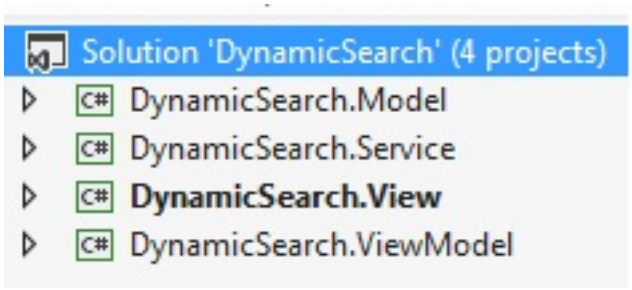
عنوان: **ایجاد فرم جستجوی پویا با استفاده از Expression ها**
نویسنده: محمد عیدی مراد
تاریخ: ۱۳۹۲/۰۳/۲۵ ۹:۲۵
آدرس: www.dotnettips.info
برچسب‌ها: LINQ, MVVM, Expression, DynamicLINQ

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

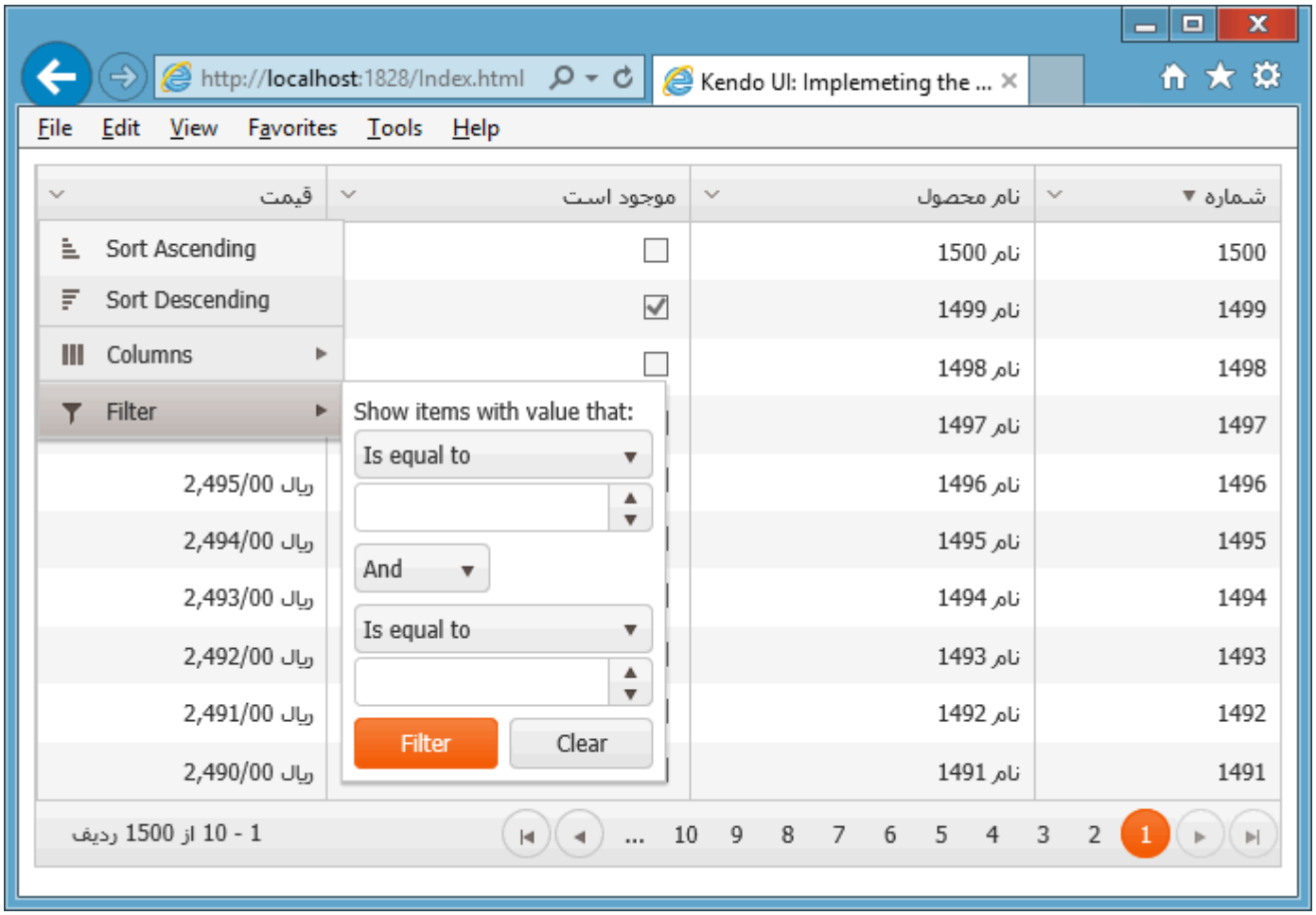
نکته: کدهای نوشته شده در این مقاله، نسخه‌های نخستین هستند و طبیعتا جا برای بهبود بسیار دارند. دوستان می‌توانند در این امر به بنده کمک کنند.

