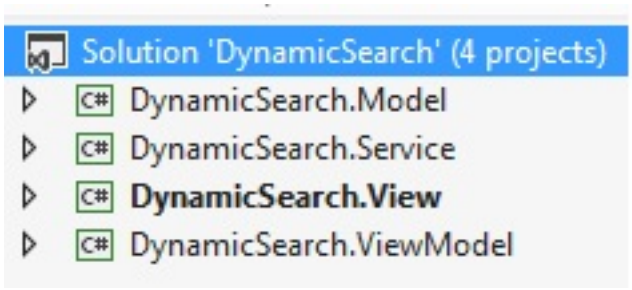
عنوان: **ایجاد فرم جستجوی پویا با استفاده از Expression ها**
نویسنده: محمد عیدی مراد
تاریخ: ۱۳۹۲/۰۳/۲۵ ۹:۲۵
آدرس: www.dotnettips.info
برچسب‌ها: LINQ, MVVM, Expression, DynamicLINQ

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

عنوان: **استفاده از Lambda Expression در پروژه های مبتنی بر WCF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۳/۳۰ ۱۹:۱۰
آدرس: www.dotnettips.info
برچسب‌ها: WCF, Serialization, Expression, Delegate

نکته : آشنایی با مفاهیم پایه WCF برای فهم بهتر مفاهیم توصیه می‌شود.

امروزه استفاده از WCF در پروژه‌های SOA بسیار فراگیر شده است. کمتر کسی است که در مورد قدرت تکنولوژی WCF نشنیده باشد یا از این تکنولوژی در پروژه‌های خود استفاده نکرده باشد. WCF مدل برنامه نویسی یکپارچه مایکروسافت برای ساخت نرم‌افزارهای سرویس گرا است و برای توسعه دهندگان امکانی را فراهم می‌کند که راهکارهایی امن، و مبتنی بر تراکنش را تولید نمایند که قابلیت استفاده در بین پلتفرم‌های مختلف را دارند. قبل از WCF توسعه دهندگان پروژه‌های نرم افزاری برای تولید پروژه‌های توزیع شده باید شرایط موجود برای تولید و توسعه را در نظر می‌گرفتند. برای مثال اگر استفاده کننده از سرویس در داخل سازمان و بر پایه دات نت تهیه شده بود از .net remoting استفاده می‌کردند و اگر استفاده کننده سرویس از خارج سازمان یا مثلا بر پایه تکنولوژی J2EE بود از Web Serviceها استفاده می‌شد. با ظهور WCF این تکنولوژی با هم تجمیع شدند(بهتر بگم تبدیل به یک تکنولوژی واحد شدند) و دیگر خبری از net remoting یا web serviceها نیست.

WCF با تمام قدرت و امکاناتی که داراست دارای نقاط ضعفی هم می‌باشد که البته این معایب (یا محدودیت) بیشتر جهت سازگار سازی سرویس‌های نوشته شده با سیستم‌ها و پروتکل‌های مختلف است.
برای انتقال داده‌ها از طریق WCF بین سیستم‌های مختلف باید داده‌های مورد نظر حتما سریالایز شوند که مثال هایی از این دست رو در همین سایت می‌تونید مطالعه کنید:
(^) و (^) و (^)

با توجه به این که داده‌ها سریالایز می‌شوند، در نتیجه امکان انتقال داده هایی که از نوع object هستند در WCF وجود ندارد. بلکه نوع داده باید صراحتا ذکر شود و این نوع باید قابلیت سریالایز شدن را دارا باشد.برای مثال شما نمی‌تونید متدی داشته باشید که پارامتر ورودی آن از نوع delegate باشد یا کلاسی باشد که صفت [Serializable] در بالای اون قرار نداشته باشد یا کلاسی باشد که صفت DataContract برای خود کلاس و صفت DataMember برای خاصیت‌های اون تعریف نشده باشد. حالا سوال مهم این است اگر متدی داشته باشیم که پارامتر ورودی آن حتما باید از نوع delegate باشد چه باید کرد؟

برای تشریح بهتر مسئله یک مثال می‌زنم؟

سرویسی داریم برای اطلاعات کتاب ها. قصد داریم متدی بنوسیم که پارامتر ورودی آن از نوع Lambda Expression است تا Query مورد نظر کاربر از سمت کلاینت به سمت سرور دریافت کند و خروجی مورد نظر را با توجه به Query ورودی به کلاینت برگشت دهد.(متدی متداول در اکثر پروژه ها). به صورت زیر عمل می‌کنیم.

*ابتدا یک Blank Solution ایجاد کنید.

