عنوان: **آپلود فایل توسط فرم‌های پویای jqGrid**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۴/۲۴ ۱۱:۳۵
آدرس: www.dotnettips.info
گروه‌ها: MVC, jqGrid

پیشنیازها

Ajax.BeginForm و ارسال فایل به سرور در ASP.NET MVC
فعال سازی و پردازش صفحات پویای افزودن، ویرایش و حذف رکوردهای jqGrid در ASP.NET MVC
فرمت کردن اطلاعات نمایش داده شده به کمک jqGrid در ASP.NET MVC
استفاده ازExpressionها جهت ایجاد Strongly typed view در ASP.NET MVC

فرم‌های پویای jqGrid نیز به صورت Ajax ایی به سرور ارسال می‌شوند و اگر نوع عناصر تشکیل دهنده‌ی آن‌ها file تعیین شوند، قادر به ارسال این فایل‌ها به سرور نخواهند بود. در ادامه نحوه‌ی یکپارچه سازی افزونه‌ی AjaxFileUpload را با فرم‌های jqGrid بررسی خواهیم کرد.

تغییرات فایل Layout برنامه

در اینجا دو فایل جدید ajaxfileupload.js و jquery.blockUI.js به مجموعه‌ی فایل‌های تعریف شده اضافه شده‌اند:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET Application</title>

    <link href="~/Content/themes/base/jquery.ui.all.css" rel="stylesheet" />
    <link href="~/Content/jquery.jqGrid/ui.jqgrid.css" rel="stylesheet" />
    <link href="~/Content/Site.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <div>
        @RenderBody()
    </div>

    <script src="~/Scripts/jquery-1.7.2.min.js"></script>
    <script src="~/Scripts/jquery-ui-1.8.11.min.js"></script>
    <script src="~/Scripts/i18n/grid.locale-fa.js"></script>
    <script src="~/Scripts/jquery.jqGrid.src.js"></script>
    <script src="~/Scripts/ajaxfileupload.js"></script>
    <script src="~/Scripts/jquery.blockUI.js"></script>

    @RenderSection("Scripts", required: false)
</body>
</html>
```

از فایل jquery.blockUI.js برای نمایش صفحه‌ی منتظر بمانید تا فایل آپلود شود، استفاده خواهیم کرد.

```
PM> Install-Package jQuery.BlockUI
```

نکته‌ای در مورد واکنشگرا کردن jqGrid

اگر می‌خواهید عرض jqGrid به تغییرات اندازه‌ی مرورگر پاسخ دهد، تنها کافی است تغییرات ذیل را اعمال کنید:

```
<div dir="rtl" id="grid1" style="width:100%;" align="center">
    <div id="rsperror"></div>
    <table id="list" cellpadding="0" cellspacing="0"></table>
    <div id="pager" style="text-align:center;"></div>
</div>
```

```
    <script type="text/javascript">
        $(document).ready(function () {

            // Responsive jqGrid
            $(window).bind('resize', function () {
                var targetContainer = "#grid1";
                var targetGrid = "#list";

                $(targetGrid).setGridWidth($(targetContainer).width() - 2, true);
            }).trigger('resize');


            $('#list').jqGrid({
                caption: "آزمایش هفتم",
                /// .....
            }).navGrid(
                /// .....
            ).jqGrid('gridResize', { minWidth: 400, minHeight: 150 });
        });
    </script>
```

در اینجا به تغییرات resize صفحه گوش فرا داده شده و سپس به کمک متد توکار setGridWidth، به صورت پویا اندازه‌ی عرض jqGrid تغییر خواهد کرد.
همچنین اگر می‌خواهید کاربر بتواند اندازه‌ی گرید را دستی تغییر دهد، به انتهای تعاریف گرید، تعریف متد gridResize را نیز اضافه کنید.

نحوه‌ی تعریف ستونی که قرار است فایل آپلود کند