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: **صفحه بندی، مرتب سازی و جستجوی پویای اطلاعات به کمک Kendo UI Grid**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۱۶ ۱۶:۳۰
آدرس: www.dotnettips.info
گروه‌ها: JavaScript, ASP.NET Web API, jQuery, DynamicLINQ, Kendo UI, json.net

پس از آشنایی مقدماتی با Kendo UI DataSource ، اکنون می‌خواهیم از آن جهت صفحه بندی، مرتب سازی و جستجوی پویای سمت سرور استفاده کنیم. در مثال قبلی، هر چند صفحه بندی فعال بود، اما پس از دریافت تمام اطلاعات، این اعمال در سمت کاربر انجام و مدیریت می‌شد.



مدل برنامه

در اینجا قصد داریم لیستی را با ساختار کلاس Product در اختیار Kendo UI گرید قرار دهیم:

```
namespace KendoUI03.Models
{
    public class Product
    {
        public int Id { set; get; }
        public string Name { set; get; }
        public decimal Price { set; get; }
        public bool IsAvailable { set; get; }
    }
}
```

پیشنیاز تامین داده مخصوص Kendo UI Grid

برای ارائه اطلاعات مخصوص Kendo UI Grid، ابتدا باید درنظر داشت که این گرید، درخواست‌های صفحه بندی خود را با فرمت ذیل ارسال می‌کند. همانطور که مشاهده می‌کنید، صرفا یک کوئری استرینگ با فرمت JSON را دریافت خواهیم کرد:

```
/api/products?{"take":10,"skip":0,"page":1,"pageSize":10,"sort":[{"field":"Id","dir":"desc"}]}
```

سپس این گرید نیاز به سه فیلد، در خروجی JSON نهایی خواهد داشت:

```
{
"Data":
[
{"Id":1500,"Name":"1500 نام","Price":2499.0,"IsAvailable":false},
{"Id":1499,"Name":"1499 نام","Price":2498.0,"IsAvailable":true}
],
"Total":1500,
"Aggregates":null
}
```

فیلد Data که رکوردهای گرید را تامین می‌کنند. فیلد Total که بیانگر تعداد کل رکوردها است و Aggregates که برای گروه بندی بکار می‌رود.

می‌توان برای تمام این‌ها، کلاس و Parser تهیه کرد و یا ... پروژه‌ی سورس بازی به نام Kendo.DynamicLinq نیز چنین کاری را میسر می‌سازد که در ادامه از آن استفاده خواهیم کرد. برای نصب آن تنها کافی است دستور ذیل را صادر کنید:

```
PM> Install-Package Kendo.DynamicLinq
```

Kendo.DynamicLinq به صورت خودکار System.Linq.Dynamic را نیز نصب می‌کند که از آن جهت صفحه بندی پویا استفاده خواهد شد.

