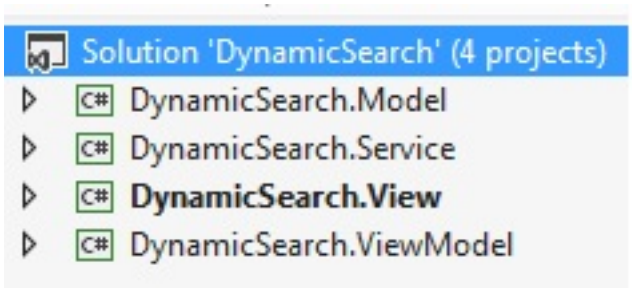
عنوان: **ایجاد فرم جستجوی پویا با استفاده از Expression ها**
نویسنده: محمد عیدی مراد
تاریخ: ۱۳۹۲/۰۳/۲۵ ۹:۲۵
آدرس: www.dotnettips.info
برچسب‌ها: LINQ, MVVM, Expression, DynamicLINQ

در مواردی نیاز است کاربر را جهت انتخاب فیلدهای مورد جستجو آزاد نگه داریم. برای نمونه جستجویی را در نظر بگیرید که کاربر قصد دارد: "دانش آموزانی که نام آنها برابر علی است و شماره دانش آموزی آنها از 100 کمتر است" را پیدا کند در شرایطی که فیلدهای نام و شماره دانش آموزی و عمل گر کوچک‌تر را خود کاربر به دلخواه برگزیرده.
روش‌های زیادی برای پیاده سازی این نوع جستجوها وجود دارد. در این مقاله سعی شده گام‌های ایجاد یک ساختار پایه برای این نوع فرم‌ها و یک ایجاد فرم نمونه بر پایه ساختار ایجاد شده را با استفاده از یکی از همین روش‌ها شرح دهیم.
اساس این روش تولید عبارت Linq بصورت پویا با توجه به انتخاب‌های کاربرمی باشد.
1- برای شروع یک سلوشن خالی با نام DynamicSearch ایجاد می‌کنیم. سپس ساختار این سلوشن را بصورت زیر شکل می‌دهیم.



در این مثال پیاده سازی در قالب ساختار MVVM در نظر گرفته شده. ولی محدودتی از این نظر برای این روش قائل نیستیم.
2- کار را از پروژه مدل آغاز می‌کنیم. جایی که ما برای سادگی کار، 3 کلاس بسیار ساده را به ترتیب زیر ایجاد می‌کنیم:

```
namespace DynamicSearch.Model
{
    public class Person
    {
        public Person(string name, string family, string fatherName)
        {
            Name = name;
            Family = family;
            FatherName = fatherName;
        }

        public string Name { get; set; }
        public string Family { get; set; }
        public string FatherName { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DynamicSearch.Model
{
    public class Teacher : Person
    {
        public Teacher(int id, string name, string family, string fatherName)
            : base(name, family, fatherName)
        {
            ID = id;
        }

        public int ID { get; set; }

        public override string ToString()
```

```
        {
            return string.Format("Name: {0}, Family: {1}", Name, Family);
        }
    }
}

namespace DynamicSearch.Model
{
    public class Student : Person
    {
        public Student(int stdId, Teacher teacher, string name, string family, string fatherName)
            : base(name, family, fatherName)
        {
            StdID = stdId;
            Teacher = teacher;
        }

        public int StdID { get; set; }
        public Teacher Teacher { get; set; }
    }
}
```

-3 در پروژه سرویس یک کلاس بصورت زیر ایجاد می‌کنیم:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using DynamicSearch.Model;

namespace DynamicSearch.Service
{
    public class StudentService
    {
        public IList<Student> GetStudents()
        {
            return new List<Student>
                {
                    new Student(1,new Teacher(1,"Ali","Rajabi","Reza"),"Mohammad","Hoeyni","Sadegh"),
                    new Student(2,new Teacher(2,"Hasan","Noori","Mohsen"),"Omid","Razavi","Ahmad"),
                };
        }
    }
}
```

-4 تا اینجا تمامی داده‌ها صرفا برای نمونه بود. در این مرحله ساخت اساس جستجو گر پویا را شرح می‌دهیم.

جهت ساخت عبارت، نیاز به سه نوع جزء داریم:

-اتصال دهنده عبارات ("و" ، "یا")

-عملوند (در اینجا فیلدی که قصد مقایسه با عبارت مورد جستجوی کاربر را داریم)

-عملگر (">" ، "<" ، "=" ، ....)

برای ذخیره المان‌های انتخاب شده توسط کاربر، سه کلاس زیر را ایجاد می‌کنیم (همان سه جزء بالا):