```
colModel: [
                    {
                        name: '@(StronglyTyped.PropertyName<Product>(x=>x.ImageName))',
                        index: '@(StronglyTyped.PropertyName<Product>(x => x.ImageName))',
                        align: 'center', width: 220,
                        editable: true,
                        edittype: 'file',
                        formatter: function (cellvalue, options, rowObject) {
                            return "<img src='/images/" + cellvalue + "?rnd=" + new Date().getTime() +
"' />";
                        },
                        unformat: function (cellvalue, options, cell) {
                            return $('img', cell).attr('src').replace('/images/', '');
                        }
                    }
                ],
```

edittype ستونی که قرار است فایل آپلود کند، باید به file تنظیم شود. همچنین چون در اینجا این فایل آپلودی، تصویر یک محصول است، از formatter برای تبدیل مسیر فایل به تصویر و از unformat برای بازگشت این مسیر به مقدر اصلی آن استفاده خواهیم کرد. از unformat برای حالت ویرایش اطلاعات استفاده می‌شود. از formatter برای تغییر اطلاعات دریافتی از سرور به فرمت دلخواهی در سمت کلاینت می‌توان کمک گرفت.
Rnd اضافه شده به انتهای آدرس تصویر، جهت جلوگیری از کش شدن آن تعریف شده‌است.

کتابخانه‌ی JqGridHelper

در قسمت‌های قبل مطالب بررسی jqGrid یک سری کلاس مانند JqGridData برای بازگشت اطلاعات مخصوص jqGrid و یا JqGridRequest برای دریافت پارامترهای ارسالی توسط آن به سرور، تهیه کردیم؛ به همراه کلاس‌هایی مانند جستجو و مرتب سازی پویای اطلاعات.
اگر این کلاس‌ها را از پروژه‌ها و مثال‌های ارائه شده خارج کنیم، می‌توان به کتابخانه‌ی JqGridHelper رسید که فایل‌های آن در پروژه‌ی پیوست موجود هستند.

همچنین در این پروژه، کلاسی به نام StronglyTyped با متد PropertyName جهت دریافت نام رشته‌ای یک خاصیت تعریف شده‌است. گاهی از اوقات این تنها چیزی است که کدهای سمت کلاینت، جهت سازگار شدن با Refactoring و Strongly typed تعریف شدن نیاز دارند و نه ... محصور کننده‌هایی طویل و عریض که هیچگاه نمی‌توانند تمام قابلیت‌های یک کتابخانه‌ی غنی جاوا اسکریپتی را به همراه داشته باشند.
با کمی جستجو، برای jqGrid نیز می‌توانید از این دست محصور کننده‌هارا پیدا کنید اما ... هیچکدام کامل نیستند و دست آخر مجبور خواهید شد در بسیاری از موارد مستقیما JavaScript نویسی کنید.

یکپارچه سازی افزونه‌ی AjaxFileUpload با فرم‌های jqGrid

پس از این مقدمات، ستون ویژه‌ی actions که inline edit را فعال می‌کند، چنین تعریفی را پیدا خواهد کرد:

```
colModel: [
                    {
                        name: 'myac', width: 80, fixed: true, sortable: false,
                        resize: false, formatter: 'actions',
                        formatoptions: {
                            keys: true,
                            afterSave: function (rowid, response) {
                                doInlineUpload(response, rowid);
                            },
                            delbutton: true,
                            delOptions: {
                                url: "@Url.Action("DeleteProduct","Home")"
                            }
                        }
                    }
                ],
```

در اینجا afterSave اضافه شده‌است تا کار ارسال فایل به سرور را در حالت ویرایش inline فعال کند.
و ویژگی‌های قسمت‌های add، edit و delete فرم‌های پویای jqGrid باید به نحو ذیل تغییر کنند:

```
$('#list').jqGrid({
                caption: "آزمایش هفتم",
                // ....
            }).navGrid(
                '#pager',
                //enabling buttons
                { add: true, del: true, edit: true, search: false },
                //edit option
                {
                    width: 'auto',
                    reloadAfterSubmit: true, checkOnUpdate: true, checkOnSubmit: true,
                    beforeShowForm: function (form) {
                        centerDialog(form, $('#list'));
                    },
                    afterSubmit: doFormUpload,
                    closeAfterEdit: true
                },
                //add options
                {
                    width: 'auto', url: '@Url.Action("AddProduct","Home")',
                    reloadAfterSubmit: true, checkOnUpdate: true, checkOnSubmit: true,
                    beforeShowForm: function (form) {
                        centerDialog(form, $('#list'));
                    },
                    afterSubmit: doFormUpload,
                    closeAfterAdd: true
                },
                //delete options
                {
                    url: '@Url.Action("DeleteProduct","Home")',
                    reloadAfterSubmit: true
                }).jqGrid('gridResize', { minWidth: 400, minHeight: 150 });
```

با اکثر این تنظیمات در مطلب « فعال سازی و پردازش صفحات پویای افزودن، ویرایش و حذف رکوردهای jqGrid در ASP.NET MVC » آشنا شده‌اید. تنها قسمت جدید آن شامل رویدادگردان afterSubmit است. در اینجا است که افزونه‌ی AjaxFileUpload

فعال شده و سپس اطلاعات المان فایل را به سرور ارسال می‌کند.
افزونه‌ی AjaxFileUpload پس از ارسال اطلاعات عناصر غیر فایلی فرم، باید فعال شود. به همین جهت است که از رویداد afterSubmit در حالت نمایش فرم‌های پویا و رویداد afterSave در حالت ویرایش inline استفاده کرده‌ایم.
در ادامه تعاریف متدهای doInlineUpload و doUpload بکار گرفته شده در رویداد afterSubmit را مشاهده می‌کنید:

```
function doInlineUpload(response, rowId) {
            return doUpload(response, null, rowId);
        }

        function doFormUpload(response, postdata) {
            return doUpload(response, postdata, null);
        }

        function doUpload(response, postdata, rowId) {
            // دریافت خروجی متد ثبت اطلاعات از سرور
            // و استفاده از آی دی رکورد ثبت شده برای انتساب فایل آپلودی به آن رکورد
            var result = $.parseJSON(response.responseText);
            if (result.success === false)
                return [false, "عملیات ثبت موفقیت آمیز نبود", result.id];

            var fileElementId = '@(StronglyTyped.PropertyName<Product>(x=>x.ImageName))';
            if (rowId) {
                fileElementId = rowId + "_" + fileElementId;
            }

            var val = $("#" + fileElementId).val();
            if (val == '' || val === undefined) {
                // فایلی انتخاب نشده
                return [false, "لطفا فایلی را انتخاب کنید", result.id];
            }