تامین کننده‌ی داده سمت سرور

همانند مطلب کار با Kendo UI DataSource ، یک ASP.NET Web API Controller جدید را به پروژه اضافه کنید و همچنین مسیریابی‌های مخصوص آن‌را به فایل global.asax.cs نیز اضافه نمائید.

```
using System.Linq;
using System.Net.Http;
using System.Web.Http;
using Kendo.DynamicLinq;
using KendoUI03.Models;
using Newtonsoft.Json;

namespace KendoUI03.Controllers
{
    public class ProductsController : ApiController
    {
        public DataSourceResult Get(HttpRequestMessage requestMessage)
        {
            var request = JsonConvert.DeserializeObject<DataSourceRequest>(
                requestMessage.RequestUri.ParseQueryString().GetKey(0)
            );

            var list = ProductDataSource.LatestProducts;
            return list.AsQueryable()
                       .ToDataSourceResult(request.Take, request.Skip, request.Sort, request.Filter);
        }
    }
}
```

تمام کدهای این کنترلر همین چند سطر فوق هستند. با توجه به ساختار کوئری استرینگی که در ابتدای بحث عنوان شد، نیاز است آن‌را توسط کتابخانه‌ی JSON.NET تبدیل به یک نمونه از DataSourceRequest نمائیم. این کلاس در Kendo.DynamicLinq تعریف شده‌است و حاوی اطلاعاتی مانند take و skip کوئری LINQ نهایی است.
ProductDataSource.LatestProducts صرفا یک لیست جنریک تهیه شده از کلاس Product است. در نهایت با استفاده از متد الحاقی جدید ToDataSourceResult ، به صورت خودکار مباحث صفحه بندی سمت سرور به همراه مرتب سازی اطلاعات، صورت گرفته و اطلاعات نهایی با فرمت DataSourceResult بازگشت داده می‌شود. DataSourceResult نیز در Kendo.DynamicLinq تعریف شده و سه فیلد یاد شده‌ی Data، Total و Aggregates را تولید می‌کند.

تا اینجا کارهای سمت سرور این مثال به پایان می‌رسد.

تهیه View نمایش اطلاعات ارسالی از سمت سرور

اعمال مباحث بومی سازی

```
<head>
    <meta charset="utf-8" />
    <meta http-equiv="Content-Language" content="fa" />
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

    <title>Kendo UI: Implemeting the Grid</title>

    <link href="styles/kendo.common.min.css" rel="stylesheet" type="text/css" />
    <!--شیوه نامه‌ی مخصوص راست به چپ سازی-->
    <link href="styles/kendo.rtl.min.css" rel="stylesheet" />
    <link href="styles/kendo.default.min.css" rel="stylesheet" type="text/css" />
    <script src="js/jquery.min.js" type="text/javascript"></script>
    <script src="js/kendo.all.min.js" type="text/javascript"></script>

    <!--محل سفارشی سازی پیام‌ها و مسایل بومی-->
    <script src="js/cultures/kendo.culture.fa-IR.js" type="text/javascript"></script>
    <script src="js/cultures/kendo.culture.fa.js" type="text/javascript"></script>
    <script src="js/messages/kendo.messages.en-US.js" type="text/javascript"></script>

    <style type="text/css">
        body {
            font-family: tahoma;
            font-size: 9pt;
        }
    </style>

    <script type="text/javascript">
        // جهت استفاده از فایل: kendo.culture.fa-IR.js
        kendo.culture("fa-IR");
    </script>
</head>
```

- در اینجا چند فایل js و css جدید اضافه شده‌اند. فایل kendo.rtl.min.css جهت تامین مباحث RTL توکار Kendo UI کاربرد دارد.
- سپس سه فایل kendo.culture.fa.js، kendo.culture.fa-IR.js و kendo.messages.en-US.js نیز اضافه شده‌اند. فایل‌های fa و fa-Ir آن هر چند به ظاهر برای ایران طراحی شده‌اند، اما نام ماه‌های موجود در آن عربی است که نیاز به ویرایش دارد. به همین جهت به سورس این فایل‌ها، جهت ویرایش نهایی نیاز خواهد بود که در پوشه‌ی src\js\cultures مجموعه‌ی اصلی Kendo UI موجود هستند (ر.ک. فایل پیوست).
- فایل kendo.messages.en-US.js حاوی تمام پیام‌های مرتبط با Kendo UI است. برای مثال«رکوردهای 10 تا 15 از 1000 ردیف» را در اینجا می‌توانید به فارسی ترجمه کنید.
- متد kendo.culture کار مشخص سازی فرهنگ بومی برنامه را به عهده دارد. برای مثال در اینجا به fa-IR تنظیم شده‌است. این مورد سبب خواهد شد تا از فایل kendo.culture.fa-IR.js استفاده گردد. اگر مقدار آن‌را به fa تنظیم کنید، از فایل kendo.culture.fa.js کمک گرفته خواهد شد.

