عنوان: ذخیره TreeView ساخته شده توسط KendoUI در Asp.net MVC
نویسنده: حسین حسینی 128
تاریخ: ۱۳۹۲/۰۸/۱۳ ۱۲:۵
آدرس: www.dotnettips.info
گروه‌ها: MVC, jQuery, Kendo UI

همانطور که از نمونه مثال‌های خود Kendo UI مشاهده میشود ، نحوه استفاده از TreeView آن به صورت زیر است :

```
<div>
@(Html.Kendo().TreeView()
        .Name("treeview")
        .TemplateId("treeview-template")
        .HtmlAttributes(new { @class = "demo-section" })
        .DragAndDrop(true)
        .BindTo(Model.Where(e=>e.ParentFolderID==null).OrderBy(e=>e.Order), mappings =>
        {
            mappings.For<DAL.Folder>(binding => binding
                    .ItemDataBound((item, folder) =>
                    {
                        item.Text = folder.FolderName;
                        item.SpriteCssClasses = "folder";
                        item.Expanded=true;
                        item.Id = folder.FolderID.ToString();
                    })
                    .Children(folder => folder.Folder1));
            mappings.For<DAL.Folder>(binding => binding
                    .ItemDataBound((item, folder) =>
                    {
                        item.Text = folder.FolderName;
                        item.SpriteCssClasses = " folder";
                        item.Expanded = true;
                        item.Id = folder.FolderID.ToString();
                    }));
        })
)
</div>
```

```
<style type="text/css" scoped>
    .demo-section {
        width: 200px;
    }

    #treeview  .k-sprite ,#treeview2 .k-sprite {
        background-image: url("@Url.Content("/Content/kendo/images/coloricons-sprite.png")");
    }
    .rootfolder { background-position: 0 0; }
    .folder { background-position: 0 -16px; }
    .pdf { background-position: 0 -32px; }
    .html { background-position: 0 -48px; }
    .image { background-position: 0 -64px; }
    .delete-link,.edit-link {
        width: 12px;
        height: 12px;
        overflow: hidden;
        display: inline-block;
        vertical-align: top;
        margin: 2px 0 0 3px;
        -webkit-border-radius: 5px;
        -mox-border-radius: 5px;
        border-radius: 5px;
    }
    .delete-link{
        background: transparent url("@Url.Content("/Content/kendo/images/close.png")") no-repeat 50%
50%;
    }
    .edit-link{
        background: transparent url("@Url.Content("/Content/kendo/images/edit.png")") no-repeat 50%
50%;
    }
</style>
```

استفاده از این TreeView ساده است ولی اگر احتیاج داشته باشیم که پس از drag&drop کردن گره‌ها آن را ذخیره کنیم چگونه باید عمل کنیم؟ این ریالسمت در خود Kendo تعبیه نشده است ، پس به صورت زیر عمل میکنیم :

پس از ساختن TreeView و اصلاح آن به شکل دلخواه لینک زیر را در ادامه اش می‌آوریم :

```
<a href="#" id="serialize">ذخیره</a>
```

سپس در تگ اسکریپت‌های خود این کد جاوا اسکریپت را برای serialize کردن تمام گره‌های TreeView مینویسیم :

```
$('#serialize').click(function () {
            serialized = serialize();
            window.location.href = "Folder/SaveMenu?serial=" + serialized + "!";
        });

        function serialize() {
            var tree = $("#treeview").data("kendoTreeView");
            var json = treeToJson(tree.dataSource.view());
            return JSON.stringify(json);
        }

        function treeToJson(nodes) {

            return $.map(nodes, function (n, i) {

                var result = { id: n.id};
                //var result = { text: n.text, id: n.id, expanded: n.expanded, checked: n.checked };
                if (n.hasChildren)
                    result.items = treeToJson(n.children.view());

                return result;
            });
        }
```

حال به تشریح کدها میپردازیم :

تابع serialize درخط اول تمام عناصر داخلی treeview را گرفته ، به صورت یک datasource قابل انقیاد درآورده وسپس datasource آن را به یک نمایش قابل تجزیه تبدیل میکند و به متد treeToJSON میفرستد.

```
var tree = $("#treeview").data("kendoTreeView");
var json = treeToJson(tree.dataSource.view());
```