            $('#grid1').block({ message: '<h4>در حال ارسال فایل به سرور</h4>' });
            $.ajaxFileUpload({
                url: "@Url.Action("UploadFiles", "Home")", // مسیری که باید فایل به آن ارسال شود
                secureuri: false,
                fileElementId: fileElementId, // آی دی المان ورودی فایل
                dataType: 'json',
                data: { id: result.id }, // اطلاعات اضافی در صورت نیاز
                complete: function () {
                    $('#grid1').unblock();
                },
                success: function (data, status) {
                    $("#list").trigger("reloadGrid");
                },
                error: function (data, status, e) {
                    alert(e);
                }
            });

            return [true, "با تشکر!", result.id];
        }
```

امضای رویدادگردان‌های afterSubmit و afterSave یکی نیست. به همین جهت دو متد اضافی به جای یک متد doUpload مورد استفاده قرار گرفته‌اند.
متد doUpload توسط پارامتر response، اطلاعات بازگشتی پس از ذخیره سازی متداول اطلاعات فرم را دریافت می‌کند. برای مثال ابتدا اطلاعات معمولی یک محصول در بانک اطلاعاتی ذخیره شده و سپس id آن به همراه یک خاصیت به نام success از طرف سرور بازگشت داده می‌شوند.
اگر success مساوی true بود، ادامه‌ی کار آپلود فایل انجام خواهد شد. در اینجا ابتدا بررسی می‌شود که آیا فایلی از طرف کاربر انتخاب شده‌است یا خیر؟ اگر خیر، یک پیام اعتبارسنجی سفارشی به او نمایش داده خواهد شد.
خروجی متد doUpload حتما باید به شکل یک آرایه سه عضوی باشد. عضو اول آن true و false است؛ به معنای موفقیت یا عدم موفقیت عملیات. عضو دوم پیام اعتبارسنجی سفارشی است و عضو سوم، Id ردیف.
در ادامه افزونه‌ی jQuery.BlockUI فعال می‌شود تا ارسال فایل به سرور را به کاربر گوشزد کند.
سپس فراخوانی متداول افزونه‌ی ajaxFileUpload را مشاهده می‌کنید. تنها نکته‌ی مهم آن فراخوانی متد reloadGrid در حالت success است. به این ترتیب گرید را وادار می‌کنیم تا اطلاعات ذخیره شده در سمت سرور را دریافت کرده و سپس تصویر را به نحو صحیحی نمایش دهد.

کدهای سمت سرور آپلود فایل

```
[HttpPost]
        public ActionResult AddProduct(Product postData)
        {
            // ...
            return Json(new { id = postData.Id, success = true }, JsonRequestBehavior.AllowGet);
        }

        [HttpPost]
        public ActionResult EditProduct(Product postData)
        {
            // ...
            return Json(new { id = postData.Id, success = true }, JsonRequestBehavior.AllowGet);
        }


        // todo: change `imageName` according to the form's file element name
        [HttpPost]
        public ActionResult UploadFiles(HttpPostedFileBase imageName, int id)
        {
            // ....
            return Json(new { FileName = product.ImageName }, "text/html",
JsonRequestBehavior.AllowGet);
        }
```

در اینجا تنها نکته‌ی مهم، خروجی‌های JSON این متدها هستند.
در حالت‌های Add و Edit، نیاز است id رکورد ثبت شده بازگشت داده شود. این id در سمت کلاینت توسط پارامتر response دریافت می‌شود. از آن در افزونه‌ی ارسال فایل به سرور استفاده خواهیم کرد. اگر به متد UploadFiles دقت کنید، این id را دریافت می‌کند. بنابراین می‌توان یک ربط منطقی را بین فایل ارسالی و رکورد متناظر با آن برقرار کرد.
Content type مقدار بازگشتی از متد UploadFiles حتما باید text/html باشد (افزونه‌ی ارسال فایل‌ها، اینگونه کار می‌کند).

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید:

jqGrid07.zip

نظرات خوانندگان

نویسنده: daniyal
تاریخ: ۱۳۹۳/۰۵/۰۴ ۲۲:۱۱

سلام
strongly typeها فقط نام پروپرتی‌ها را بر می‌گرداند اگر بخواهیم مقدار یکی از data annotationها مثل display برگرداند و یا اگر بخواهیم از validationهای data annotationها استفاده کنیم باید چه کار کنیم .ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۵/۰۵ ۱:۱۰

- به پیشنیازهای بحث مراجعه کنید؛ مورد آخر .
- مستقل هست از مباحث اعتبارسنجی سمت کاربر تعریف شده توسط data annotationها. از افزونه‌ی jQuery Validator استفاده نمی‌کند. سیستم خاص خودش را دارد. فقط از اعتبارسنجی سمت سرور حاصل data annotationها می‌شود در آن استفاده کرد .