*یک ClassLibrary به نام Model ایجاد کنید و کلاسی به نام Book در آن بسازید .(همانطور که میبینید کلاس مورد نظر سریالایز شده است):

```
[DataContract]
    public class Book
    {
        [DataMember]
        public int Code { get; set; }

        [DataMember]
        public string Title { get; set; }
    }
```

* یک WCF Service Application ایجاد کنید

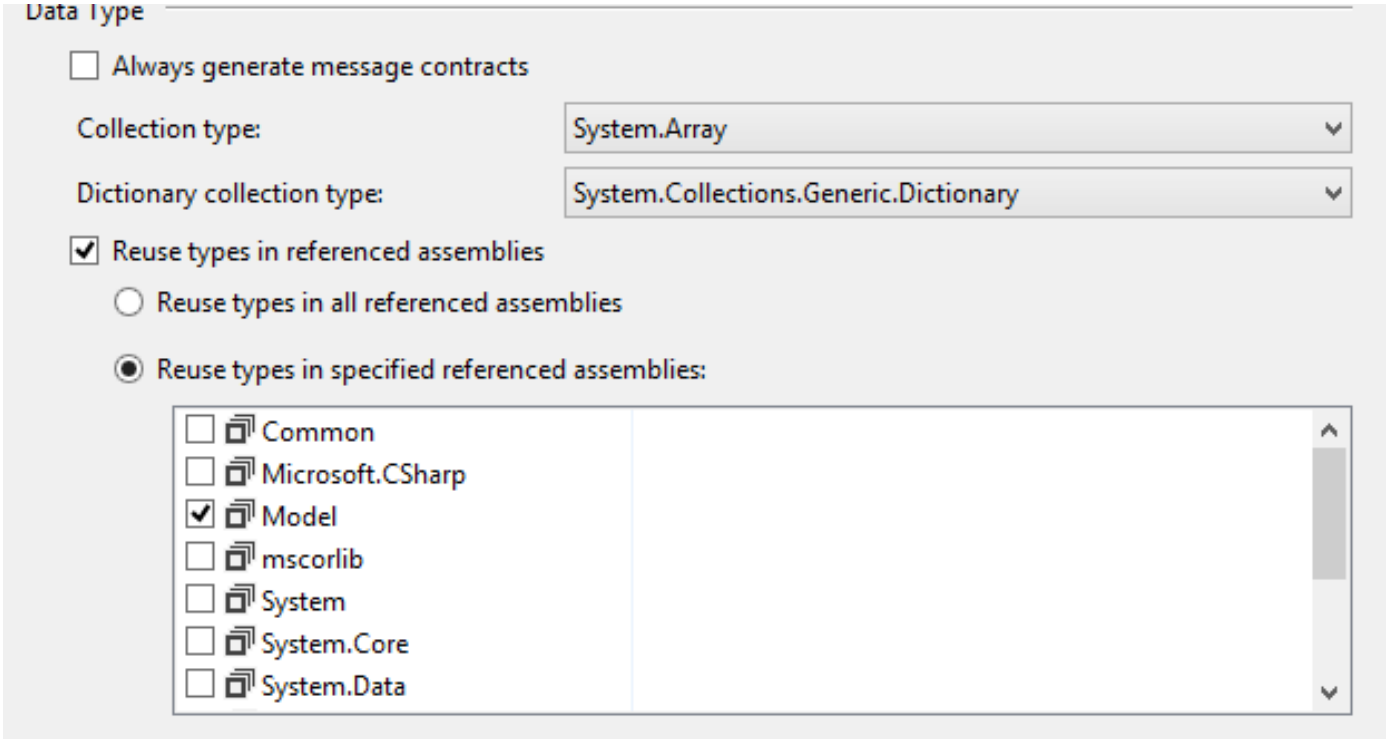
یک Contract برای ارتباط بین سرور و کلاینت می‌سازیم:

```
using System;
using System.Collections.Generic;
using System.Linq.Expressions;
using System.ServiceModel;

namespace WcfLambdaExpression
{
    [ServiceContract]
    public interface IBookService
    {
        [OperationContract]
        IEnumerable<Book> GetByExpression( Expression<Func<Book, bool>> expression );
    }
}
```

متد GetByExpression دارای پارامتر ورودی expression است که نوع آن نیز Lambda Expression می‌باشد. حال یک سرویس ایجاد می‌کنیم:

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

namespace WcfLambdaExpression
{
    public class BookService : IBookService
    {
        public BookService()
        {
            ListOfBook = new List<Book>();
        }

        public List<Book> ListOfBook
        {
            get;
            private set;
        }

        public IEnumerable<Book> GetByExpression( Expression<Func<Book, bool>> expression )
        {
            ListOfBook.AddRange( new Book[]
            {
                new Book(){Code = 1 , Title = "Book1"},
                new Book(){Code = 2 , Title = "Book2"},
                new Book(){Code = 3 , Title = "Book3"},
                new Book(){Code = 4 , Title = "Book4"},
                new Book(){Code = 5 , Title = "Book5"},
            } );

            return ListOfBook.AsQueryable().Where( expression );
        }
    }
}
```

بعد از Build پروژه همه چیز سمت سرور آماده است. یک پروژه دیگر از نوع Console ایجاد کنید و از روش AddServiceReference سعی کنید که سرویس مورد نظر را به پروژه اضافه کنید. در هنگام Add Service Reference برای اینکه سرویس سمت سرور و کلاینت هر دو با یک مدل کار کنند باید از یک Reference assembly استفاده کنند و کافی است از قسمت Advanced گزینه Reuse types in referenced assemblies را تیک بزنید و assemblyهای مورد نظر را انتخاب کنید.(در این پروژه باید Model و System.Xml.Linq را انتخاب کنید)



به طور حتم با خطا روبرو خواهید شد. دلیل آن هم این است که امکان سریالایز کردن برای پارامتر ورودی expression میسر نیست.

خطای مربوطه به شکل زیر خواهد بود:

```
Type 'System.Linq.Expressions.Expression`1[System.Func`2[WcfLambdaExpression.Book,System.Boolean]]'
cannot be serialized.
Consider marking it with the DataContractAttribute attribute, and marking all of its members you want
serialized with the DataMemberAttribute attribute.
If the type is a collection, consider marking it with the CollectionDataContractAttribute.
See the Microsoft .NET Framework documentation for other supported types
```

حال چه باید کرد؟

روش‌های زیادی برای بر طرف کردن این محدودیت وجود دارد. اما در این پست روشی رو که خودم از اون استفاده می‌کنم رو براتون شرح می‌دهم.

در این روش باید از XElement استفاده شود که در فضای نام System.Linq.Xml قرار دارد. یعنی آرگومان ورودی سمت کلاینت باید به فرمت Xml سریالایز شود و سمت سرور دوباره دی سریالایز شده و تبدیل به یک Lambda Expression شود. اما سریالایز کردن Lambda Expression واقعا کاری سخت و طاقت فرساست . با توجه به این که در اکثر پروژه‌ها این متدها به صورت Generic نوشته می‌شوند. برای حل این مسئله بعد از مدتی جستجو، کلاسی رو پیدا کردم که این کار رو برام انجام می‌داد. بعد از مطالعه دقیق و مشاهده روش کار کلاس، تغییرات مورد نظرم رو اعمال کردم و الان در اکثر پروژه هام دارم از این کلاس استفاده می‌کنم.

یک مثال از روش استفاده :

برای اینکه از این کلاس در هر دو پروژه (سرور و کلاینت) استفاده می‌کنیم باید یک Class Library جدید به نام Common بسازید و یک ارجاع از اون رو به هر دو پروژه سمت سرور و کلاینت بدید.

سرویس و Contract بالا رو به صورت زیر باز نویسی کنید.

```
[ServiceContract]
    public interface IBookService
    {
        [OperationContract]
        IEnumerable<Book> GetByExpression( XElement expression );
    }
```

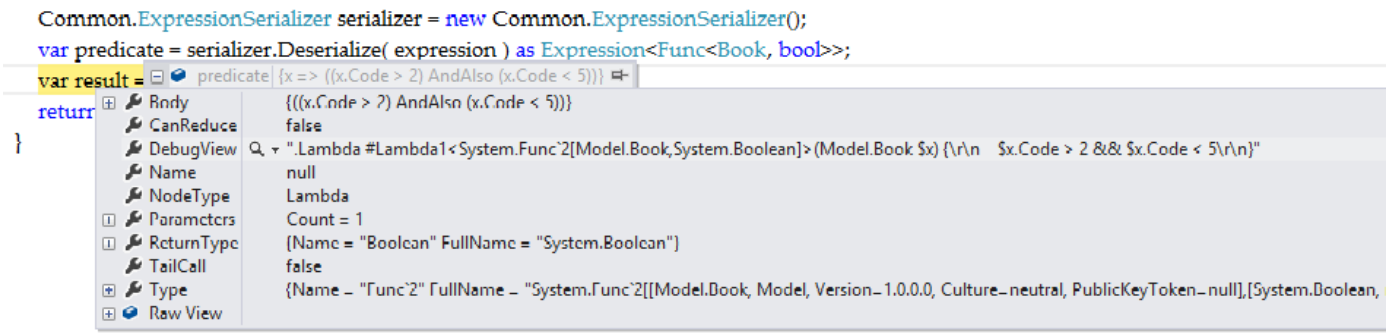
و سرویس :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Xml.Linq;

namespace WcfLambdaExpression
{
    public class BookService : IBookService
    {
        public BookService()
        {
            ListOfBook = new List<Book>();
        }

        public List<Book> ListOfBook
        {
            get;
            private set;
        }

        public IEnumerable<Book> GetByExpression( XElement expression )
        {
            ListOfBook.AddRange( new Book[]
            {
                new Book(){Code = 1 , Title = "Book1"},
                new Book(){Code = 2 , Title = "Book2"},
                new Book(){Code = 3 , Title = "Book3"},
                new Book(){Code = 4 , Title = "Book4"},
                new Book(){Code = 5 , Title = "Book5"},
            } );

             Common.ExpressionSerializer serializer = new Common.ExpressionSerializer();

            return ListOfBook.AsQueryable().Where( serializer.Deserialize( expression ) as
Expression<Func<Book, bool>> );
        }
    }
```