```
using System;
using System.Linq.Expressions;

namespace DynamicSearch.ViewModel.Base
{
    public class AndOr
    {
        public AndOr(string name, string title,Func<Expression,Expression,Expression> func)
        {
            Title = title;
            Func = func;
            Name = name;
        }

        public string Title { get; set; }
        public Func<Expression, Expression, Expression> Func { get; set; }
        public string Name { get; set; }
```

```
    }
}

using System;

namespace DynamicSearch.ViewModel.Base
{
    public class Feild : IEquatable<Feild>
    {
        public Feild(string title, Type type, string name)
        {
            Title = title;
            Type = type;
            Name = name;
        }

        public Type Type { get; set; }
        public string Name { get; set; }
        public string Title { get; set; }
        public bool Equals(Feild other)
        {
            return other.Title == Title;
        }
    }
}

using System;
using System.Linq.Expressions;

namespace DynamicSearch.ViewModel.Base
{
    public class Operator
    {
        public enum TypesToApply
        {
            String,
            Numeric,
            Both
        }

        public Operator(string title, Func<Expression, Expression, Expression> func, TypesToApply
typeToApply)
        {
            Title = title;
            Func = func;
            TypeToApply = typeToApply;
        }

        public string Title { get; set; }
        public Func<Expression, Expression, Expression> Func { get; set; }
        public TypesToApply TypeToApply { get; set; }
    }
}
```

توسط کلاس زیر یک سری اعمال متداول را پیاده سازی کرده ایم و پیاده سازی اضافات را بعهده کلاس‌های ارث برنده از این کلاس گذاشته ایم:

```
using System.Collections.ObjectModel;
using System.Linq;
using System.Linq.Expressions;

namespace DynamicSearch.ViewModel.Base
{
    public abstract class SearchFilterBase<T> : BaseViewModel
    {
        protected SearchFilterBase()
        {
            var containOp = new Operator("شامل باشد", (expression, expression1) =>
Expression.Call(expression, typeof(string).GetMethod("Contains"), expression1),
Operator.TypesToApply.String);
            var notContainOp = new Operator("شامل نباشد", (expression, expression1) =>
            {
                var contain = Expression.Call(expression, typeof(string).GetMethod("Contains"),
expression1);
                return Expression.Not(contain);
            }, Operator.TypesToApply.String);
```

```
            var equalOp = new Operator("=", Expression.Equal, Operator.TypesToApply.Both);
            var notEqualOp = new Operator("<>", Expression.NotEqual, Operator.TypesToApply.Both);
            var lessThanOp = new Operator("<", Expression.LessThan, Operator.TypesToApply.Numeric);
            var greaterThanOp = new Operator(">", Expression.GreaterThan,
Operator.TypesToApply.Numeric);
            var lessThanOrEqual = new Operator("<=", Expression.LessThanOrEqual,
Operator.TypesToApply.Numeric);
            var greaterThanOrEqual = new Operator(">=", Expression.GreaterThanOrEqual,
Operator.TypesToApply.Numeric);

            Operators = new ObservableCollection<Operator>
                {
                      equalOp,
                      notEqualOp,
                      containOp,
                      notContainOp,
                      lessThanOp,
                      greaterThanOp,
                      lessThanOrEqual,
                      greaterThanOrEqual,
                };