راست به چپ سازی گرید

تنها کاری که برای راست به چپ سازی Kendo UI Grid باید صورت گیرد، محصور سازی div آن در یک div با کلاس مساوی k-

rtl است:

```
<div class="k-rtl">
        <div id="report-grid"></div>
    </div>
```

k-rtl و تنظیمات آن در فایل kendo.rtl.min.css قرار دارند که در ابتدای head صفحه تعریف شده‌است.

تامین داده و نمایش گرید

در ادامه کدهای کامل DataSource و Kendo UI Grid را ملاحظه می‌کنید:

```
<script type="text/javascript">
        $(function () {
            var productsDataSource = new kendo.data.DataSource({
                transport: {
                    read: {
                        url: "api/products",
                        dataType: "json",
                        contentType: 'application/json; charset=utf-8',
                        type: 'GET'
                    },
                    parameterMap: function (options) {
                        return kendo.stringify(options);
                    }
                },
                schema: {
                    data: "Data",
                    total: "Total",
                    model: {
                        fields: {
                            "Id": { type: "number" }, //تعیین نوع فیلد برای جستجوی پویا مهم است
                            "Name": { type: "string" },
                            "IsAvailable": { type: "boolean" },
                            "Price": { type: "number" }
                        }
                    }
                },
                error: function (e) {
                    alert(e.errorThrown);
                },
                pageSize: 10,
                sort: { field: "Id", dir: "desc" },
                serverPaging: true,
                serverFiltering: true,
                serverSorting: true
            });

            $("#report-grid").kendoGrid({
                dataSource: productsDataSource,
                autoBind: true,
                scrollable: false,
                pageable: true,
                sortable: true,
                filterable: true,
                reorderable: true,
                columnMenu: true,
                columns: [
                    { field: "Id", title: "شماره", width: "130px" },
                    { field: "Name", title: "نام محصول" },
                    {
                        field: "IsAvailable", title: "موجود است",
                        template: '<input type="checkbox" #= IsAvailable ? checked="checked" : "" #
disabled="disabled" ></input>'
                    },
                    { field: "Price", title: "قیمت", format: "{0:c}" }
                ]
            });
        });
    </script>
```

- با تعاریف مقدماتی Kendo UI DataSource پیشتر آشنا شده‌ایم و قسمت read آن جهت دریافت اطلاعات از سمت سرور کاربرد

دارد.
- در اینجا ذکر contentType الزامی است. زیرا ASP.NET Web API بر این اساس است که تصمیم می‌گیرد، خروجی را به صورت JSON ارائه دهد یا XML.
- با استفاده از parameterMap، سبب خواهیم شد تا پارامترهای ارسالی به سرور، با فرمت صحیحی تبدیل به JSON شده و بدون مشکل به سرور ارسال گردند.
- در قسمت schema باید نام فیلدهای موجود در DataSourceResult دقیقا مشخص شوند تا گرید بداند که data را باید از چه فیلدی استخراج کند و تعداد کل ردیف‌ها در کدام فیلد قرار گرفته‌است.
- نحوه‌ی تعریف model را نیز در اینجا ملاحظه می‌کنید. ذکر نوع فیلدها در اینجا بسیار مهم است و اگر قید نشوند، در حین جستجوی پویا به مشکل برخواهیم خورد. زیرا پیش فرض نوع تمام فیلدها string است و در این حالت نمی‌توان عدد 1 رشته‌ای را با یک فیلد از نوع int در سمت سرور مقایسه کرد.
- در اینجا serverFiltering، serverPaging و serverSorting نیز به true تنظیم شده‌اند. اگر این مقدار دهی‌ها صورت نگیرد، این اعمال در سمت کلاینت انجام خواهند شد.