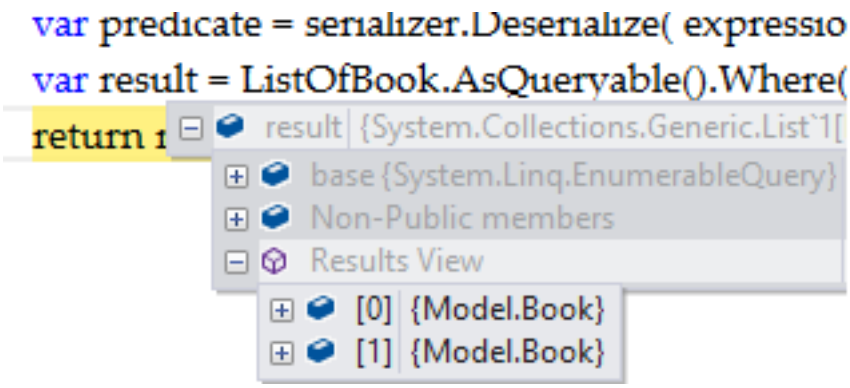
بعد از Build پروژه از روش Add Service Reference استفاده کنید و می‌بینید که بدون هیچ گونه مشکلی سرویس مورد نظر به پروژه Console اضافه شد. برای استفاده سمت کلاینت به صورت زیر عمل کنید.

```
using System;
using System.Linq.Expressions;
using TestExpression.MyBookService;

namespace TestExpression
{
    class Program
    {
        static void Main( string[] args )
        {
            BookServiceClient bookService = new BookServiceClient();

            Expression<Func<Book, bool>> expression = x => x.Code > 2 && x.Code < 5;

            Common.ExpressionSerializer serializer = new Common.ExpressionSerializer();

            bookService.GetByExpression( serializer.Serialize( expression ) );
        }
    }
}
```

بعد از اجرای پروژه، در سمت سرور خروجی‌های زیر رو مشاهده می‌کنیم.



خروجی هم به صورت زیر خواهد بود:



دریافت سورس کامل Expression-Serialization

نظرات خوانندگان

نویسنده: سایلنت
تاریخ: ۱۳۹۲/۰۸/۰۲ ۱۵:۱۱

بسیار عالیه . تازه شروع کردم به یادگیری WCF از مقالات شما نهایت استفاده رو بردم .

نویسنده: محمد
تاریخ: ۱۳۹۲/۰۹/۱۹ ۱۷:۱۶

سلام و ممنون از مقاله خوبتون، اما متاسفانه کلاس شما رو نمیشه برای JSON استفاده نمود.

```
string json = JsonConvert.SerializeObject(serializer.Serialize(predicate3));
predicate3 = JsonConvert.DeserializeObject<Expression<Func<Entity, bool>>>(json);
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۱۹ ۲۲:۵۸

- اینکار اضافی است. چون xml را تبدیل به json می‌کنید؛ بعد json را تبدیل به xml.

+ خروجی serializer.Serialize از نوع XElement است. بنابراین در قسمت آرگومان جنریک JsonConvert.DeserializeObject باید XElement ذکر شود. مرحله بعدی آن فراخوانی serializer.Deserialize روی این خروجی است.

```
Expression<Func<Book, bool>> expression = x => x.Code > 2 && x.Code < 5;
var expressionSerializer = new Common.ExpressionSerializer();
var xml = expressionSerializer.Serialize(expression);
var xmlToJson = JsonConvert.SerializeObject(xml);
var xmlObject = JsonConvert.DeserializeObject<XElement>(xmlToJson);
var exp2 = expressionSerializer.Deserialize(xmlObject) as Expression<Func<Book, bool>>;
```