تابع treeToJSON درخت را به عنوان یکسری گره گرفته و تمام عناصر آن را به فرمت json میبرد . (قسمتی که به صورت توضیحی درآمده میتواند برای بدست آوردن تمام اطلاعات گره از جمله متن و انتخاب شدن checkbox آن و غیره مورد استفاده قرار بگیرد) در ادامه این تابع اگر گره درحال استفاده فرزندی داشته باشد به صورت بازگشتی همین تابع برای آن فراخوانده میشود.

```
var result = { id: n.id};
 //var result = { text: n.text, id: n.id, expanded: n.expanded, checked: n.checked };
 if (n.hasChildren)
 result.items = treeToJson(n.children.view());
```

سپس اطلاعات برگشتی که در فرمت json هستند در خط آخر serialaize به رشته تبدیل میشوند و به رویداد کلیک که از آن فراخوانده شده بود بازمیگردند.

```
return JSON.stringify(json);
```

خط آخر رویداد نیز یک Action در Controller مورد نظر را هدف قرار میدهد و رشته بدست آمده را به آن ارسال میکند و صفحه redirect میشود.

```
window.location.href = "Folder/SaveMenu?serial=" + serialized + "!";
```

رشته ای که با عنوان serialize به کنترلر ارسال میشود مانند زیر است :

```
"[{\"id\":\"2\"},{\"id\":\"5\",\"items\":[{\"id\":\"3\"},{\"id\":\"6\"},{\"id\":\"7\"}]}]!"
```

این رشته مربوط به درختی به شکل زیر است :

همانطور که میبینید گره دوم که "پوشه چهارم45" نام دارد شامل سه فرزند است که در رشته داده شده با عنوان item شناخته شده است. حال باید این رشته با برنامه نویسی سی شارپ جداسازی کرد :

```
string serialized;
Dictionary<int, int> numbers = new Dictionary<int, int>();
```

```
public ActionResult SaveMenu(string serial)
        {
            var newfolders = new List<Folder>();
            serialized = serial;
            calculte_serialized(0);
            return RedirectToAction("Index");
        }
```

```
void calculte_serialized(int parent)
        {
            while (serialized.Length > 0)
            {
                var id_index=serialized.IndexOf("id");
                if (id_index == -1)
                {
                    return;
                }
                serialized = serialized.Substring(id_index + 5);
                var quote_index = serialized.IndexOf("\"");
                var id=serialized.Substring(0, quote_index);
                numbers.Add(int.Parse(id), parent);
                serialized = serialized.Substring(quote_index);
                var condition = serialized.Substring(0,3);
                switch (condition)
                {
                    case "\"},":
                        break;
                    case "\",\"":
                        calculte_serialized(int.Parse(id));
                        break;
                    case "\"}]":
                        return;
                        break;
                    default:
                        break;
                }
            }
        }
```

calculte_serialized با گرفتن 0 کار خود را شروع میکند یعنی از گره هایی که پدر ندارند و تمام idها را همراه با پدرشان در یک دیکشنری میریزد و هرکجا که به فرزندی برخورد به صورت بازگشتی فراخوانی میشود. پس از اجرای کامل آن ما درخت را در یک دیکشنری به صورت عنصرهای مجزا در اختیار داریم که میتوانیم در پایگاه داده ذخیره کنیم.

نظرات خوانندگان

نویسنده: امیر بختیاری
تاریخ: ۱۳۹۲/۱۱/۲۵ ۸:۳۱

با سلام و تشکر از شما
اگر tree دارای چک باکس باشد و بخواهیم نود هایی که چک خورده است را ذخیره کنیم چگونه عمل کنیم؟
قابلیت drag هم نداشت مهم نیست . یک tree ثابت از دیتا بیس پر می‌شود و بعد گزینه هایی که تیک دارد را در جدولی دیگر ذخیره می‌کنیم

عنوان: **یک تکنیک جالب در نحوه نام گذاری فیلدهای دیتابیس به منظور استفاده بهینه از فایل های T4 در MVC5**
نویسنده: صادق نجاتی
تاریخ: ۱۳۹۳/۰۲/۱۰ ۱۰:۴۰
آدرس: www.dotnettips.info
گروه‌ها: MVC, Kendo UI, T4