پس از تعریف DataSource، تنها کافی است آن‌را به خاصیت dataSource یک kendoGrid نسبت دهیم.
- autoBind: true سبب می‌شود تا اطلاعات DataSource بدون نیاز به فراخوانی متد read آن به صورت خودکار دریافت شوند.
- با تنظیم scrollable: false، اعلام می‌کنیم که قرار است تمام رکوردها در معرض دید قرارگیرند و اسکرول پیدا نکنند.
- pageable: true صفحه بندی را فعال می‌کند. این مورد نیاز به تنظیم pageSize: 10 در قسمت DataSource نیز دارد.
- با sortable: true مرتب سازی ستون‌ها با کلیک بر روی سرستون‌ها فعال می‌گردد.
- filterable: true به معنای فعال شدن جستجوی خودکار بر روی فیلدها است. کتابخانه‌ی Kendo.DynamicLinq حاصل آن‌را در سمت سرور مدیریت می‌کند.
- reorderable: true سبب می‌شود تا کاربر بتواند محل قرارگیری ستون‌ها را تغییر دهد.
- ذکر columnMenu: true اختیاری است. اگر ذکر شود، امکان مخفی سازی انتخابی ستون‌ها نیز مسیر خواهد شد.
- در آخر ستون‌های گرید مشخص شده‌اند. با تعیین format: "{0:c}" سبب نمایش فیلدهای قیمت با سه رقم جدا کننده خواهیم شد. مقدار ریال آن از فایل فرهنگ جاری تنظیم شده دریافت می‌گردد. با استفاده از template تعریف شده نیز سبب نمایش فیلد bool به صورت یک checkbox خواهیم شد.

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید:

KendoUI03.zip

نظرات خوانندگان

نویسنده: حمیدرضا کبیری
تاریخ: ۱۳۹۳/۰۸/۱۶ ۱۹:۳

آیا kendo UI کاملا از زبان فارسی پشتیبانی میکند ؟
برای calender آن ، به تقویم شمی گزینه ای موجود هست ؟
این گزینه با ورژن ۲۰۱۴/۱/۳۱۸ مطابقت دارد ، آیا با ورژنهای جدید مشکلی نخواهد داشت ؟

نویسنده: احمد رجبی
تاریخ: ۱۳۹۳/۰۸/۱۶ ۲۰:۱۵

میتوانید با اضافه کردن این اسکریپت تمامی قسمتهای kendo را به زبان فارسی ترجمه کنید.

نویسنده: سعیدجلالی
تاریخ: ۱۳۹۳/۰۸/۱۷ ۸:۴۴

با تشکر از مطلب مفید شما من از wrapper mvc مجموعه kendo استفاده میکنم
توی مطالب شما در مورد استفاده از Kendo.DynamicLinq صحبت شد خواستم بدونم آیا وقتی از wrapper هم استفاده میکنیم استفاده از این پکیج لازم هست؟

```
@(Html.Kendo().Grid(Model)
      .Name("gridKendo")
      .Columns(columns =>
      {
          columns.Bound(c => c.Name).Width(50);
          columns.Bound(c => c.Family).Width(100);
          columns.Bound(c => c.Tel);
      })
      .Pageable(pager => pager
          .Input(true)
          .Numeric(true)
          .Info(true)
          .PreviousNext(true)
          .Refresh(true)
          .PageSizes(true)
      )
      )
```

چون من با استفاده از telerik profiler وقتی درخواست رو بررسی مکنم توی دستور sql چنین دستوری رو در انتها مشاهده میکنم:
صفحه اول:

```
SELECT *
FROM (
    SELECT
    FROM table a
)
WHERE ROWNUM <= :TAKE
```

صفحات بعد:

```
SELECT *
FROM (
    SELECT
        a.*,
        ROWNUM OA_ROWNUM
    FROM (
        FROM table a
```

```
    ) a
    WHERE ROWNUM <= :TAKE
)
WHERE OA_ROWNUM > :SKIP
```

پایگاه داده اوراکل است.