            SelectedAndOr = AndOrs.FirstOrDefault(a => a.Name == "Suppress");
            SelectedFeild = Feilds.FirstOrDefault();
            SelectedOperator = Operators.FirstOrDefault(a => a.Title == "=");
        }

        public abstract IQueryable<T> GetQuarable();

        public virtual ObservableCollection<AndOr> AndOrs
        {
            get
            {
                return new ObservableCollection<AndOr>
                    {
                        new AndOr("And","و", Expression.AndAlso),
                        new AndOr("Or","یا",Expression.OrElse),
                        new AndOr("Suppress","نادیده",(expression, expression1) => expression),
                    };
            }
        }
        public virtual ObservableCollection<Operator> Operators
        {
            get { return _operators; }
            set { _operators = value; NotifyPropertyChanged("Operators"); }
        }
        public abstract ObservableCollection<Feild> Feilds { get; }

        public bool IsOtherFilters
        {
            get { return _isOtherFilters; }
            set { _isOtherFilters = value; }
        }
        public string SearchValue
        {
            get { return _searchValue; }
            set { _searchValue = value; NotifyPropertyChanged("SearchValue"); }
        }
        public AndOr SelectedAndOr
        {
            get { return _selectedAndOr; }
            set { _selectedAndOr = value; NotifyPropertyChanged("SelectedAndOr");
NotifyPropertyChanged("SelectedFeildHasSetted"); }
        }
        public Operator SelectedOperator
        {
            get { return _selectedOperator; }
            set { _selectedOperator = value; NotifyPropertyChanged("SelectedOperator"); }
        }
        public Feild SelectedFeild
        {
            get { return _selectedFeild; }
            set
            {
                Operators = value.Type == typeof(string) ? new
ObservableCollection<Operator>(Operators.Where(a => a.TypeToApply == Operator.TypesToApply.Both ||
a.TypeToApply == Operator.TypesToApply.String)) : new ObservableCollection<Operator>(Operators.Where(a
=> a.TypeToApply == Operator.TypesToApply.Both || a.TypeToApply == Operator.TypesToApply.Numeric));
                if (SelectedOperator == null)
                {
```

```
                    SelectedOperator = Operators.FirstOrDefault(a => a.Title == "=");
                }

                NotifyPropertyChanged("SelectedOperator");
                NotifyPropertyChanged("SelectedFeild");
                _selectedFeild = value;
                NotifyPropertyChanged("SelectedFeildHasSetted");
            }
        }
        public bool SelectedFeildHasSetted
        {
            get
            {
                return SelectedFeild != null &&
                       (SelectedAndOr.Name != "Suppress" || !IsOtherFilters);
            }
        }