بدون شک دوستانی که با تکنولوژی محبوب ASP.NET MVC5 کار کرده اند این نکته را می‌دانند که اگر فایل‌های T4 که وظیفه Scaffolding را به عهده دارند به پروژه خود اضافه کنند می‌توانند نحوه تولید خودکار Controllerها و Viewهای متناظر را سفارشی کنند. مثلا می‌توان این فایل‌ها را طوری طراحی کرد که Controller و Viewهای تولیدی به طور اتوماتیک چند زبانه و یا Responsive تولید شوند (این موضوعات بحث اصلی مقاله نیستند) و اما بحث اصلی را با یک مثال آغاز می‌کنیم :
فرض کنید در دیتابیس خود یک Table دارید که قرار است اطلاعات یک Slider را در خود نگه دارد. این Table دارای یک فیلد از نوع nvarchar برای ذخیره آدرس تصویر ارسالی توسط کاربر است.
در حالت عادی اگر از روی مدل این Table اقدام به تولید خودکار Controller و View متناظر کنید، یک editor (تکست باکس) برای دریافت آدرس تصویر تولید خواهد شد که برنامه نویس یا طراح باید به طور دستی آن را (به طور مثال) با Kendo uploader جایگزین نماید. ما می‌خواهیم برای فیلدهایی که قرار است آدرس تصویر را در خود نگه دارد به طور اتوماتیک از Kendo uploader استفاده شود. راه حل چیست؟
بسیار ساده است. ابتدا باید در نظر داشت که هنگام طراحی Table در دیتا بیس فیلد مورد نظر را به این شکل نامگذاری کنید : ExampleIMGURL (نحوه نام گذاری دلخواه است) مقصود آن است که نام هر فیلدی که قرار است آدرس یک تصویر را در خود نگه دارد باید حاوی کلمه (IMGURL) باشد. مجددا ذکر می‌شود که نحوه نامگذاری اختیاری است. سپس فایل Create.t4 را باز کنید و کد :

```
@Html.EditorFor(model => model.<#= property.PropertyName #>)
```

را با کد زیر جایگزین کنید :

```
<#
if (GetAssociationName(property).Contains ("IMGURL")
{
#>
     @Html.Kendo().Upload().Name("<#= property.PropertyName #>")
<#
}
else
{
#>
     @Html.EditorFor(model => model.<#= property.PropertyName #>)
<#
}
#>
```

کد بالا چک می‌کند اگر نام فیلد مد نظر حاوی " IMGURL " باشد یک کندو آپلودر تولید کرده در غیر این صورت یک ادیتور ساده تولید می‌کند. البته این فقط یک مثال است و بدون شک دامنه استفاده از این تکنیک وسیع‌تر است.

اگر این مطلب مفید واقع شد با در نظر گرفتن نظرات ارسالی به تکنیک‌های آتی اشاره خواهد شد.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۱۰ ۱۱:۳۰

قابلیت سفارشی سازی EditorFor در ASP.NET MVC پیش بینی شده‌است و با استفاده از UIHint قابل انتساب به خواص مدل مورد نظر است. البته این مورد برای حالت Code first یا حالتیکه از ViewModels استفاده کنید بیشتر کاربرد دارد.
یک مثال:
فایلی را به نام Upload.cshtml ، در مسیر Views/Shared/EditorTemplates با محتوای ذیل ایجاد کنید:

```
@model string
@Html.Kendo().Upload().Name("@ViewData.ModelMetadata.PropertyName")
```

سپس برای استفاده از آن فقط کافی است خاصیت مدنظر را با ویژگی UIHint مزین کنید:

```
[UIHint("Upload")]
public string ImageUrl {set;get;}
```

نویسنده: صادق نجاتی
تاریخ: ۱۳۹۳/۰۲/۲۹ ۱۱:۵

ضمن تشکر از آقای نصیری؛
بدون شک نقش UIHint در سفارشی سازی انکار ناپذیر است. ولی همانطور که گفته شد دامنه استفاده از این تکنیک وسیع‌تر است. مثلا حالتی را در نظر بگیرید که می‌خواهیم از طریق Scaffolding برای یک جدول بانک اطلاعاتی که یک فیلد آن آدرس یک تصویر را نگهداری می‌کند View ایجاد نماییم. خوب ما در صفحه Index می‌خواهیم تصویر مورد نظر با اندازه 100 * 100 پیکسل نمایش دهیم (چون قرار است لیستی از تصاویر نمایش داده شود باید در اندازه قابل نمایشی باشد) ولی در صفحه Details باید اندازه بزرگتری از تصویر را به نمایش بگذاریم. حال اگر از UIHint استفاده کنیم تنها یکی از موارد قبل (سفارشی سازی در لیست و جزئیات) محقق خواهد شد. اگر بخواهیم انجام این کارها را به صورت اتوماتیک به Scaffolding بسپاریم باید مطابق آنچه گفته شد ، فایل‌های T4 را (List.t4 و Details.t4) سفارشی سازی نماییم.