نویسنده: سعیدجلالی
تاریخ: ۱۳۹۳/۰۸/۱۷ ۹:۱۲

امکان فارسی شدن تمام بخش‌ها وجود دارد.
تقویم هم فارسی شده است در این سایت برای نسخه‌های جدیدتر هم باید دوتا فایل جاوا اسکریپت all و mvc رو خودتون تغییر بدهید (با توجه به الگوی انجام شده در فایل فارسی شده فوق)
ولی برای تقویم زمانبدی scheduler من فارسی ندیده ام

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۱۷ ۹:۳۱

مطلب فوق نه وابستگی خاصی به وب فرم‌ها دارد و نه ASP.NET MVC. ویو آن یک فایل HTML ساده‌است و سمت سرور آن فقط یک کنترلر ASP.NET Web API که با تمام مشتقات ASP.NET سازگار است. در این حالت یک نفر می‌تواند ASP.NET نگارش خودش را خلق کند؛ بدون اینکه نگران جزئیات وب فرم‌ها باشد یا ASP.NET MVC. ضمنا دانش جاوا اسکریپتی آن هم قابل انتقال است؛ چون اساسا Kendo UI برای فناوری سمت سرور خاصی طراحی نشده‌است و حالت اصل آن با PHP، Java و امثال آن هم کار می‌کند.

نویسنده: میثم آقااحمدی
تاریخ: ۱۳۹۳/۰۸/۱۷ ۱۳:۱۷

در کنترلر این خط باعث بارگذاری تمامی داده‌ها می‌شود

```
var list = ProductDataSource.LatestProducts;
```

آیا راه حلی وجود دارد که دیتای به تعداد همان pagesize از پایگاه خوانده شود؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۱۷ ۱۳:۲۸

- این فقط یک مثال هست و منبع داده‌ای صرفا جهت دموی ساده‌ی برنامه. فقط برای اینکه با یک کلیک بتوانید برنامه را اجرا کنید و نیازی به برپایی و تنظیم بانک اطلاعاتی و امثال آن نداشته باشد.
- شما در کدها و کوئری‌های مثلا EF در اصل با یک سری IQueryable کار می‌کنید. همینجا باید متد الحاقی ToDataSourceResult را اعمال کنید تا نتیجه‌ی نهایی در حداقل بار تعداد رفت و برگشت و با کوئری مناسبی بر اساس پارامترهای دریافتی به صورت خودکار تولید شود. در انتهای کار بجای مثلا ToList بنویسید ToDataSourceResult.

نویسنده: امین
تاریخ: ۱۳۹۳/۰۸/۱۷ ۱۴:۴۱

سلام من در ویو خودم نمیتونم اطلاعاتم رو تو kendo.grid ببینم و برای من یک لیست استرینگ در ویو نمایش داده میشه و به این شکل در کنترلر و ویو کد نویسی کردم .

```
public class EFController : Controller    {
        //
        // GET: /EF/

        public ActionResult AjaxConnected([DataSourceRequest] DataSourceRequest request )
```

```
        {
            using (var dbef=new dbTestEntities())
            {
                IQueryable<Person> persons = dbef.People;
                DataSourceResult result = persons.ToDataSourceResult(request);
                return Json(result.Data,JsonRequestBehavior.AllowGet);
            }
        }


    }
```

و ویو

```
@{    ViewBag.Title = "AjaxConnected";
}


<h2>AjaxConnected</h2>
@(Html.Kendo().Grid<TelerikMvcApp2.Models.Person>(  )
      .Name("Grid")
      .DataSource(builder => builder
          .Ajax()
          .Read(operationBuilder => operationBuilder.Action("AjaxConnected", "EF"))


      )
      .Columns(factory =>
      {
          factory.Bound(person => person.personId);
          factory.Bound(person => person.Name);
          factory.Bound(person => person.LastName);
      })
      .Pageable()
      .Sortable())
```

