از آموزش مفید و کمیاب WPF ... تا جایی که من میدونم امکان استفاده از کامپوننت‌های WPF در WinApplication وجود دارد . آیا بلعکس این هم امکان پذیر است ؟ استفاده کامپوننت‌های Win Form در Wpf امکان داره ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۲/۰۸ ۱۵:۳۵

بله. WPF برای اینکار دارای WindowsFormsHost هست. (خارج از موضوع بحث جاری است)

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۲/۰۸ ۱۵:۳۵

ممنون از شما
این صفحه رو مطالعه کنید.

عنوان: آشنایی با WPF قسمت سوم: Layouts بخش دوم
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۲/۰۹ ۰:۵۵
آدرس: www.dotnettips.info
گروه‌ها: WPF

در مقاله قبلی در مورد تعدادی از Layoutها صحبت کردیم و در این بخش به ادامه‌ی آن پرداخته و دو مبحث GridPanel و Custom Layout را بررسی می‌کنیم.

GridPanel

پنل پیش فرضی است که موقع ایجاد یک پروژه جدید WPF ایجاد می‌شود. چیدمان این نوع پنل به صورت سطر و ستون است و کارکرد آن بسیار مشابه جداول در HTML می‌باشد؛ با این تفاوت که در اینجا انعطاف پذیری بیشتری وجود دارد. هر سلول می‌تواند شامل چندین کنترل شود و یا هر کنترل می‌تواند چندین سلول را به خود احتصاص دهند و حتی می‌تواند روی کنترل‌های دیگر قرار بگیرند و همپوشانی کنترل‌ها را داشته باشیم.

تگ Grid Panel شامل دو تگ برای تعریف سطرها و ستون‌ها می‌باشد با استفاده از تگ Row Definition و Column Definition به تعیین تعداد سطر و ستون‌ها و اندازه آن‌ها می‌پردازیم:

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="*" />
        <RowDefinition Height="28" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="200" />
    </Grid.ColumnDefinitions>
</Grid>
```

گرید پنل بالا شامل 4 سطر و دو ستون است و تعیین اندازه آن‌ها توسط دو خاصیت Width و Height مشخص شده است که نحوه مقداردهی آن‌ها به صورت زیر است:

Fixed : یک مقدار ثابت، مثل سطر آخری که در کد بالا قرار می‌گیرد. این مقدار بر اساس یک واحد منطقی است و نه پیکسل که در این مقاله قبلا بررسی کرده‌ایم.

Auto : به مقداری که احتیاج دارد فضایی را بخود اختصاص می‌دهد.

***** : هر آنچه از فضای موجود باقی مانده است را به خود اختصاص می‌دهد. علامت ستاره یک واحد نسبی است؛ به این صورت که می‌توانید مقدار فضا را به صورت زیر نیز بیان کنید.*3 و *2 به این معنی است که از پنج قسمت فضای باقیمانده سه قسمت و بعدی دو قسمت را به خود اختصاص می‌دهد. عبارت * با *1 برابر است. عموما با این علامت فضا را به شکل درصد بیان می‌کنند:

```
<ColumnDefinition Width="69*" />   <!-- Take 69% of remainder -->
    <ColumnDefinition Width="31*"/> <!-- Take 31% of remainder -->
```

نحوه‌ی اضافه کردنالمان‌ها به گرید به صورت زیر پس از تعیین تعداد سطرها و ستون‌ها انجام می‌گیرد و جایگاه هر المان در ستون یا سطر مربوطه توسط یک attached Dependency Property به نام‌های Grid.Column یا Grid.Row صورت می‌گیرد. خصوصیات Horizontal alignment و vertical Alignment هم برای تعیین موقعیت قرار گیری اشیاء در سلول به کار می‌روند و فاصله‌ی آن‌ها (کنترل ها) از لبه‌های گرید با margin محاسبه می‌شود.

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
```

```
        <RowDefinition Height="*" />
        <RowDefinition Height="28" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="200" />
    </Grid.ColumnDefinitions>
    <Label Grid.Row="0" Grid.Column="0" Content="Name:"/>
    <Label Grid.Row="1" Grid.Column="0" Content="E-Mail:"/>
    <Label Grid.Row="2" Grid.Column="0" Content="Comment:"/>
    <TextBox Grid.Column="1" Grid.Row="0" Margin="3" />
    <TextBox Grid.Column="1" Grid.Row="1" Margin="3" />
    <TextBox Grid.Column="1" Grid.Row="2" Margin="3" />
    <Button Grid.Column="1" Grid.Row="3" HorizontalAlignment="Right"
            MinWidth="80" Margin="3" Content="Send"  />
</Grid>
```

تغییر اندازه در سمت کد هم می‌تواند توسط کدهای صورت گیرد.

```
Auto sized GridLength.Auto
Star sized new GridLength(1,GridUnitType.Star)
```

```
Fixed size new GridLength(100,GridUnitType.Pixel)
```

مثال:

```
Grid grid = new Grid();

ColumnDefinition col1 = new ColumnDefinition();
col1.Width = GridLength.Auto;
ColumnDefinition col2 = new ColumnDefinition();
col2.Width = new GridLength(1,GridUnitType.Star);

grid.ColumnDefinitions.Add(col1);
grid.ColumnDefinitions.Add(col2);
```

قابلیت تغییر اندازه‌ی سطر و ستون توسط کاربر

یکی از تگ‌های ویژه داخل گری،د تگ Grid Splitter است. برای قرارگیری تگ splitter ابتدا باید یک سطر یا ستون بین سطر و ستون هایی که میخواهید از یکدیگر جدا شوند ایجاد کنید و اندازه‌ی آن را auto تعیین کنید و سپس مانند بقیه‌ی اشیا توسط Grid.Column یا Grid.Row مانند کد زیر تگ splitter را به آن اختصاص دهید.

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*"/>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
    <Label Content="Left" Grid.Column="0" />
    <GridSplitter HorizontalAlignment="Right"
                  VerticalAlignment="Stretch"
                  Grid.Column="1" ResizeBehavior="PreviousAndNext"
                  Width="5" Background="#FFBCBCBC"/>
    <Label Content="Right" Grid.Column="2" />
</Grid>
```

خاصیت ResizeBehavior مشخص می‌کند که ستون یا سطرهای کناری کدام باید تغییر اندازه داشته باشند.

BasedOnAlignment	مقدار پیش فرض این گزینه است و مشخص می‌کند سطر یا ستونی طرفی باید تغییر اندازه دهد که در Alignment آن آمده است
CurrentAndNext	ستون یا سطر جاری به همراه ستون یا سطر بعدی
PreviousAndCurrent	ستون یا سطر جاری به همراه ستون یا سطر قبلی
PreviousAndNext	سطر یا ستون قبلی و بعدی که بهترین گزینه برای انتخاب است.

خاصیت ResizeDirection جهت تغییر اندازه را مشخص می‌کند که شامل سه مقدار Row,Column و Auto است که مقدار پیش فرض آن auto است و نیازی به ذکر آن نیست و خود سیستم میداند که باید تغییر اندازه در چه جهتی صورت بگیرد.

ساخت Custom Layout یا یک پنل سفارشی (اختصاصی)

در این دو قسمت، شما با پنل‌های متفاوتی آشنا شدید که قابلیت‌های مفیدی داشتند؛ ولی گاهی اوقات هیچ کدام از این‌ها به کار شما نمی‌آیند و دوست دارید پنلی داشته باشید که مطابق میل شما عمل کند. برای ساخت یک پنل سفارشی یک کلاس می‌سازیم که از کلاس Panel ارث بری می‌کند. در اینجا دو متد برای Override کردن وجود دارند:

MeasureOverride : تعیین اندازه پنل بر اساس اندازه تعیین شده برای المان‌های فرزند و فضای موجود.

ArrangeOverride : مرتب سازی المان‌ها در فضای موجود نهایی.

کد نمونه:

```
public class MySimplePanel : Panel
{
    // Make the panel as big as the biggest element
    protected override Size MeasureOverride(Size availableSize)
    {
        Size maxSize = new Size();

        foreach( UIElement child in InternalChildern)
        {
            child.Measure( availableSize );
            maxSize.Height = Math.Max( child.DesiredSize.Height, maxSize.Height);
            maxSize.Width= Math.Max( child.DesiredSize.Width, maxSize.Width);
        }
    }