        private ObservableCollection<Operator> _operators;
        private Feild _selectedFeild;
        private Operator _selectedOperator;
        private AndOr _selectedAndOr;
        private string _searchValue;
        private bool _isOtherFilters = true;
    }
}
```

توضیحات: در این ویو مدل پایه سه لیست تعریف شده که برای دو تای آنها پیاده سازی پیش فرضی در همین کلاس دیده شده ولی برای لیست فیلدها پیاده سازی به کلاس ارث برنده واگذار شده است.

در گام بعد، یک کلاس کمکی برای سهولت ساخت عبارات ایجاد می‌کنیم:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Reflection;
using AutoMapper;

namespace DynamicSearch.ViewModel.Base
{
  public static  class ExpressionExtensions
    {
        public static List<T> CreateQuery<T>(Expression whereCallExpression, IQueryable entities)
        {
            return entities.Provider.CreateQuery<T>(whereCallExpression).ToList();
        }

        public static MethodCallExpression CreateWhereCall<T>(Expression condition, ParameterExpression
pe, IQueryable entities)
        {
            var whereCallExpression = Expression.Call(
                typeof(Queryable),
                "Where",
                new[] { entities.ElementType },
                entities.Expression,
                Expression.Lambda<Func<T, bool>>(condition, new[] { pe }));
            return whereCallExpression;
        }

        public static void CreateLeftAndRightExpression<T>(string propertyName, Type type, string
searchValue, ParameterExpression pe, out Expression left, out Expression right)
        {
            var typeOfNullable = type;
            typeOfNullable = typeOfNullable.IsNullableType() ? typeOfNullable.GetTypeOfNullable() :
typeOfNullable;
            left = null;

            var typeMethodInfos = typeOfNullable.GetMethods();
            var parseMethodInfo = typeMethodInfos.FirstOrDefault(a => a.Name == "Parse" &&
a.GetParameters().Count() == 1);

            var propertyInfos = typeof(T).GetProperties();
            if (propertyName.Contains("."))
```

```
            {
                left = CreateComplexTypeExpression(propertyName, propertyInfos, pe);
            }
            else
            {
                var propertyInfo = propertyInfos.FirstOrDefault(a => a.Name == propertyName);
                if (propertyInfo != null) left = Expression.Property(pe, propertyInfo);
            }

            if (left != null) left = Expression.Convert(left, typeOfNullable);

            if (parseMethodInfo != null)
            {
                var invoke = parseMethodInfo.Invoke(searchValue, new object[] { searchValue });
                right = Expression.Constant(invoke, typeOfNullable);
            }
            else
            {
                //type is string
                right = Expression.Constant(searchValue.ToLower());
                var methods = typeof(string).GetMethods();
                var firstOrDefault = methods.FirstOrDefault(a => a.Name == "ToLower" &&
!a.GetParameters().Any());
                if (firstOrDefault != null) left = Expression.Call(left, firstOrDefault);
            }
        }

        public static Expression CreateComplexTypeExpression(string searchFilter,
IEnumerable<PropertyInfo> propertyInfos, Expression pe)
        {
            Expression ex = null;
            var infos = searchFilter.Split('.');
            var enumerable = propertyInfos.ToList();
            for (var index = 0; index < infos.Length - 1; index++)
            {
                var propertyInfo = infos[index];
                var nextPropertyInfo = infos[index + 1];
                if (propertyInfos == null) continue;
                var propertyInfo2 = enumerable.FirstOrDefault(a => a.Name == propertyInfo);
                if (propertyInfo2 == null) continue;
                var val = Expression.Property(pe, propertyInfo2);
                var propertyInfos3 = propertyInfo2.PropertyType.GetProperties();
                var propertyInfo3 = propertyInfos3.FirstOrDefault(a => a.Name == nextPropertyInfo);
                if (propertyInfo3 != null) ex = Expression.Property(val, propertyInfo3);
            }

            return ex;
        }

        public static Expression AddOperatorExpression(Func<Expression, Expression, Expression> func,
Expression left, Expression right)
        {
            return func.Invoke(left, right);
        }

        public static Expression JoinExpressions(bool isFirst, Func<Expression, Expression, Expression>
func, Expression expression, Expression ex)
        {
            if (!isFirst)
            {
                return func.Invoke(expression, ex);
            }

            expression = ex;
            return expression;
        }
    }
}
```

-5 ایجاد کلاس فیلتر جهت معرفی فیلدها و معرفی منبع داده و ویو مدلی ارث برنده از کلاس‌های پایه ساختار، جهت ایجاد فرم نمونه:

```
using System.Collections.ObjectModel;
using System.Linq;
using DynamicSearch.Model;
using DynamicSearch.Service;
```

```
using DynamicSearch.ViewModel.Base;

namespace DynamicSearch.ViewModel
{
    public class StudentSearchFilter : SearchFilterBase<Student>
    {
        public override ObservableCollection<Feild> Feilds
        {
            get
            {
                return new ObservableCollection<Feild>
                    {
                        new Feild("نام دانش آموز",typeof(string),"Name"),
                         new Feild("نام خانوادگی دانش آموز",typeof(string),"Family"),
                        new Feild("نام خانوادگی معلم",typeof(string),"Teacher.Name"),
                        new Feild("شماره دانش آموزی",typeof(int),"StdID"),
                    };
            }
        }

        public override IQueryable<Student> GetQuarable()
        {
            return new StudentService().GetStudents().AsQueryable();
        }
    }
}
```

-6 ایجاد ویو نمونه:

در نهایت زمل فایل موجود در پروژه ویو:

```
<Window x:Class="DynamicSearch.View.MainWindow"
             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"
        xmlns:viewModel="clr-namespace:DynamicSearch.ViewModel;assembly=DynamicSearch.ViewModel"
        xmlns:view="clr-namespace:DynamicSearch.View"
        mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
    <Window.Resources>
        <viewModel:StudentSearchViewModel x:Key="StudentSearchViewModel" />
        <view:VisibilityConverter x:Key="VisibilityConverter" />
    </Window.Resources>
    <Grid   DataContext="{StaticResource StudentSearchViewModel}">
        <WrapPanel Orientation="Vertical">
            <DataGrid AutoGenerateColumns="False" Name="asd" CanUserAddRows="False"
ItemsSource="{Binding BindFilter}">
                <DataGrid.Columns>
                    <DataGridTemplateColumn>
                        <DataGridTemplateColumn.CellTemplate>
                            <DataTemplate DataType="{x:Type viewModel:StudentSearchFilter}">
                                <ComboBox MinWidth="100"  DisplayMemberPath="Title"
ItemsSource="{Binding AndOrs}" Visibility="{Binding IsOtherFilters,Converter={StaticResource
VisibilityConverter}}"
                                          SelectedItem="{Binding
SelectedAndOr,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}" />
                            </DataTemplate>
                        </DataGridTemplateColumn.CellTemplate>
                    </DataGridTemplateColumn>
                    <DataGridTemplateColumn >
                        <DataGridTemplateColumn.CellTemplate>
                            <DataTemplate DataType="{x:Type viewModel:StudentSearchFilter}">
                                <ComboBox IsEnabled="{Binding SelectedFeildHasSetted}" MinWidth="100"
DisplayMemberPath="Title" ItemsSource="{Binding Feilds}" SelectedItem="{Binding
SelectedFeild,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged }"/>
                            </DataTemplate>
                        </DataGridTemplateColumn.CellTemplate>
                    </DataGridTemplateColumn>
                    <DataGridTemplateColumn>
                        <DataGridTemplateColumn.CellTemplate>
                            <DataTemplate DataType="{x:Type viewModel:StudentSearchFilter}">
                                <ComboBox MinWidth="100"  DisplayMemberPath="Title"
ItemsSource="{Binding Operators}" IsEnabled="{Binding SelectedFeildHasSetted}"
                                          SelectedItem="{Binding
```

```
SelectedOperator,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}" />
                            </DataTemplate>
                        </DataGridTemplateColumn.CellTemplate>
                    </DataGridTemplateColumn>
                    <DataGridTemplateColumn Width="*">
                        <DataGridTemplateColumn.CellTemplate>
                            <DataTemplate DataType="{x:Type viewModel:StudentSearchFilter}">
                                <TextBox IsEnabled="{Binding SelectedFeildHasSetted}" MinWidth="200"
Text="{Binding SearchValue,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>
                                <!--<TextBox Text="{Binding
SearchValue,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>-->
                            </DataTemplate>
                        </DataGridTemplateColumn.CellTemplate>
                    </DataGridTemplateColumn>
                </DataGrid.Columns>
            </DataGrid>

            <Button Content="+" HorizontalAlignment="Left" Command="{Binding AddFilter}"/>
            <Button Content="Result" Command="{Binding ExecuteSearchFilter}"/>
            <DataGrid ItemsSource="{Binding Results}">

            </DataGrid>
        </WrapPanel>
    </Grid>
</Window>
```

در این مقاله، هدف معرفی روند ایجاد یک جستجو گر پویا با قابلیت استفاده مجدد بالا بود و عمدا از توضیح جزء به جزء کدها صرف نظر شده. علت این امر وجود منابع بسیار راجب ابزارهای بکار رفته در این مقاله و سادگی کدهای نوشته شده توسط اینجانب می‌باشد.

برخی منابع جهت آشنایی با Expression ها:

http://msdn.microsoft.com/en-us/library/bb882637.aspx

انتخاب پویای فیلدها در LINQ

http://www.persiadevelopers.com/articles/dynamiclinqquery.aspx

نکته: کدهای نوشته شده در این مقاله، نسخه‌های نخستین هستند و طبیعتا جا برای بهبود بسیار دارند. دوستان می‌توانند در این امر به بنده کمک کنند.

پیشنهادات جهت بهبود:
- جداسازی کدهای پیاده کننده منطق از ویو مدل‌ها جهت افزایش قابلیت نگهداری کد و سهولت استفاده در سایر ساختارها
- افزودن توضیحات به کد
- انتخاب نامگذاری‌های مناسب تر

DynamicSearch.zip