و یک لیست استرینگ بهم در عمل خروجی میده و از خود قالب kendogrid خبری نیست . من اطلاعات رو به طور json پاس میدم و ajaxi میگیرم.
حالا قبلش همچین خطلایی داشتم که به allowget ایراد میگرفت ولی در کل با JsonRequestBehavior.AllowGet حل شد و حالا فقط یه لسیت بهم خروجی میده! و از ظاهر گرید خبری نیست. و اگر به جای json نوشته بشه view و با ویو return کنم ظاهر kendogrid رو دارم اما خروجی دارای مقداری نیست! اینم خروجی استرینگ من :(

```
[{"personId":1,"Name":"Amin","LastName":"Saadati"},{"personId":2,"Name":"Fariba","LastName":"Ghochani"},{"personId":4,"Name":"Milad","LastName":"Rahmani"},{"personId":5,"Name":"rima","LastName":"rad"},{"personId":6,"Name":"ali","LastName":"kiva"},{"personId":7,"Name":"sahel","LastName":"abasi"},{"personId":8,"Name":"medi","LastName":"ghaem"},{"personId":9,"Name":"mino","LastName":"kafash"},{"personId":10,"Name":"behzad","LastName":"tizro"},{"personId":11,"Name":"toti","LastName":"saadati"},{"personId":12,"Name":"parinaz","LastName":"karami"},{"personId":13,"Name":"sadegh","LastName":"hojati"},{"personId":14,"Name":"milad","LastName":"ebadipor"},{"personId":15,"Name":"farid","LastName":"riazi"},{"personId":16,"Name":"said","LastName":"abdoliar"},{"personId":17,"Name":"behzad","LastName":"ariafotahi"},{"personId":18,"Name":"jamshid","LastName":"k
```

این سوال رو در چند سایت پرسیدم و به جوابی برایش نرسیدم. و نمیدونم ایراد کدهای نوشته شده ام کجاست!
متشکرم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۱۷ ۱۵:۲

- قصد پشتیبانی از wrapperهای آن‌را ندارم. لطفا خارج از موضوع سؤال نپرسید. اگر کسی دوست داشت در این زمینه مطلب منتشر کند، خوب. ولی من چنین قصدی ندارم.
- عرض کردم اگر از wrapperها استفاده کنید، به علت عدم درک زیر ساخت اصلی Kendo UI، قادر به دیباگ کار نخواهید بود.
- اگر متن را مطالعه کنید در قسمت «پیشنیاز تامین داده مخصوص Kendo UI Grid » دقیقا شکل نهایی خروجی JSON مورد نیاز ارائه شده‌است. این خروجی در سه فیلد data، total و aggregate قرار می‌گیرد. شما الان فقط قسمت data آن‌را بازگشت

داده‌اید؛ بجای اصل و کل آن. نام این سه فیلد هم مهم نیست؛ اما هر چیزی که تعیین می‌شوند، باید در قسمت data source در خاصیت schema آن مانند مثالی که در مطلب جاری آمده (در قسمت «تامین داده و نمایش گرید »)، دقیقا مشخص شوند، تا Kendo UI بداند که اطلاعات مختلف را باید از چه فیلدهایی از JSON خروجی دریافت کند.

نویسنده: وحید محمّدطاهری
تاریخ: ۱۳۹۳/۱۰/۰۷ ۱۴:۲۴

با سلام و خدا قوت

آقای نصیری، model ای که باید در قسمت schema تعریف بشه چطوری میشه اونو دینامیک تولید کرد.
من یک چنین حالتی رو ایجاد کردم ولی نمی‌دونم چطوری باید اسم ستونو براش مشخص کنم.

```
public class Field
    {
        public string Type { get; set; }
    }
```

```
public class Fields : System.Collections.ObjectModel.Collection<Field>
    {
        public System.Collections.IEnumerable ToList()
        {
            return System.Linq.Enumerable.ToList( System.Linq.Enumerable.Select( this ,
                                                                                  field => new {
field.Type } ) );
        }
    }
```

این قسمت اطلاعاتی است که برای ایجاد گرید باز گردانده می‌شود.

```
return new InitializeInfo
                  {
                          ...
                          Model = GetModel()
                  };
```

```
private Model GetModel ()
        {
            return new Model{ Fields = GetFields().ToList() };
        }
```