    // Arrange the child elements to their final position
    protected override Size ArrangeOverride(Size finalSize)
    {
        foreach( UIElement child in InternalChildern)
        {
            child.Arrange( new Rect( finalSize ) );
        }
    }
}
```

لینک‌های زیر تعدادی از پنل‌های سفارشی پر طرفدار هستند که بر روی اینترنت به اشتراک گذاشته شده اند:

TreeMapPanel

Animating Tile Panel

Radial Panel

Element Flow Panel

Ribbon Panel

خواصی که باید در Layoutها با آنها بیشتر آشنا شویم:

Horizontal & Vertical Alignment

با دادن این خاصیت به کنترل‌های موجود، نحوه قرار گیری و موقعیت آن‌ها مشخص می‌گردد. جدول زیر بر ساس انواع موقعیت‌های مختلف تشکیل شده است:

Margin & Padding

این خاصیت‌ها حتما برای شما آشنا هستند. خاصیت margin فاصله کنترل از لبه‌های Layout است و خاصیت Padding فاصله محتویات کنترل از لبه‌های کنترل است.

Clipping

در صورتی که خاصیت ClipToBounds پنل برابر False باشند به این معناست که المان‌ها میتوانند از لبه‌های پنل خارج شوند، در صورتی که برابر True باشد مقدار خارج شده نمایش نمی‌یابد.

Scrolling

موقعیکه از پنلی استفاده می‌کنید که با تمام شدن ناحیه‌اش روبرو شده‌اید ولی کنترل‌های داخلش هنوز ادامه دارند، نیاز به یک اسکرول به شدت احساس می‌شود. در این حالت می‌توان از ScrollViewer استفاده کرد.

```
<ScrollViewer>
    <StackPanel>
        <Button Content="First Item" />
        <Button Content="Second Item" />
```

```
        <Button Content="Third Item" />
    </StackPanel>
</ScrollViewer>
```

عنوان: آشنایی با WPF قسمت چهارم: کنترل ها
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۲/۱۹ ۱:۲۰
آدرس: www.dotnettips.info
گروه‌ها: WPF

WPF همانند Windows Form شامل ابزارها یا کنترل‌های داخلی است که می‌توانند در تهیه‌ی یک برنامه بسیار کارآمد باشند. در این بخش به بررسی تعدادی از این کنترل‌ها می‌پردازیم و مابقی آن‌ها را در قسمت‌های آینده بررسی خواهیم کرد. در این نوشتار سعی بر این است که یک فرم ساده را با آن ایجاد کرده و مورد استفاده قرار دهیم.
این فرم دارای اطلاعاتی شامل : نام، جنسیت ، زمینه‌های کاری، کشور، تاریخ تولد و تصویر می‌باشد.

TextBlock
همان Label قدیمی خودمان است که برای نمایش متون کاربر دارد. متن داخل آن بین دو تگ قرار می‌گیرد و یا از خاصیت Text آن کمک گرفته خواهد شد. حتما از خاصیت Width و height آن برای مقداردهی کمک بگیرید، زیرا در غیر آن صورت کل Container خود را خواهد پوشاند. در صورتی که متنی در مکان خود جا نشود می‌توان از دو ویژگی استفاده کرد. آن را برش داد یا به خطوط بعدی شکست. برای حذف یا برش باقی مانده متن می‌توان از خصوصیت TextTrimming استفاده کرد که سه مقدار می‌گیرد:

None مقدار پیش فرض

CharacterEllipsis با نزدیک شدن به آخر پهنای کار از ... استفاده می‌نماید. در صورتی که لیستی یا مورد مشابهی دارید میتواند بسیار کاربردی باشد.

WordEllipsis این گزینه هم مانند مورد بالاست با این تفاوت که سعی دارد تا آنجا که ممکن است خود را به آخرین حرف کلمه برساند تا شکستگی در وسط کلمه اتفاق نیفتد و آخرین کلمه کامل دیده شود و بعد ... قرار بگیرد؛ هر چند در تست‌های خودم تفاوتی مشاهده نکردم.

گزینه TextWrapping جهت شکستن یک خط به خطوط است؛ موقعی که متن شما به انتهای صفحه می‌رسد، این ویژگی باعث می‌شود متن به بیرون از پنجره نرفته و یک خط به سمت پایین حرکت کند. این گزینه سه مقدار را دارد:

تصویر زیر حالت اصلی نمایش بدون نیاز به Wrap شدن است:

None : مقدار پیش فرض که خصوصیت Wrap را به همراه ندارد.

Wrap : فعال سازی ویژگی TextWrapping

WrapWithOverflow : فرق این گزینه با گزینه بالا در این است که ، گزینه بالا در هر موقعیتی که پیش بیاید عمل خط شکن را روی عبارت یا حتی آن کلمه انجام می‌دهد. ولی در این گزینه فرصت خط شکنی مثل بالا فراهم نیست و اگر روی کلمه‌ای خط شکنی رخ دهد، مابقی آن کلمه از ناحیه‌ی خودش خارج شده و از کلمه‌ی بعدی، خط شکنی صورت می‌گیرد.

خصوصیت **LineStackingStrategy** :

این خصوصیت فاصله‌ی بین خطوط را با استفاده از یک واحد منطقی dp مشخص می‌کند. هر چند دو گزینه دیگر هم دارد که دو تصویر زیر را در این صفحه به شما نمایش می‌دهد:

برای ساخت فرم از یک گرید با سه ستون و 6 سطر استفاده می‌کنم.

```
<Grid Margin="5">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"></ColumnDefinition>
        <ColumnDefinition Width="*"></ColumnDefinition>
            <ColumnDefinition Width="Auto"></ColumnDefinition>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
            <RowDefinition Height="Auto"></RowDefinition>
            <RowDefinition Height="Auto"></RowDefinition>
            <RowDefinition Height="Auto"></RowDefinition>
            <RowDefinition Height="Auto"></RowDefinition>
            <RowDefinition Height="Auto"></RowDefinition>
            <RowDefinition Height="Auto"></RowDefinition>

        </Grid.RowDefinitions>
</Grid>
```

در ستون اول نام فیلدهای مورد نظر را می‌نویسیم و در ستون دوم هم کنترل‌های مد نظر هر فیلد را قرار خواهیم داد. در صورتی که دوست دارید کار از راست به چپ پشتیبانی کند از گزینه OverflowDirection در تگ پنجره Window استفاده نمایید.

در داخل گرید بعد از تعریف سطر و ستون، همانطور که قبلا توضیح دادیم کنترل‌های TextBlock را اضافه می‌کنیم:

```
<TextBlock Grid.Column="0"  Grid.Row="0" VerticalAlignment="Center" HorizontalAlignment="Left"
>Name</TextBlock>
        <TextBlock Grid.Column="0"  Grid.Row="1" VerticalAlignment="Center" HorizontalAlignment="Left"
>Gender</TextBlock>
        <TextBlock Grid.Column="0"  Grid.Row="2" VerticalAlignment="Center" HorizontalAlignment="Left"
>Field Of Work</TextBlock>
        <TextBlock Grid.Column="0"  Grid.Row="3" VerticalAlignment="Center" HorizontalAlignment="Left"
>Country</TextBlock>
        <TextBlock Grid.Column="0"  Grid.Row="4" VerticalAlignment="Center" HorizontalAlignment="Left"
>Birth Date</TextBlock>
```

```
<TextBox Grid.Row="0" Grid.Column="1" Name="Txtname" HorizontalAlignment="Left" Margin="5" Width="200"
></TextBox>
```

برای فیلد نام، از کنترل TextBox استفاده کردم که با محدود کردن Width آن اندازه ثابت به آن دادم. در صورتی که width ذکر نشود یا به Auto ذکر شود، در صورتی که متنی که کاربر تایپ می‌کند، بیش از اندازه تعیین شده کنترل Textbox باشد، کنترل هم همراه متن بزرگتر خواهد شد و تا پایان محدوده سلولی اش در گرید کش خواهد آمد.

Buttons

برای فیلد جنسیت Gender هم از RadioButton کمک گرفتم که با استفاده از خاصیت GroupName می‌توان دسته‌ای از این کنترل‌ها را با هم مرتبط ساخت تا با انتخاب یک آیتم جدید از همان گروه، آیتم قبلی که انتخاب شده بود از حالت انتخاب خارج شده و آیتم جدیدی انتخاب شود. از خاصیت IsChecked می‌توان برای انتخاب یک آیتم بهره برد.

به صورت کلی دکمه‌ها به چند دسته زیر تقسیم می‌شوند:

Button

ToggleButton

CheckBox

RadioButton

که همگی این عناصر از کلاسی به نام ButtonBase مشتق شده اند. کد زیر RadioButtonها را به صورت عمودی چینش کرده است:

```
<StackPanel Orientation="Vertical" Grid.Row="1" Grid.Column="1" Margin="10">
            <RadioButton GroupName="Gender" Name="RdoMale" IsChecked="True" >Male</RadioButton>
            <RadioButton GroupName="Gender" Name="RdoFemale" Margin="0 5 0 0" >Female</RadioButton>
        </StackPanel>
```

برای فیلد زمینه کاری ، لیست کشورها و تاریخ تولد از کدهای زیر کمک گرفتم:

```
<StackPanel Orientation="Horizontal" Grid.Row="2" Grid.Column="1" Margin="10">
            <CheckBox Name="ChkActor" >Actor/Actress</CheckBox>
            <CheckBox Name="ChkDirector" >Director</CheckBox>
            <CheckBox Name="ChkProducer" >Producer</CheckBox>
        </StackPanel>

        <ListBox Grid.Row="3" Grid.Column="1" Margin="10"  Height="80">
        <ListBoxItem>
                <TextBlock>UnitedStates</TextBlock>
            </ListBoxItem>
            <ListBoxItem>
                <TextBlock >UK</TextBlock>
            </ListBoxItem>
            <ListBoxItem>
                <TextBlock >France</TextBlock>
            </ListBoxItem>
            <ListBoxItem>
                <TextBlock >Japan</TextBlock>
            </ListBoxItem>
        </ListBox>

        <Calendar Grid.Row="4" Grid.Column="1" HorizontalAlignment="Left" Margin="10"></Calendar>
```

برای لیست کشورها می‌توان از یک ListBox یا ComboBox استفاده کرده که هر آیتم داخل آنها در یک تگ ListBoxItem یا ComboBoxItem قرار می‌گیرد. اگر از حالت ListBox استفاده می‌کنید، در صورتی که آیتم‌ها از ارتفاع لیست بیشتر شود به طور خودکار یک Scrollbar برای آن‌ها در نظر گرفته خواهد شد و نیازی نیست که آن را دستی اضافه کنید.

برای تصویر شخص، قصد دارم آن را در گوشه‌ی سمت راست و بالا قرار دهم. برای همین محل ستون آن را ستون سوم یا

اندیس دوم انتخاب کرده و از آنجا که این عکس حالت پرسنلی دارد، می‌تواند چند سطر را به خود اختصاص دهد که با کمک خاصیت Rowspan، چهار سطر، کنترل را ادامه دادم. برای ستون‌ها هم می‌توان از خاصیت ColumnSpan استفاده کرد. همچنین دوست دارم یک دکمه هم روی تصویر در گوشه‌ی سمت چپ و پایین قرار داده که کاربر با انتخاب آن به انتخاب عکس یا تغییر آن بپردازد. برای همین از یک پنل گرید استفاده کردم و کنترل دکمه را روی تصویر قرار دادم. همپوشانی کنترل‌ها در اینجا صورت گرفته است.

```
<Grid Grid.Row="0" Grid.Column="2" Grid.RowSpan="4">
            <Grid.RowDefinitions>
                <RowDefinition Height="*"></RowDefinition>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*"></ColumnDefinition>
            </Grid.ColumnDefinitions>
            <Image HorizontalAlignment="Right" Source="man.jpg" Stretch="UniformToFill"
VerticalAlignment="Top" Width="100" Height="150"></Image>
            <Button Width="25" Height="15"    Padding="0"  HorizontalAlignment="Left"
VerticalAlignment="Bottom" Margin="0,0,0,83">
                 <TextBlock VerticalAlignment="Center" Margin="0 -7 0 0">...</TextBlock>
            </Button>
        </Grid>
```

خاصیت **Stretch** کنترل Image در بالا، نحوه‌ی نمایش تصویر را نشان می‌دهد که چهار مقدار دارد:

None : تصویر، اندازه‌ی اصلی خود را حفظ کرده و هر مقدار آن که در کنترل جا شود، نمایش می‌یابد و بسته به سایز تصویر ممکن است گوشه هایی از تصویر نمایش نیابد.

Fill : تصویر را داخل کنترل به زور جا داده تا پهنا و ارتفاع عکس، هم اندازه کنترل می‌شود.

Uniform : تصویر بزرگ را با در نظر گرفتن نسبت پهنا و ارتفاع تصویر، با یکدیگر در کنترل جا می‌دهد.

UniformToFill : تصویر، کل کنترل را می‌گیرد ولی نسبت پهنا و عرض را حفظ کرده ولی قسمت هایی از تصویر در کنترل دیده نمی‌شود.

همانطور که قبلا هم گفتیم، خود کنترل دکمه شامل زیر کنترل‌هایی می‌شود که یکی از آن‌ها TextBlock است و از طریق خصوصیت *.TextBlock دیگر خصوصیات آن قابل تنظیم است و البته برای خصوصی سازی بیشتر هم می‌توان یک TextBlock را به صورت Nested یعنی داخل تگ Button تعریف کنید که ما همین کار را کرده ایم.

فرم نهایی ما به صورت زیر است:

در صورتی که دوست دارید جهت ListBox را از عمودی به افقی تغییر دهید می‌توانید از پنل‌های Stack یا Wrap استفاده کنید که تعریف آن به شکل زیر است:

```
<ListBox>
    <ListBox.ItemsPanel>
        <ItemsPanelTemplate>
            <VirtualizingStackPanel Orientation="Horizontal" />
        </ItemsPanelTemplate>
    </ListBox.ItemsPanel>
</ListBox>
```

بدین صورت ListBox به شکل زیر تغییر می‌یابد:

و در صورتی که می‌خواهید Scroll حذف شود و از Wrap استفاده کنید، کد را به شکل زیر تعریف کنید. فراموش نکنید که اسکرول افقی را غیرفعال کنید؛ وگرنه نتیجه کار به شکل بالا خواهد بود.

```
<ListBox ScrollViewer.HorizontalScrollBarVisibility="Disabled">
    <ListBox.ItemsPanel>
        <ItemsPanelTemplate>
            <WrapPanel />
        </ItemsPanelTemplate>
    </ListBox.ItemsPanel>
</ListBox>
```

نتیجه:

Calendar

تقویم یکی دیگر از کنترل‌های موجود است که شامل خصوصیات زیر است:

DisplayDate : تاریخ پیش فرض و اولیه تقویم را مشخص می‌کند؛ در صورتی که ذکر نشود تاریخ جاری درج می‌شود.

```
<Calendar DisplayDate="01.01.2010" />
```

از خصوصیات دیگر در این زمینه می‌توان به *DisplayDateStart* و *DisplayDateEnd* اشاره کرد که محدوده‌ی نمایش تاریخ تقویم را مشخص می‌کند. کد زیر تنها تاریخ‌های روز اول ماه ابتدای سال 2015، تا روز اول ماه پنجم 2015 را نمایش می‌دهد:

```
<Calendar DisplayDateStart="01.01.2015" DisplayDateEnd="05.01.2015" />
```

SelectionMode : نحوه‌ی انتخاب تاریخ را مشخص می‌کند:

SingleDate : فقط یک تاریخ قابل انتخاب است.

SingleRange : می‌توانید از یک تاریخ تا تاریخ دیگر را انتخاب کنید. ولی نمی‌توانید مجددا چند انتخاب دیگر را در جای جای تقویم داشته باشید. مثلا از تاریخ 5 آپریل تا 10 آپریل را انتخاب کرده‌اید؛ ولی دیگر نمی‌توانید تاریخ 15 آپریل یا محدوده‌ی 15 آپریل تا 20 آپریل را انتخاب کنید. چون تنها قادر به انتخاب یک رنج یا محدوده تاریخی هستید.

MultipleRanges : بر خلاف گزینه‌ی بالایی هر محدوده تاریخی قابل انتخاب است.