عنوان: استفاده ازExpressionها جهت ایجاد Strongly typed view در ASP.NET MVC
نویسنده: محسن موسوی
تاریخ: ۱۳۹۳/۰۴/۱۵ ۰:۰
آدرس: www.dotnettips.info
گروه‌ها: MVC, Reflection, Kendo UI, jqGrid

مدل زیر را در نظر بگیرید:

```
/// <summary>
    ///
    /// </summary>
    public class CompanyModel
    {
        /// <summary>
        /// Table Identity
        /// </summary>
        public int Id { get; set; }

        /// <summary>
        /// Company Name
        /// </summary>
        [DisplayName("نام شرکت")]
        public string CompanyName { get; set; }

        /// <summary>
        /// Company Abbreviation
        /// </summary>
        [DisplayName("نام اختصاری شرکت")]
        public string CompanyAbbr { get; set; }

    }
```

از View زیر جهت نمایش لیستی از شرکت‌ها متناظر با مدل جاری استفاده میشود:

```
@{
    const string viewTitle = "شرکت ها";
    ViewBag.Title = viewTitle;
    const string gridName = "companies-grid";
}
<div class="col-md-12">
    <div class="form-panel">
        <header>
            <div class="title">
                <i class="fa fa-book"></i>
                @viewTitle
            </div>
        </header>
        <div class="panel-body">
            <div id="@gridName">
            </div>
        </div>
    </div>
</div>
</div>
@section scripts
{
    <script type="text/javascript">
        $(document).ready(function () {
            $("#@gridName").kendoGrid({
                dataSource: {
                    type: "json",
                    transport: {
                        read: {
                            url: "@Html.Raw(Url.Action(MVC.Company.CompanyList()))",
                            type: "POST",
                            dataType: "json",
                            contentType: "application/json"
                        }
                    },
                    schema: {
                        data: "Data",
                        total: "Total",
                        errors: "Errors"
                        }
                    },
                    pageSize: 10,
```

```
                    serverPaging: true,
                    serverFiltering: true,
                    serverSorting: true
                },
                pageable: {
                    refresh: true
                },
                sortable: {
                    mode: "multiple",
                    allowUnsort: true
                },
                editable: false,
                filterable: false,
                scrollable: false,
                columns: [ {
                    field: "CompanyName",
                    title: "نام شرکت",
                    sortable: true,
                }, {
                    field: "CompanyAbbr",
                    title: "مخفف نام شرکت",
                    sortable: true
                }]
            });
        });
    </script>
}
```

مشکلی که در کد بالا وجود دارد این است که با تغییر نام هر یک از متغییر هایمان ، اطلاعات گرید در ستون مربوطه نمایش داده نمیشود.همچنین عناوین ستونها نیز از DisplayName مدل پیروی نمیکنند.توسط متدهای الحاقی زیر این مشکل برطرف شده است.

```
/// <summary>
    ///
    /// </summary>
    public static class PropertyExtensions
    {
        /// <summary>
        ///
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <param name="expression"></param>
        /// <returns></returns>
        public static MemberInfo GetMember<T>(this Expression<Func<T, object>> expression)
        {
            var mbody = expression.Body as MemberExpression;

            if (mbody != null) return mbody.Member;
            //This will handle Nullable<T> properties.
            var ubody = expression.Body as UnaryExpression;
            if (ubody != null)
            {
                mbody = ubody.Operand as MemberExpression;
            }
            if (mbody == null)
            {
                throw new ArgumentException("Expression is not a MemberExpression", "expression");
            }
            return mbody.Member;
        }

        /// <summary>
        ///
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <param name="expression"></param>
        /// <returns></returns>
        public static string PropertyName<T>(this Expression<Func<T, object>> expression)
        {
            return GetMember(expression).Name;
        }

        /// <summary>
        ///
        /// </summary>
```

```
        /// <typeparam name="T"></typeparam>
        /// <param name="expression"></param>
        /// <returns></returns>
        public static string PropertyDisplay<T>(this Expression<Func<T, object>> expression)
        {
            var propertyMember = GetMember(expression);
            var displayAttributes = propertyMember.GetCustomAttributes(typeof(DisplayNameAttribute),
true);
            return displayAttributes.Length == 1 ?
((DisplayNameAttribute)displayAttributes[0]).DisplayName : propertyMember.Name;
        }
    }
```

```
public static string PropertyName<T>(this Expression<Func<T, object>> expression)
```

جهت بدست آوردن نام متغییر هایمان استفاده مینماییم.

```
public static string PropertyDisplay<T>(this Expression<Func<T, object>> expression)
```

جهت بدست آوردن DisplayNameAttribute استفده میشود. درصورتیکه این DisplayNameAttribute یافت نشود نام متغییر بازگشت داده میشود.

