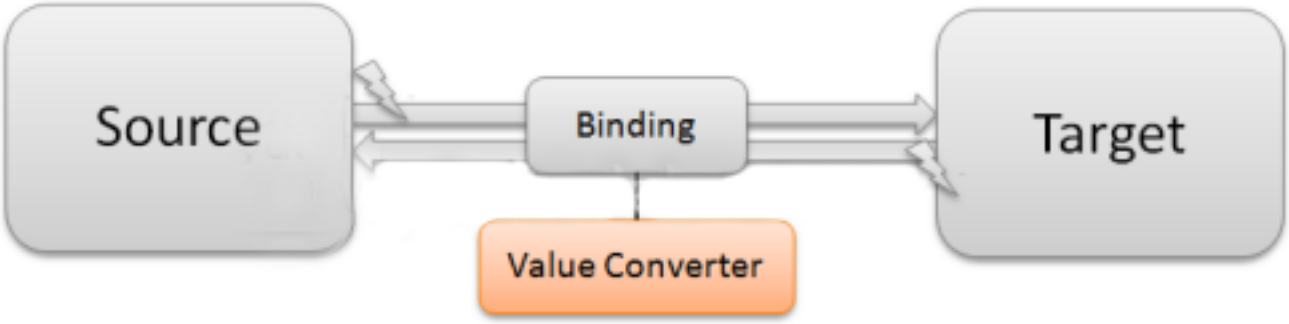
عنوان: آشنایی با WPF قسمت پنجم : DataContext بخش دوم
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۲/۲۴ ۱۰:۵۵
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF

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 کلاسClassName  x:Key="کلید این آیتم در ریسورس"/>نام دلخواهی که در بالا تعریف کرده اید:
    </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