```
<Calendar SelectionMode="MultipleRange" />
```

نکته بعدی در مورد غیرفعال کردن بعضی از تاریخ هاست که شما قصد ندارید به کاربر اجازه دهید آن‌ها را انتخاب کند. برای مثال تاریخ‌های 1 آپریل تا 10 آپریل را از دسترس خارج کنید. برای همین از خصوصیت BlackoutDates استفاده می‌کنیم که نحوه‌ی تعریف آن به شرح زیر است که در این کد دو محدوده‌ی تاریخی غیر فعال شده اند:

```
<Calendar>
    <Calendar.BlackoutDates>
        <CalendarDateRange Start="01/01/2010" End="01/06/2010" />
        <CalendarDateRange Start="05/01/2010" End="05/03/2010" />
    </Calendar.BlackoutDates>
</Calendar>
```

DisplayMode : به طور پیش فرض، تقویم ماه‌ها را نشان می‌دهد ولی میتوانید آن را توسط این خصوصیت روی سال یا دهه و ماه هم تنظیم کنید.

با انتخاب سال *Year* ، تقویم ماه‌های یک سال را نمایش می‌دهد.

با انتخاب دهه *Decades* سال‌های یک دهه‌ی تعیین شده را نشان می‌دهد و با انتخاب ماه Month روزهای هر ماه در آن نمایش داده می‌شود.

در هنگام انتخاب این گزینه، به داخل تقویم نگاه نکنید، بلکه به سر تیتر آن نگاه کنید.

```
<Calendar DisplayMode="Year" />
```

2015
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec

عنوان: **آشنایی با WPF قسمت پنجم : DataContext بخش اول**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۲/۲۱ ۲۲:۵۰
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF

یکی از مهمترین قسمت‌های برنامه، کار با داده‌های بانک اطلاعاتی (یا در کل منابع اطلاعاتی) است. اینکه چگونه با آن‌ها ارتباط برقرار کنیم و آن‌ها را در یک قالب کاربر پسند به کاربران برنامه نشان دهیم. افزودن شیء DataContext و مفاهیمی چون DataBinding باعث ارتباط سریع‌تر و راحت‌تری با منبع داده‌ها شده است. همچنین این قابلیت وجود دارد که هر گونه به روز آوری در اطلاعات دریافت شده، شما را با خبر سازد تا بتوانید طبق آن چه که می‌خواهید اطلاعات نمایشی را به روز کنید. در این مقاله به نحوه‌ی ارتباط بین منبع داده با DataContext و سپس کنترل‌هایی را چون Grid و ListBox و ... در رابطه با این منابع داده بررسی می‌کنیم.

در مورد بررسی ارتباط با داده‌ها در WPF باید سه مورد را بشناسیم:

DataContext: این شیء اتصالش را به منبع داده‌ها برقرار کرده و هر موقع داده‌ای را نیاز داریم، از طریق این شیء تامین می‌شود.
DataBinding: یک واسطه بین DataContext و هر آن چیزی است که قرار است از داده‌ها تغذیه کند. در تعریفی رسمی‌تر می‌گوییم: روشی ساده و قدرتمند بوده و واسطی است بین مدل تجاری و رابط کاربری. هر زمانی که داده‌ای تغییر کند، ما را آگاه می‌سازد که می‌تواند یک ارتباط یک طرفه یا دو طرفه باشد.

DataTemplate: نحوه‌ی فرمت بندی و نمایش داده‌ها را تعیین می‌کند.
ابتدا کار را با یک مثال ساده آغاز می‌کنیم. قصد داریم فرمی را که در قسمت قبلی ساختیم، با استفاده از یک منبع داده پر کنیم: ابتدا قبل از هر چیزی کلاس فرم قبلی را پیاده سازی می‌کنیم. در این پیاده سازی از یک enum برای انتخاب زمینه‌های کاری هم کمک گرفته ایم و همچنین با یک متد ایستا، منبع داده‌ی تک رکوردی را جهت تست برنامه آماده کرده‌ایم:

```
public enum FieldOfWork
    {
        Actor=0,
        Director=1,
        Producer=2
    }
    public class Person
    {
        public string Name { get; set; }

        public bool Gender { get; set; }

        public string ImageName { get; set; }

        public string Country { get; set; }

        public DateTime Date { get; set; }

        public IList<FieldOfWork> FieldOfWork { get; set; }
        public static Person GetPerson()
        {
            return new Person()
            {
                Name = "Leo",
                Gender = true,
                ImageName ="man.jpg",
                Country = "Italy",
                Date = DateTime.Now
            };
        }
    }
```

حالا لازم است که این منبع داده را در اختیار DataContext بگذاریم. وارد بخش کد نویسی شده و در سازنده‌ی پنجره کد زیر را می‌نویسیم:

```
DataContext = Person.GetPerson();
```

```
public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            DataContext = Person.GetPerson();
        }
    }
```

با این کار، ارتباط شما با منبع داده آغاز می‌شود و طبق درخواست‌هایی که از DataBinding به آن می‌رسد، اطلاعات را تحویل DataBinding می‌دهد. برای نمایش داده‌ها در کنترل‌ها و استفاده از DataBinding، به سراغ خصوصیات وابسته می‌رویم. در حال حاضر فعلا برنامه را با دو کنترل عکس و نام که رشته‌ای هستند آغاز می‌کنیم؛ چون بقیه‌ی کنترل‌ها کمی متفاوت هستند.
همانطور که می‌دانید متن کنترل TextBox توسط خصوصیت Text پر می‌شود و برای همین در این خصوصیت می‌نویسیم:

```
Text="{Binding Name}"
```

علامت {} را باز کرده و در ابتدا نام Binding را می‌آوریم. سپس بعد از یک فاصله، نام پراپرتی کلاسی را که حاوی اطلاعات مدنظر است، می‌نویسیم و بدین صورت اتصال برقرار می‌شود. برای کنترل عکس هم وضعیت به همین صورت است:

```
Source="{Binding ImageName}"
```

حال برنامه را اجرا کرده و دو کنترل textbox و Image را بررسی می‌کنیم:

کلمه‌ی Leo داخل کادر متنی قرار گرفته و عکس اینبار به صورت ایستا خوانده نشده، بلکه نام عکس از طریق یک منبع داده برای آن فراهم شده است.

اطلاع از به روزرسانی در منبع داده‌ها:
حال این نکته پیش می‌آید که اگر همین اطلاعات دریافت شده در مدل منبع داده تغییر کند، چگونه می‌توانیم از این موضوع مطلع شده و همین اطلاعات به روز شده را که نمایش داده‌ایم، تغییر دهیم. بنابراین جهت اطلاع از این مورد، کد را به شکل زیر تغییر می‌دهیم.

کار را از یک کلاس آغاز می‌کنیم. از اینترفیس INotifyPropertyChanged ارث بری کرده و در آن یک رویداد و یک متد را تعریف

می‌کنیم و کمی در هم در تعریف Propertyها دست می‌بریم. فعلا اینکار را فقط برای پراپرتی Name انجام می‌دهیم:

```
private string _name;
        public string Name
        {
            get { return _name; }
            set
            {
                _name = value;
                OnPropertyChanged("Name");
            }
        }
        public event PropertyChangedEventHandler PropertyChanged;

        private void OnPropertyChanged(string property)
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(property));
            }
        }
```

در کد بالا یک رویداد از نوع PropertyChangedEventHandler تعریف می‌کنیم که وظیفه‌ی به روزآوری را به عهده دارد؛ ولی صدا زدن این رویداد بر عهده‌ی ماست و خود به خود صدا زده نمی‌شود. پس نیاز است متدی را فراهم کرده تا بدانیم که چه خصوصیتی تغییر یافته‌است و از آن طریق رویداد را فراخوانی کنیم و به رویداد بگوییم که کدام پراپرتی تغییر کرده است. این متد را OnpropertyChanged می‌نامیم که آرگومان ورودی آن نام خصوصیتی است که تغییر یافته است و پس از ارزیابی از صحت آن، رویداد را Invoke می‌کنیم.

در بخش Setter آن خصوصیت هم باید این متد را صدا زده و نام خصوصیت را به آن پاس بدهیم تا موقعی که مدل تغییر پیدا کرد، بگوید که خصوصیت Name بوده است که تغییر کرده است.

برای اینکه بدانیم کد واقعا کار می‌کند و تستی بر آن زده باشیم، فعلا دکمه‌ی Save را به Change تغییر می‌دهیم و کد داخل پنجره را بدین صورت تغییر می‌دهیم:

```
public partial class MainWindow : Window
    {
        private Person person;
        public MainWindow()
        {
            InitializeComponent();
            person = Person.GetPerson();
            DataContext = person;

        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            person.Name = "Leonardo Decaperio";
        }
    }
```

متغیر کلاسی را از حالت محلی Local به عمومی Global تغییر دادم که از طریق دکمه‌ی منبع داده در دسترس باشد. حال در رویداد دکمه نام بازیگر را تغییر می‌دهم. برنامه را اجرا کنید و بر روی دکمه کلیک کنید. باید بعد از یک لحظه‌ی کوتاه، نام بازیگر از Leo به Leonardo Decaperio تغییر کند.

این کد واقعا کدی مفید جهت به روزرسانی است ولی مشکلی دارد که نام پراپرتی باید به صورت String به آن پاس شود که در یک برنامه بزرگ این مورد یک مشکل خواهد شد و اگر نام خصوصیت تغییر کند باید نام داخل آن هم تغییر کند؛ پس کد را به شکل دیگری بازنویسی می‌کنیم:

```
private string _name;
        public string Name
        {
            get { return _name; }
            set
            {
                _name = value;
```

```
                OnPropertyChanged();
            }
        }

  private void OnPropertyChanged([CallerMemberName] string property="")
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(property));
            }
        }
```

در متد OnPropertyChanged در کنار پارامتر اول، ویژگی attribute به نام CallerMemberName را که در فضای نام system.runtime.compilerservice قرار دارد استفاده می‌کنیم (دات نت 4.5). این ویژگی، نام پراپرتی یا متدی که متد OnpropertyChnaged را صدا زده است، به دست می‌آورد. پارامتر اول را هم اختیاری می‌کنیم که سیستم بر ورود پارامتر اجباری نداشته باشد و نهایتا در هر پراپرتی تنها لازم است همانند بالا، خط زیر ذکر شود:

```
OnPropertyChanged();
```

اگر الان یک تست از آن بگیرید، می‌بینید که بدون مشکل کار می‌کند. حالا همین متد را در setter تمام پراپرتی‌هایی که دوست دارید از تغییر آن‌ها آگاه شوید قرار دهید.

کد این قسمت

در قسمت‌های آینده به بررسی تبدیل مقادیر و framework element و کنترل‌ها می‌پردازیم.

نظرات خوانندگان

نویسنده: مرتضی رییسی
تاریخ: ۱۳۹۴/۰۲/۲۲ ۲۳:۱۸

ممنون بابت تمام زحماتتون.
سوال: رخداد PropertyChanged کی و کجا مقدار دهی میشه؟ طبق برنامه شما هیچ وقت این رخداد مقداری نمیگیره و همیشه نال خواهد بود. لطفاً راهنمایی بفرمایید.
در ضمن اگه میشه مثالها و کلاسها رو بصورت کامل (using و namespace و بدنه کامل کلاس) بذارین و به تکه کد کوتاهی بسنده نکنید تا کمتر دچار سردرگمی بشیم.

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۲/۲۳ ۰:۱۹

اگه درست منظورتون رو متوجه شدم باشم پاسخ شما در متن بالا قرار داره:
"در متد OnPropertyChanged در کنار پارامتر اول، ویژگی attribute به نام CallerMemberName را که در فضای نام system.runtime.compilerservice قرار دارد استفاده می‌کنیم (دات نت 4.5). این ویژگی، نام پراپرتی یا متدی که متد OnpropertyChnaged را صدا زده است، به دست می‌آورد. "
تعریف ویژگی CallermemberName قبل از پارامتر باعث می‌شود به طور خودکار نام پراپرتی Name به سمت متد ارسال شود. یعنی با کد زیر برابری خواهد کرد:

```
OnPropertyChanged("Name");
```

test-1da6cfbbddbf48e1bd8b88c0c24a2c7b.zip

نویسنده: مرتضی رییسی
تاریخ: ۱۳۹۴/۰۲/۲۳ ۱۸:۴۱

مشکل پیدا شد.

```
public  class Person:INotifyPropertyChanged
```

اینترفیس رو پیاده سازی نکرده بودم. ممنون

عنوان: آشنایی با WPF قسمت پنجم : DataContext بخش دوم
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۲/۲۴ ۱۰:۵۵
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF

در ادامه قسمت قبلی قصد داریم دو کنترل دیگر را نیز بایند کنیم؛ ولی از آنجا که مقادیر آن‌ها رشته‌ای یا عددی نیست و مقداری متفاوت هست، از مبحثی به نام ValueConverter استفاده خواهیم کرد.

Value Converter چیست؟

موقعی که شما قصد بایند کردن دو نوع داده متفاوت را به هم دارید، نیاز به یک کد واسط پیدا می‌کنید تا این کد واسط مقادیر شما را از مبدا دریافت کرده و تبدیل به نوعی کند که مقصد بتواند از آن استفاده کند یا بلعکس. **ValueConverter** نام کلاسی است که از یک اینترفیس به نام **IValueConverter** ارث بری کرده است و شامل دو متد تبدیل نوع از مبدا به مقصد **Convert** و دیگری از مقصد به مبدا **ConvertBack** می‌شود که خیلی کمتر پیاده سازی می‌شود.

پیاده سازی یک کلاس مبدل سه مرحله دارد:

مرحله اول: ساخت کلاس ValueConverter

مرحله دوم : تعریف آن به عنوان یک منبع یا ریسورس

مرحله سوم : استفاده از آن در عملیات بایند کردن Binding

در مرحله اول، نحوه پیاده سازی کلاس ValueConverter به شکل زیر است:

```
public class BoolToVisibilityConverter : IValueConverter
{
    public object Convert(object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        // Do the conversion from bool to visibility
    }

    public object ConvertBack(object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        // Do the conversion from visibility to bool
    }
}
```

در متد تبدیل باید مقداری را که کنترل نیاز دارد، بر اساس مقادیر کلاس، ایجاد و بازگشت دهید.

در مرحله‌ی دوم نحوه تعریف مبدل ما در پنجره XAML به صورت زیر می‌باشد:

```
<Window x:Class="test.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:نامی دلخواه="clr-namespace:فضای نامی که کلاس مبدل در آن قرار دارد"
        >

    <Window.Resources>
        <نام دلخواهی که در بالا تعریف کرده اید: ClassNameنام کلاس  x:Key="کلید این آیتم در ریسورس"/>
    </Window.Resources>
```

کلمه‌های کلیدی xmlns که مخفف XML NameSpace هستند جهت تعریف دسترسی به فضاهای نام طراحی شده‌اند و طبق تعریف مایکروسافت همان مفهوم تگ‌های تعریف فضای نام در XML را دارند که توسط مایکروسافت توسعه یافته‌اند. این تعریف‌ها در تگ ریشه (در اینجا window) تعریف می‌شوند. دو فضای نام اولی که به طور پیش فرض در همه جای پروژه قرار دارند، اشاره به فریم ورک WPF دارند. کلمه‌ی کلیدی x در خط شماره سه، نام دلخواهی است که دسترسی ما را به خصوصیات یا تعاریف XAML موجود در sdk باز می‌کند؛ مثلا استفاده از خصوصیاتی چون x:key یا x:class را به همراه دارد.

پس الان باید خط چهارم برای ما روشن باشد؛ فضای نام جدیدی را در برنامه خودمان ایجاد کرده‌ایم که این تگ به آن اشاره می‌کند و نام دلخواهی هم برای اشاره به این فضای نام برایش در نظر گرفته‌ایم. هر موقع در برنامه این نام دلخواه تعیین شده قرار گیرد، یعنی اشاره به این فضای نام که در قسمت Window.resource خط هشتم تعریف شده است.

در خط هشتم، یک ریسورس (منبع) را به برنامه معرفی کرده‌ایم:

ریسورس‌ها برای ذخیره سازی داده‌ها در سطح یک کنترل، سطح محلی در یک پنجره، یا سطح عمومی در کل پروژه به کار می‌روند. محدودیتی در ذخیره داده‌ها وجود ندارد و هر چقدر که دوست دارید می‌توانید داده به آن پاس کنید. این داده‌ها می‌توانند یک سری اطلاعات ذخیره شده در یک ساختار ساده تا یک ساختار سلسه مراتبی از کنترل‌ها باشند. ریسورس‌ها به شما این اجازه را می‌دهند تا داده‌ها را در یک مکان ذخیره کرده و آن‌ها را در یک یا چندجا مورد استفاده قرار دهید.

از آن جا که مباحث ریسورس‌ها را در یک مقاله‌ی جداگانه بررسی می‌کنیم، فقط به ذکر نکات بالا جهت کد فعلی بسنده خواهیم کرد و ادامه‌ی آن را در یک مقاله دیگر مورد بررسی قرار می‌دهیم.

هر ریسورس دارای یک نام یا یک کلید است که با خصوصیت x:key تعریف می‌شود.

ریسورس بالا یک کلاس را که در فضای نام دلخواهی قرار دارد، تعریف می‌کند و یک کلید هم به آن انتساب می‌دهد.

مرحله‌ی سوم معرفی ریسورس به عملیات Binding است:

```
{Binding نام پراپرتی کلاس, Converter={StaticResource کلید آیتم مربوطه در ریسورس},
ConverterParameter=پارامتری که به کلاس مبدل پاس می‌شود}
```

بخش اول دقیقا همان چیزی است که در قسمت قبلی یاد گرفتیم. معرفی پراپرتی که باید عمل بایند به آن صورت گیرد. قسمت بعدی معرفی مبدل است و از آن جا که تابع مبدل ما در یک منبع است، اینگونه می‌نویسیم: {} را باز کرده و ابتدا کلمه StaticResource فاصله و سپس کلید ریسورس که تابع را از ریسورس فراخوانی کند و قسمت بعدی هم پاس کردن یک پارامتر به تابع مبدل است.

حال که با اصول نوشتار آشنا شدیم کار را آغاز می‌کنیم.

قصد داریم یک مبدل برای فیلد جنیست درست کنیم. از آنجا که این فیلد Boolean است و خصوصیت IsChecked یک RadioButton هم Boolean است، می‌توان یک ارتباط مستقیم را ایجاد کرد. ولی مشکل در اینجا هست که True برای مذکر است و false برای مؤنث. در نتیجه تنها Radiobutton مربوطه به جنس مذکر به این حالت پاسخ می‌دهد و از آنجا که برای جنس مونث false در نظر گرفته شده است، انتخاب آن هم false خواهد بود. پس باید در مبدل، مقداری که کنترل می‌خواهد را شناسایی کرده و اگر مقدار با آن برابر بود، True را بازگردانیم. مقداری که هر کنترل درخواست می‌کند را از طریق پارامتر به تابع مبدل ارسال می‌کنیم. radiobutton مذکر، مقدار True را به عنوان پارامتر ارسال می‌کند و radiobutton مونث هم مقدار false را به عنوان پارامتر ارسال می‌کند. اگر تابع مبدل را ببینید، این مقدارها با پارامترها همخوانی دارند، True در غیر این صورت false بر

میگرداند.

مرحله اول تعریف کلاس ValueConveter:

```
using System;
using System.Globalization;
using System.Windows;
using System.Windows.Data;

namespace test.ValueConverters
{
    public class GenderConverter: IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            var ParameterString = parameter as string;
            if (ParameterString == null)
                return DependencyProperty.UnsetValue;

            bool bparam;

            bool test = bool.TryParse(parameter.ToString(), out bparam);
            if (test)
            {
                return ((bool)value).Equals(bparam);
            }
            return DependencyProperty.UnsetValue;
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}
```

در کد بالا پارامتر ارسالی را دریافت می‌کنیم و اگر برابر با Null باشد، مقداری برگشتی را عدم ذکر شدن خصوصیت وابسته اعلام می‌کنیم. در نتیجه انگار این خصوصیت مقداردهی نشده است. اگر مخالف Null باشد، کار ادامه پیدا می‌کند. در نهایت مقایسه‌ای بین پارامتر و مقدار پراپرتی (value) صورت گرفته و نتیجه‌ی مقایسه را برگشت می‌دهیم.

برای تعریف این مبدل در محیط XAML به صورت زیر اقدام می‌کنیم:

```
<Window x:Class="test.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:valueConverters="clr-namespace:test.ValueConverters"
        Title="MainWindow" Height="511.851" Width="525">

    <Window.Resources>
        <valueConverters:GenderConverter x:Key="GenderConverter"/>
    </Window.Resources>
</window>
```

کلمه valueConveter به فضای نام test.valueConverters اشاره می‌کند که در قسمت ریسورس، از کلاس GenderConverter آن استفاده می‌کنیم و کلیدی که برای این ریسورس در نظر گرفتیم را GenderConverter تعریف کردیم (هم نامی کلید ریسورس و نام کلاس مبدل اتفاقی است و ارتباطی با یکدیگر ندارند).

نکته: در صورتی که بعد از تعریف ریسورس با خطای زیر روبرو شدید و محیط طراحی Design را از دست دادید یکبار پروژه را بیلد کنید تا مشکل حل شود.

```
The name "GenderConverter" does not exist in the namespace "clr-namespace:test.ValueConverters".
```

اکنون در عملیات بایندینگ دو کنترل اینگونه می‌نویسیم:

```
<RadioButton GroupName="Gender" IsChecked="{Binding Gender, Converter={StaticResource GenderConverter}, ConverterParameter=True}" Name="RdoMale"  >Male</RadioButton>
                <RadioButton GroupName="Gender" IsChecked="{Binding Gender, Converter={StaticResource GenderConverter}, ConverterParameter=False}" Name="RdoFemale" Margin="0 5 0 0"  >Female</RadioButton>
```

حال برنامه را اجرا کرده و نتیجه را ببینید. برای تست بهتر می‌توانید جنسیت فرد را در منبع داده تغییر دهید. همچنین از آنجا که این مقدار برای جنس مذکر نیازی به بررسی و تبدیل ندارد می‌توانید عملیات بایند را عادی بنویسید:

```
<RadioButton GroupName="Gender" IsChecked="{Binding Gender}" Name="RdoMale"  >Male</RadioButton>
```

این کد می‌تواند برای تمامی وضعیت‌های دو مقداری چون جنسیت و وضعیت تاهل و ... هم به کار برود. ولی حالا سناریویی را تصور کنید که که مقادیر از دو تا بیشتر شود؛ مثل وضعیت تحصیلی ، دسترسی‌ها و غیره که این‌ها نیاز به داده‌های شمارشی چون Enumها دارند. روند کار دقیقا مانند بالاست:

هر کنترل مقداری از یک enum را میپذیرد که میتواند آن مقدار را با استفاده از پارامتر، به تابع مبدل ارسال کند و سپس تابع چک می‌کند که آیا چنین مقداری در enum مدنظر یافت می‌شود یا خیر؛ این کد الان کار می‌کندو واقعا هم کد درستی است و هیچ مشکلی هم ندارد. ولی برای یک لحظه تصور کنید پنجره شما شامل چهار خصوصیت enum دار است. یعنی الان باید 4 تابع مبدل بنویسید. پس باید کد را بازنویسی کنیم تا به هر enum که مدنظر است پاسخ دهد؛ به این ترتیب تنها یک تابع مبدل می‌نویسیم.

جهت یادآوری نگاهی به کلاس برنامه می‌اندازیم:

```
public enum FieldOfWork
    {
        Actor=0,
        Director=1,
        Producer=2
    }
    public class Person : INotifyPropertyChanged
    {
        public bool Gender { get; set; }

        public string ImageName { get; set; }

        public string Country { get; set; }

        public DateTime Date { get; set; }

        public FieldOfWork FieldOfWork { get; set; }

        private string _name;
        public string Name
        {
            get { return _name; }
            set
            {
                _name = value;
                OnPropertyChanged();
            }
        }
```

فعلا IList را از پراپرتی FieldOfWork بر میداریم تا این سناریو باشد که تنها یک Enum قابل انتخاب است:

```
public FieldOfWork FieldOfWork { get; set; }
```

بعدا حالت قبلی را بررسی میکنیم.

کد کلاس مبدل را به صورت زیر می‌نویسیم:

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Data;

namespace test.ValueConverters
{
    public class EnumConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            var ParameterString = parameter as string;
            if (ParameterString == null)
                return DependencyProperty.UnsetValue;

            if (Enum.IsDefined(value.GetType(), value) == false)
                return DependencyProperty.UnsetValue;

            object paramvalue = Enum.Parse(value.GetType(), ParameterString);
            return paramvalue.Equals(value);
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}
```

در مرحله اول مثل کد قبلی بررسی می‌شود که آیا پارامتری ارسال شده است یا خیر. در مرحله دوم بررسی می‌شود که نوع داده مقدار پراپرتی چیست یعنی چه Enum ایی مورد استفاده قرار گرفته است. اگر در Enum مقداری که در پارامتر به آن ذکر شده است وجود نداشته باشد، بهتر هست که کار در همین جا به پایان برسد؛ زیرا که یک پارامتر اشتباهی ارسال شده است و چنین مقداری در Enum وجود ندارد. در غیر اینصورت کار ادامه می‌یابد و پارامتر را به enum تبدیل کرده و با مقدار مقایسه می‌کنیم. اگر برابر باشند نتیجه true را باز میگردانیم.