بنابراین View مربوطه را اینگونه بازنویسی میکنیم:

```
@using Models
@{
    const string viewTitle = "شرکت ها";
    ViewBag.Title = viewTitle;
    const string gridName = "companies-grid";
}
<div class="col-md-12">
    <div class="form-panel">
        <header>
            <div class="title">
                <i class="fa fa-book"></i>
                @viewTitle
            </div>
        </header>
        <div class="panel-body">
            <div id="@gridName">
            </div>
        </div>
    </div>
</div>
</div>
@section scripts
{
    <script type="text/javascript">
        $(document).ready(function () {
            $("#@gridName").kendoGrid({
                dataSource: {
                    type: "json",
                    transport: {
                        read: {
                            url: "@Html.Raw(Url.Action(MVC.Company.CompanyList()))",
                            type: "POST",
                            dataType: "json",
                            contentType: "application/json"
                        }
                    },
                    schema: {
                        data: "Data",
                        total: "Total",
                        errors: "Errors"
                        }
```

```
                    },
                    pageSize: 10,
                    serverPaging: true,
                    serverFiltering: true,
                    serverSorting: true
                },
                pageable: {
                    refresh: true
                },
                sortable: {
                    mode: "multiple",
                    allowUnsort: true
                },
                editable: false,
                filterable: false,
                scrollable: false,
                columns: [ {
                    field: "@(PropertyExtensions.PropertyName<CompanyModel>(a => a.CompanyName))",
                    title: "@(PropertyExtensions.PropertyDisplay<CompanyModel>(a => a.CompanyName))",
                    sortable: true,
                }, {
                    field: "@(PropertyExtensions.PropertyName<CompanyModel>(a => a.CompanyAbbr))",
                    title: "@(PropertyExtensions.PropertyDisplay<CompanyModel>(a => a.CompanyAbbr))",
                    sortable: true
                }]
            });
        });
    </script>
}
```

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۴/۱۶ ۱۲:۳۸

با تشکر از شما. حالت پیشرفته‌تر این مساله، کار با مدل‌های تو در تو هست. برای مثال:

```
public class CompanyModel
    {
        public int Id { get; set; }
        public string CompanyName { get; set; }
        public string CompanyAbbr { get; set; }

        public Product Product { set; get; }
    }

    public class Product
    {
        public int Id { set; get; }
    }
```

در اینجا اگر بخواهیم Product.Id را بررسی کنیم:

```
var data = PropertyExtensions.PropertyName<CompanyModel>(x => x.Product.Id);
```

فقط Id آن دریافت می‌شود.
راه حلی که از کدهای EF برای این مساله استخراج شده به صورت زیر است (نمونه‌اش متد Include تو در تو بر روی چند خاصیت):

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

namespace PropertyExtensionsApp
{
    public class PropertyHelper : ExpressionVisitor
    {
        private Stack<string> _stack;
        public string GetNestedPropertyPath(Expression expression)
        {
            _stack = new Stack<string>();
            Visit(expression);
            return _stack.Aggregate((s1, s2) => s1 + "." + s2);
        }

        protected override Expression VisitMember(MemberExpression expression)
        {
            if (_stack != null)
                _stack.Push(expression.Member.Name);
            return base.VisitMember(expression);
        }

        public string GetNestedPropertyName<TEntity>(Expression<Func<TEntity, object>> expression)
        {
            return GetNestedPropertyPath(expression);
        }
    }
}
```

در این حالت خواهیم داشت:

```
var name = new PropertyHelper().GetNestedPropertyName<CompanyModel>(x => x.Product.Id);
```

که خروجی Product.Id را بر می‌گرداند.

نویسنده: محسن موسوی
تاریخ: ۱۳۹۳/۰۷/۲۶ ۱۸:۸

در نهایت این متد به این شکل اصلاح شود:

```
/// <summary>
        ///
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <param name="expression"></param>
        /// <returns></returns>
        public static string PropertyName<T>(this Expression<Func<T, object>> expression)
        {
            return new PropertyHelper().GetNestedPropertyName(expression);
        }
```