متد GetColumns شامل 3 ستون می‌باشد که نوع، عنوان و سایر مشخصات رو توش تعریف کردم

```
private Fields GetFields()
        {
            var fields = new Fields();
            foreach ( var column in GetColumns() )
            {
                fields.Add( new Field { Type = column.DataType } );
            }
            return fields;
        }
```

الان خروجی که تولید میشه اینجوریه

```
"model": {

    "fields": [
        {
            "type": "string"
```

```
        },
        {
            "type": "string"
        },
        {
            "type": "datetime"
        }
    ]
}
```

ممنون میشم یه راهنمایی کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۰۷ ۱۴:۴۱

- پویا هست و خروجی دسترسی هم گرفتید. زمانیکه تعریف می‌کنید:

```
new Field { Type = column.DataType }
```

یعنی در لیست نهایی، خاصیتی با نام ثابت Type و با مقدار متغیر column.DataType را تولید کن (نام خاصیت، مقدار ثابت نام خاصیت را در JSON نهایی تشکیل می‌دهد).
+ نیازی هم به این همه پیچیدگی نداشت. تمام کارهایی را که انجام دادید با تهیه خروجی ساده List<Field> از یک متد دلخواه، یکی هست و نیازی به anonymous type کار کردن نبود.
- به همان کلاس فیلد، خواص دیگر مورد نیاز را اضافه کنید (عنوان و سایر مشخصات یک فیلد) و در نهایت لیست ساده List<Field> را بازگشت دهید. هر خاصیت کلاس Field، یک ستون گرید را تشکیل می‌دهد.
- همچنین دقت داشته باشید اگر از روش مطلب جاری استفاده می‌کنید، اطلاعات ستون‌های نهایی باید در فیلد Data نهایی قرار گیرند (قسمت «پیشنیاز تامین داده مخصوص Kendo UI Grid» در بحث).

نویسنده: وحید محمّدطاهری
تاریخ: ۱۳۹۳/۱۰/۰۷ ۱۵:۴۸

با تشکر از پاسختون

درسته این به صورت پویا تولید میشه ولی شکل model ای که شما در این مطلب توضیح دادید با این چیزی که کد من تولید می‌کنه فرق میکنه

برای شما اول نام فیلد هست بعد نوع اون فیلد، در حالی که نحوه تولید داینامیک اینو نمی‌دونم چطوری باید باشه.

```
model: {
                    fields: {
                        "Id": { type: "number" }, //تعیین نوع فیلد برای جستجوی پویا مهم است
                        "Name": { type: "string" },
                        "IsAvailable": { type: "boolean" },
                        "Price": { type: "number" }
                    }
                }
      }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۰۷ ۱۶:۱۹

باید از Dictionary استفاده کنید برای تعریف خواص پویا:

```
public class Field
{
  [JsonExtensionData]
  public Dictionary<string, object> Property { get; set; }
```

```
}

public class FieldType
{
  public string Type { get; set; }
}
```

و بعد نحوه استفاده از آن به صورت زیر خواهد بود:

```
var data = new
{
  model = new
  {
   fields = new List<Field>
   {
    new Field
    {
      Property = new Dictionary<string, object>
      {
       {"Id", new FieldType { Type = "number" }},
       {"Name", new FieldType { Type = "string" }}
      }
    }
   }
  }
};
var dataJson = JsonConvert.SerializeObject(data, Formatting.Indented);
```

با این خروجی:

```
{
  "model": {
    "fields": [
      {
        "Id": {
          "Type": "number"
        },
        "Name": {
          "Type": "string"
        }
      }
    ]
  }
}
```

- اگر از Web API استفاده می‌کنید، ذکر سطر JsonConvert.SerializeObject ضروری نیست و به صورت توکار از JSON.NET استفاده می‌کند.
- اگر از ASP.NET MVC استفاده می‌کنید، نیاز است از آن کمک بگیرید . از این جهت که خاصیت JsonExtensionData سبب می‌شود تا نام ثابت خاصیت Property، از خروجی نهایی حذف شود و اعضای دیکشنری، جزئی از خاصیت‌های موجود شوند.
- نکته‌ی « گرفتن خروجی CamelCase از JSON.NET » را هم باید مد نظر داشته باشید.