کد قسمت ریسورس را با کلاس جدید به روز می‌کنیم:

```
<Window.Resources>
        <valueConverters:GenderConverter x:Key="GenderConverter"/>
        <valueConverters:EnumConverter x:Key="EnumConverter"></valueConverters:EnumConverter>
    </Window.Resources>
```

کد چک باکس‌ها هم به شکل زیر تغییر می‌یابد:

```
<CheckBox Name="ChkActor" IsChecked="{Binding FieldOfWork, Converter={StaticResource EnumConverter},
ConverterParameter=Actor}" >Actor/Actress</CheckBox>
                <CheckBox Name="ChkDirector" IsChecked="{Binding FieldOfWork, Converter={StaticResource
EnumConverter}, ConverterParameter=Director}" >Director</CheckBox>
                <CheckBox Name="ChkProducer" IsChecked="{Binding FieldOfWork, Converter={StaticResource
EnumConverter}, ConverterParameter=Producer}" >Producer</CheckBox>
```

کلاس GetPerson که منبع داده ما را فراهم می‌کند هم به شکل زیر است:

```
public static Person GetPerson()
        {
            return new Person()
            {
                Name = "Leo",
                Gender =true,
                ImageName ="man.jpg",
                Country = "Italy",
                FieldOfWork = test.FieldOfWork.Actor,
                Date = DateTime.Now.AddDays(-3)
```

```
            };
        }
```

برنامه را اجرا کنید تا نتیجه کار را ببینید. باید چک باکس Actor تیک خورده باشد. میتوانید منبع داده را تغییر داده تا نتیجه کار را ببینید.

بگذارید فیلد FieldOfWork را به حالت قبلی یعنی IList برگردانیم. در بسیاری از اوقات ما چند گزینه از یک Enum را انتخاب می‌کنیم، مثل داشتن چند سطح دسترسی یا چند سمت کاری و ...

کلاس را به همراه کد GetPerson به شکل زیر تغییر می‌دهیم:

```
public enum FieldOfWork
    {
        Actor=0,
        Director=1,
        Producer=2
    }
    public class Person : INotifyPropertyChanged
    {
        public bool Gender { get; set; }

        public string ImageName { get; set; }

        public string Country { get; set; }

        public DateTime Date { get; set; }

        public IList<FieldOfWork> FieldOfWork { get; set; }

        private string _name;
        public string Name
        {
            get { return _name; }
            set
            {
                _name = value;
                OnPropertyChanged();
            }
        }

        public static Person GetPerson()
        {
            return new Person
            {
                Name = "Leo",
                Gender = true,
                ImageName = "man.jpg",
                Country = "Italy",
                FieldOfWork = new FieldOfWork[] { test.FieldOfWork.Actor, test.FieldOfWork.Producer },
                Date = DateTime.Now.AddDays(-3)
            };
        }
}
```

دو Enum بازیگر و تهیه کننده را انتخاب کرده‌ایم، پس در زمان اجرا باید این دو گزینه انتخاب شوند.
کد مبدل را به صورت زیر می‌نویسیم:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Data;

namespace test.ValueConverters
{
    public class EnumList : IValueConverter
```

```
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            var ParameterString = parameter as string;
            if (ParameterString == null)
                return DependencyProperty.UnsetValue;

         var enumlist= (IList) value;

            if(enumlist==null && enumlist.Count<1)
                return DependencyProperty.UnsetValue;

            if (Enum.IsDefined(enumlist[0].GetType(), ParameterString) == false)
                return DependencyProperty.UnsetValue;


            /*
              foreach (var item  in enumlist)
            {
                object paramvalue = Enum.Parse(item.GetType(), ParameterString);
                bool result = item.Equals(paramvalue);
                if (result)
                    return true;
            }
            return false;
             */
            return (from object item in enumlist let paramvalue = Enum.Parse(item.GetType(),
ParameterString) select item.Equals(paramvalue)).Any(result => result);
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}
```

مقدار پراپرتی را به نوع IList تبدیل می‌کنید و اگر لیست معتبر بود، برنامه را ادامه می‌دهیم؛ در غیر اینصورت تابع خاتمه می‌یابد. بعد از اینکه صحت وجود لیست اعلام شد، بررسی میکنیم آیا Enum چنین مقداری که پارامتر ذکر کرده است را دارد یا یک پارامتر اشتباهی است.

در حلقه‌ای که به شکل توضیح درآمده، همه آیتم‌های مربوطه در لیست را بررسی کرده و اگر آیتمی برابر پارامتر باشد، True بر میگرداند و در صورتی که حلقه به اتمام برسد و آیتم پیدا نشود، مقدار False را برمی‌گرداند. این حلقه از آن جهت به شکل توضیح درآمده است که کد Linq آن در زیر نوشته شده است.

تعریف کلاس بالا در ریسورس:

```
<Window.Resources>
        <valueConverters:GenderConverter x:Key="GenderConverter"/>
        <valueConverters:EnumConverter x:Key="EnumConverter"></valueConverters:EnumConverter>
        <valueConverters:EnumList x:Key="EnumList"></valueConverters:EnumList>
    </Window.Resources>
```

و تغییر کلید ریسورس در خطوط چک باکس ها، باقی موارد همانند قبل ثابت هستند:

```
<CheckBox Name="ChkActor" IsChecked="{Binding FieldOfWork, Converter={StaticResource EnumList},
ConverterParameter=Actor}" >Actor/Actress</CheckBox>
                <CheckBox Name="ChkDirector" IsChecked="{Binding FieldOfWork, Converter={StaticResource
EnumList}, ConverterParameter=Director}" >Director</CheckBox>
                <CheckBox Name="ChkProducer" IsChecked="{Binding FieldOfWork, Converter={StaticResource
EnumList}, ConverterParameter=Producer}" >Producer</CheckBox>
```

نتیجه‌ی کار باید به شکل زیر باشد:

فیلد جنسیت و زمینه کاری از تابع مبدل به دست آمده است.

بدیهی است خروجی‌های بالا برای کنترل هایی است که مقدار Boolean را می‌پذیرند و برای سایر کنترل‌ها باید با کمی تغییر در کد و نوع برگشتی که تحویل خروجی متد مبدل می‌شود، دهید.

دانلود فایل‌های این قسمت

عنوان: آشنایی با WPF قسمت ششم : DataContext بخش سوم
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۳/۱۰ ۱:۰
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF

در قسمت قبلی با مبدل‌ها آشنا شدیم و با استفاده از این ویژگی، دو کنترل Radio Button و CheckBox را بایند کردیم. الان تنها دو کنترل مانده تا آن‌ها را متصل کنیم؛ کنترل ListBox و تقویم، که در این قسمت لیست را بررسی می‌کنیم.

ListBox

در مورد لیست، ما قبلا نام کشورها را با استفاده از تگ ListBoxItem به طور دستی اضافه می‌کردیم و هر گونه ویرایش و اضافه کردن عکس و دیگر اشیاء را داخل این تگ برای هر آیتم جداگانه انجام می‌دادیم؛ مثل تصویر زیر که هر آیتم شامل یک تگ تصویر و دو تگ TextBlock است که یکی از آن‌ها رنگی شده است. کد هر آیتم به طور جداگانه و دستی اضافه شده است.

ولی در روش بایندینگ چنین چیزی ممکن نیست و تنها با استفاده از یک Template موارد بالا را ایجاد می‌کنیم. پس محتویات سابق ListBox را حذف کرده و تگهای زیر را جهت افزودن یک قالب داده Data Template به شیء لیست اضافه می‌کنیم. حال اگر داده‌های لیست شده خود را روانه DataContext کنید باید این اطلاعات نمایش داده شوند.

