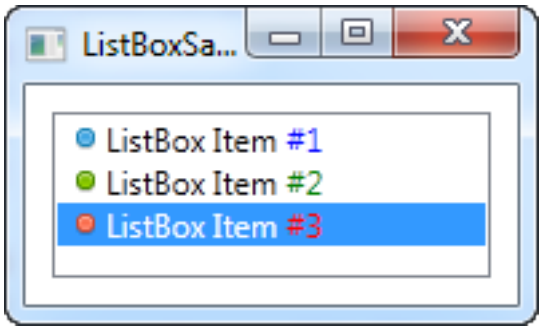
عنوان: آشنایی با WPF قسمت ششم : DataContext بخش سوم
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۳/۱۰ ۱:۰
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF

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;
        }
    }
}
```

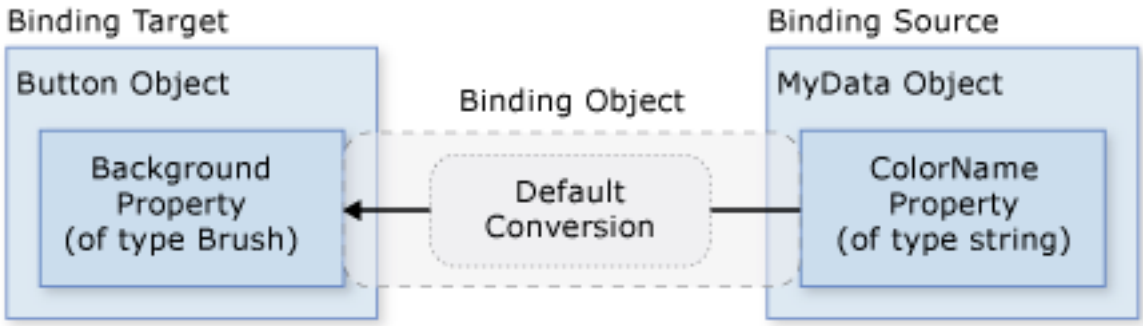
برنامه را اجرا کرده و انتظار داریم که بتوانیم لیست پر شده‌ای از داده‌ها را ببینیم؛ ولی در کمال تعجب لیست خالی است. خطایی هم برگردانده نمی‌شود.

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 :

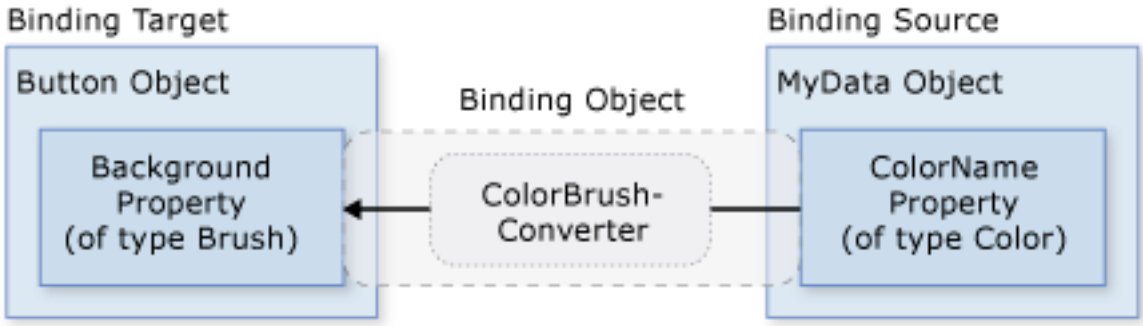
```
public MainWindow()
        {
            InitializeComponent();
            person = Person.GetPerson();
            DataContext = person;

            //خط جدید
            MyListBox.ItemsSource = new Country().GetCountries();
        }
```

حال برنامه را اجرا کرده تا نتیجه را مشاهده کنید.

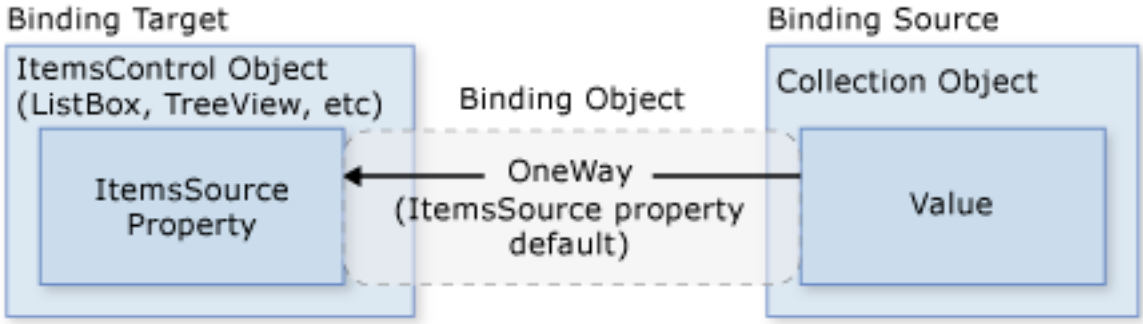
شکل‌های زیر یک نمودار از ارتباط با Object برای واکشی داده هاست:



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.