```
<ListBox Grid.Row="3" Name="MyListBox" Grid.Column="1" Margin="10"  Height="80" >
               <ListBox.ItemTemplate>
                    <DataTemplate>
                        <WrapPanel>
                            <Image Width="24" Height="24" Source="{Binding Flag}"></Image>
                            <TextBlock Padding="5 5 0 0" Text="{Binding Name}"></TextBlock>
                        </WrapPanel>
                    </DataTemplate>
               </ListBox.ItemTemplate>
            </ListBox>
```

در برنامه ما مشکلی که هست، کد بالا جهت اتصال به DataContext ای است که قبلا پر شده است (DataContext کل View اصلی یا والد تمامی اشیاء مشتق از آن). حتما به یاد دارید که ما این شیء را با مدل یک رکورد ذخیره شده (مدل Person) در منبع داده‌ها پر کرده بودیم. پس استفاده از این روش در حال حاضر منتفی است. ممکن است شما در طول ساخت یک پنجره چندین و چند جا نیاز به منابع داده مختلفی داشته باشید ولی عموما DataContext با یک مدل جهت نمایش یا ذخیره یک رکورد بایند شده است. پس چکار کنیم؟

ارائه این نکته ضروری است که همه اشیاء خصوصیت DataContext را دارند و ما در مثال قبلی DataContext ریشه یا والد اشیاء را پر کردیم. اگر مقاله " ساختار سلسله مراتبی " را به یاد بیاورید، گفتیم که هر شیء در صورتیکه خصوصیت وابسته‌ای برایش تعریف نشده باشد، به سمت اشیاء والد حرکت می‌کند، به این جهت بود که همه‌ی کنترل‌ها به منبع داده‌ها دسترسی داشتند. پس

ما اگر DataContext لیست را پر کنیم، لیست دلیلی برای دسترسی به DataContext اشیاء والد ندارد و خصوصیت پر شده‌ی خودش را در نظر می‌گیرد. پس بیایید این مورد را امتحان کنیم:

من کلاس زیر را جهت ارسال لیستی از کشورها به همراه آدرس پرچمشان، بر می‌گردانم:

دلیل استفاده از کلاس ObservableCollection در کد زیر به جای استفاده از اشیایی چون Ilist و ... این بود که این کلاس به اینترفیس هایی چون INotifyPropertyChanged مزین گشته و هر گونه تغییری در این مجموعه، از قبیل حذف و اضافه را اطلاع رسانی کرده و مدل تغییر یافته را به سمت ویو هدایت می‌کند.

```
using System.Collections.ObjectModel;

namespace test
{
    public class Country
    {
        public string Flag {
            get { return "Images/flags/" + Name + ".png"; }
        }
        public string Name { get; set; }

        public int Id { get; set; }

        public ObservableCollection<Country> GetCountries()
        {
            var countries = new ObservableCollection<Country>();
            countries.Add(new Country(){Id =1,Name = "Afghanistan"});
            countries.Add(new Country() { Id = 2, Name = "Albania" });
            countries.Add(new Country() { Id = 3, Name = "Angola" });

            countries.Add(new Country() { Id = 4, Name = "Bahrain" });
            countries.Add(new Country() { Id = 5, Name = "Bermuda" });
            countries.Add(new Country() { Id =6, Name = "Iran" });

            return countries;
        }
    }
}
```

برنامه را اجرا کرده و انتظار داریم که بتوانیم لیست پر شده‌ای از داده‌ها را ببینیم؛ ولی در کمال تعجب لیست خالی است. خطایی هم برگردانده نمی‌شود.

دلیل این مشکل این است که DataContext برای نمایش یک Object تهیه شده است و در مورد داده‌های لیستی باید از خصوصیتی به نام ItemsSource استفاده کرد که برای داده‌های لیستی IEnumerables، بهینه شده است.

پس به این ترتیب می‌نویسیم :

```
public MainWindow()
        {
            InitializeComponent();
            person = Person.GetPerson();
            DataContext = person;

            //خط جدید
            MyListBox.ItemsSource = new Country().GetCountries();
        }
```

حال برنامه را اجرا کرده تا نتیجه را مشاهده کنید.

شکل‌های زیر یک نمودار از ارتباط با Object برای واکشی داده هاست:

شکل زیر همان نمودار بالا را ترسیم میکند ولی دیگر از مبدل پیش فرض WPF خبری نیست و مبدل اختصاصی به اسم ColorBrush جایگزین آن شده است:

نمودار زیر هم دسترسی به مجموعه ای از داده‌های لیستی است که از طریق ItemsSource خوانده می‌شوند:

کد زیر همچنین برای اتصال به کار می‌رود:

```
public MainWindow()
        {
            InitializeComponent();
            person = Person.GetPerson();
            DataContext = person;

            //خط جدید
            MyListBox.DataContext = new Country().GetCountries();
            MyListBox.SetBinding(ItemsControl.ItemsSourceProperty, new Binding());
        }
```

روش بالا اتصال را برقرار می‌کند ولی من توصیه چندانی در استفاده از آن نمی‌کنم. آزاد گذاشتن DataContext یک لیست، یک مزیت هم دارد و آن این است که خارج از تگ Itemها یعنی همان تگ لیست، موقعی که از بایندینگ استفاده می‌کنید، در واقع از DataContext کمک گرفته می‌شود؛ چون خود ListBox یک آیتم نیست که بخواهد با آیتمی در یک لیست سر و کله بزند. بلکه می‌تواند به راحتی به یک شیء، خود را بایند کند؛ مثال زیر نمونه‌ای از آن است.

پی نوشت : روش‌های دیگر بایند کردن همچون استفاده از منابع یا ریسورس‌ها یا استفاده از ViewModelها هم هستند که در آینده در مورد آن‌ها بیشتر صحبت خواهیم کرد.

حال که توانستیم لیست را پر کنیم باید کشوری را که در رکورد واکشی شده آمده است، در لیست انتخاب کنیم.
توجه داشته باشید که باید لیست را از طریق خصوصیت ItemsSource پر کرده باشید و DataContext را دستکاری نکرده باشید.
خصوصیت Country در کلاس Person می‌تواند به دو صورت زیر باشد:

```
public int Country { get; set; }
 public Country Country { get; set; }
```

که در هر دو حال از خصوصیت SelectedValue شی ListBox استفاده می‌شود. هر دو خط زیر به ترتیب برای استفاده از مقادیر بالا به کار می‌روند:

```
<ListBox Grid.Row="3" Name="MyListBox" Grid.Column="1" Margin="10"  Height="80" SelectedValuePath="Id"
SelectedValue="{Binding Country}"  >
<ListBox Grid.Row="3" Name="MyListBox" Grid.Column="1" Margin="10"  Height="80" SelectedValuePath="Id"
SelectedValue="{Binding Country.Id}"  >
```

خصوصیت SelectedValuePath برای مشخص کردن اینکه کدام فیلد را باید در آیتم‌های لیست، جست و جو کند به کار می‌رود که ما در اینجا فیلد Id را که در کلاس Country قرار دارد، معرفی کرده‌ایم.
خصوصیت‌های دیگر یک شیء لیستی چون ListBox و ComboBox و ... SelectedIndex است که اندیس یک آیتم انتخابی را بازگردانده یا جهت انتخاب یک آیتم، اندیس آن را دریافت می‌کند. SelectedItem و SelectedItems هم شیء یا شیءهایی از مدل را (در اینجا Country) که در لیست انتخاب شده‌اند، بر می‌گرداند (فقط خواندنی).

نتیجه اینکه اگر روش بالا با دستکاری DataContext انجام می‌گرفت دیگر استفاده از فیلد Country در مدل Peron ممکن نبود.